Rules and Maps II: Recent Progress in Connectionist Symbol Processing

David S. Touretzky[1]
Deirdre W. Wheeler[2]
Gillette Elvgren III[1]

March 1990

CMU-CS-90-112

[1]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[2]Department of Linguistics
University of Pittsburgh
Pittsburgh, PA 15260

"A Computational Basis for Phonology" and "Rule Representations in a Connectionist Chunker" will appear in D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, the collected papers of the 1989 IEEE Conference on Neural Information Processing Systems - Natural and Synthetic, Denver, CO, November 1989, Morgan Kaufmann Publishers.

"Rationale for a 'Many Maps' Phonology Machine" will appear in the *Proceedings of EMCSR-90: the Tenth European Meeting on Cybernetics Systems Research*, Vienna, Austria, April 1990, World Scientific Publishing Co.

Abstract

This report contains three papers on symbol processing in connectionist networks. The first two, "A Computational Basis for Phonology" and "Rationale for a 'Many Maps' Phonology Machine," present the latest results of our ongoing project to develop a connectionist explanation for the regularities and peculiarities of human phonological behavior. The third paper, "Rule Representations in a Connectionist Chunker," introduces a new rule chunking architecture based on competitive learning, and compares its performance with that of a backpropagation-based chunker. Earlier work in these areas was described in report CMU-CS-89-158, "Rules and Maps in Connectionist Symbol Processing."

This work was supported by a contract from Hughes Research Laboratories, by National Science Foundation grant EET-8716324, and by the Office of Naval Research under contract number N00014-86-K-0678.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Hughes Research Laboratories, the National Science Foundation, the Office of Naval Research, or the U.S. Government.

Keywords: Artificial intelligence, natural language processing, linguistics, learning algorithms

A Computational Basis for Phonology

David S. Touretzky
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Deirdre W. Wheeler
Department of Linguistics
University of Pittsburgh
Pittsburgh, PA 15260

ABSTRACT

The phonological structure of human languages is intricate, yet highly constrained. Through a combination of connectionist modeling and linguistic analysis, we are attempting to develop a computational basis for the nature of phonology. We present a connectionist architecture that performs multiple simultaneous insertion, deletion, and mutation operations on sequences of phonemes, and introduce a novel additional primitive, *clustering*. Clustering provides an interesting alternative to both iterative and relaxation accounts of assimilation processes such as vowel harmony. Our resulting model is efficient because it processes utterances entirely in parallel using only feed-forward circuitry.

1 INTRODUCTION

Phonological phenomena can be quite complex, but human phonological behavior is also highly constrained. Many operations that are easily learned by a perceptron-like sequence mapping network are excluded from real languages. For example, as Pinker and Prince (1988) point out in their critique of the Rumelhart and McClelland (1986) verb learning model, human languages never reverse the sequence of segments in a word, but this is an easy mapping for a network to learn. On the other hand, we note that some phonological processes that are relatively common in human languages, such as vowel harmony, appear difficult for a sequence-mapping architecture to learn. Why are only certain types of sequence operations found in human languages, and not others? We suggest that this is a reflection of the limitations of an underlying, genetically-determined, specialized computing architecture. We are searching for this architecture.

Our work was initially inspired by George Lakoff's theory of cognitive phonology (Lakoff, 1988, 1989), which is in turn a development of the ideas of John Goldsmith (to appear). Lakoff proposes a three-level representation scheme. The M (morphophonemic) level represents the underlying form of an utterance, the P (phonemic) level is an intermediate form, and the F (phonetic) level is the derived surface form.

Lakoff uses a combination of inter-level mapping rules and intra-level well-formedness conditions to specify the relationships between P- and F-level representations and the M-level input. In a connectionist implementation, the computations performed by the mapping rules are straightforward, but we find the well-formedness conditions troubling. Goldsmith's proposal was that phonology is a goal-directed constraint satisfaction system that operates via parallel relaxation. He cites Smolensky's harmony theory[1] Lakoff has adopted this appeal to harmony theory in his description of how well-formedness conditions could work.

In our model, we further develop the Goldmsith and Lakoff mapping scheme, but we reject harmony-based well-formedness conditions for several reasons. First, harmony theory involves simulated annealing search. The timing constraints of real nervous systems rule out simulated annealing. Second, it is not clear how to construct an energy function for a connectionist network that performs complex discrete phonological operations. Finally there is our desire to explain why certain types of processes occur in human languages and others do not. Harmony theory alone is too unconstrained for this purpose.

We have implemented a model called M^3P (for "Many Maps" Model of Phonology) that allows us to account for virtually all of the phenomena in (Lakoff, 1989) using a tightly-constrained, purely-feedforward computing scheme. In the next section we describe the mapping matrix architecture that is the heart of M^3P. Next we give an example of an iterative process, Yawelmani vowel harmony,[2], which Lakoff models with a P-level well-formedness condition. Such a condition would have to be implemented by relaxation search for a "minimum energy state" in the P-level representation, which we wish to avoid. Finally we present our alternative approach to vowel harmony, using a novel clustering mechanism that eliminates the need for relaxation.

2 THE MAPPING MATRIX ARCHITECTURE

Figure 1 is an overview of our "many maps" model. M-P constructions compute how to go from the M-level representation of an utterance to the P-level representation. The derivation is described as a set of explicit changes to the M-level string. M-P constructions read the segments in the M-level buffer and write the changes, phrased as mutation, deletion, and insertion requests, into slots of a buffer called P-deriv. The M-level and P-deriv buffers are then read by the M-P mapping matrix, which produces the P-level representation as its output. The process is repeated at the next level, with P-F constructions writing changes into an F-deriv buffer, and a P-F map deriving an F-level

[1] Smolensky's "harmony theory" should not be confused with the linguistic phenomenon of "vowel harmony."

[2] Yawelmani is a dialect of Yokuts, an American Indian language from California. Our Yawelmani data is drawn from Kenstowicz and Kisseberth (1979), as is Lakoff's.

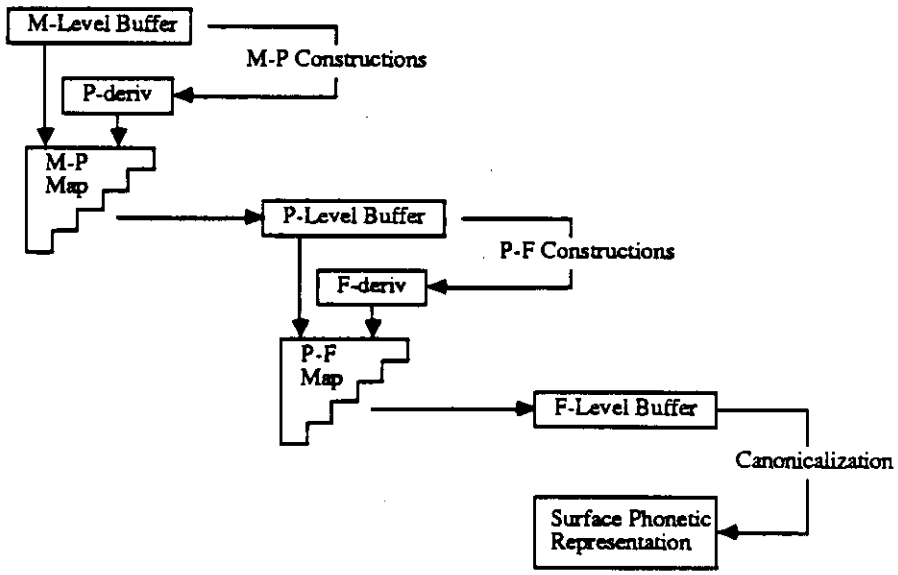

Figure 1: Overview of the "many maps" model.

representation. A final step called "canonicalization" cleans up the representations of the individual segments.

Figure 2 shows the effect of an M-P construction that breaks up CCC consonant clusters by inserting a vowel after the first consonant, producing CiCC. The input in this case is the Yawelmani word /?ugnhin/ "drinks", and the desired insertion is indicated in P-deriv. The mapping matrix derives the P-level representation right-justified in the buffer, with no segment gaps or collisions. It can do this even when mutliple simultaneous insertions and deletions are being performed. But it cannot perform arbitrary sequence manipulations, such as reversing all the segments of an utterance. Further details of the matrix architecture are given in (Touretzky, 1989) and (Wheeler and Touretzky, 1989).

3 ITERATIVE PHENOMENA

Several types of phonological processes operate on groups of adjacent segments, often by making them more similar to an immediately preceding (or following) trigger segment. Vowel harmony and voicing assimilation are two examples. In Yawelmani, vowel harmony takes the following form: an [αhigh] vowel that is preceded by an [αhigh] round vowel becomes round and back. In the form /do:s+al/ "might report", the non-round, back vowel /a/ is [$-$high], as is the preceding round vowel /o/. Therefore the /a/ becomes round, yielding the surface form [do:sol]. Similarly, in /dub+hin/ "leads by the hand", the [+high] vowel /i/ is preceded by the [+high] round vowel /u/, so the /i/ becomes round and back, giving [dubhun]. In /bok'+hin/ "finds", the /i/ does not undergo harmony because it differs in height from the preceding vowel.

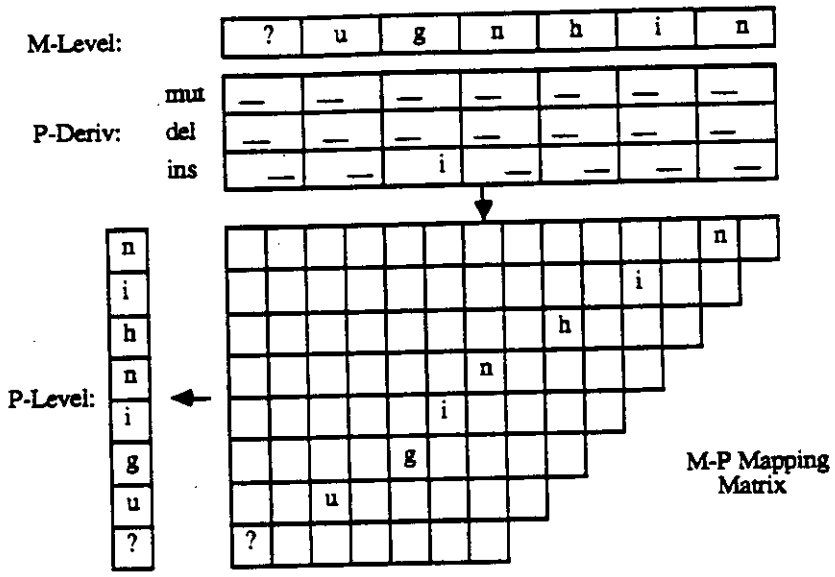

Figure 2: Performing an insertion via the M-P mapping matrix.

Harmony is described as an iterative process because it can apply to entire sequences of vowels, as in the following derivation:

/t'ul+sit+hin/	"burns for"
/t'ul+sut+hin/	*harmony on second vowel*
/t'ul+sut+hun/	*harmony on third vowel*

In Yawelmani we saw an epenthesis process that inserts a high vowel /i/ to break up lengthy consonant clusters. Epenthetic vowels may either undergo or block harmony. With the word /logw+xa/ "let's pulverize", epenthesis inserts an /i/ to break up the /gwx/ cluster, producing /logiw+xa/. Now the /a/ is preceded by a [+high, −round] vowel, so harmony does not apply, whereas in /do:s+al/, which has the same sequence of underlying vowels, it did. This is an instance of epenthesis blocking harmony. In other environments the epenthetic vowel may itself undergo harmony. For example:

/ʔugn+hin/	"drinks"
/ʔuginhin/	*epenthesis*
/ʔugunhin/	*harmony on epenthetic vowel*
/ʔugunhun/	*harmony on third vowel*

The standard generative phonology analysis of harmony utilizes the following rule, applying after epenthesis, that is supposed to iterate through the utterance from left to right, changing one vowel at a time:

$$\begin{bmatrix} +\text{syll} \\ \alpha\text{high} \end{bmatrix} \rightarrow \begin{bmatrix} +\text{round} \\ +\text{back} \end{bmatrix} / \begin{bmatrix} +\text{syll} \\ +\text{round} \\ \alpha\text{high} \end{bmatrix} C_0 \_$$

Lakoff offers an alternative account of epenthesis and harmony that eliminates iteration. He states epenthesis as an M-P construction:

M: C C {C,#}
P: [ ] i [ ]

The harmony rule is stated as a P-level well-formedness condition that applies simultaneously throughout the buffer:

P: If [+syll, +round, αhigh] C_0 X,
then if X = [+syll, αhigh], then X = [+round, +back].

Starting with /ʔugn+hin/ at M-level, Lakoff's model would settle into a representation of /ʔugunhun/ at P-level. We repeat again the crucial point that this representation is not derived by sequential application of rules; it is merely *licensed* by one application of epenthesis and two of harmony. The actual computation of the P-level representation would be performed by a parallel relaxation process, perhaps using simulated annealing, that somehow determines the sequence that best satisfies all applicable constraints at P-level.

4 THE CLUSTERING MECHANISM

Our account of vowel harmony must differ from Lakoff's because we do not wish to rely on relaxation in our model. Instead, we introduce special clustering circuitry to recognize sequences of segments that share certain properties. The clustering idea is meant to be analogous to perceptual grouping in vision. Sequences of adjacent visually-similar objects are naturally perceived as a whole. A similar mechanism operating on phonological sequences, although unprecedented in linguistic theory, does not appear implausible. Crucial to our model is the principle that perceived sequences may be operated on as a unit. This allows us to avoid iteration and give a fully-parallel account of vowel harmony.

The clustering mechanism is controlled by a small number of language-specific parameters. The rule shown below is the P-F clustering rule for Yawelmani. Cluster type [+syllabic] indicates that the rule looks only at vowels. (This is implemented by an additional mapping matrix that extracts the vowel projection of the P-level buffer. The clustering mechanism actually looks at the output of this matrix rather than at the P-level buffer directly.) The trigger of a cluster is a round vowel of a given height, and the elements are the subsequent adjacent vowels of matching height. Application of the rule causes elements (but not triggers) to undergo a change; in this case, they become round and back.

Yawelmani vowel harmony — P-F mapping:

Cluster type:	[+syllabic]
Trigger:	[+round, αhigh]
Element:	[αhigh]
Change:	[+round, +back]

The following hypothetical vowel sequence illustrates the application of this clustering rule. Consonants are omitted for clarity:

	1	2	3	4	5	6	7	8	9
	i	u	i	i	e	o	o	a	i
trigger:		+				+			
element:			+	+			+	+	

The second vowel is round, so it's a trigger. Since the third and fourth vowels match it in height, they become elements. The fifth vowel is [−high], so it is not included in the cluster. The sixth vowel triggers a new cluster because it's round; it is also [−high]. The seventh and eighth vowels are also [−high], so they can be elements, but the ninth vowel is excluded from the cluster because is [+high]. Note that vowel 7 is an element, but it also meets the specification for a trigger. Given a choice, our model prefers to mark segments as elements rather than triggers because only elements undergo the specified change. The distinction is moot in Yawlemani, where triggers are already round and back, but it matters in other languages; see (Wheeler and Touretzky, 1989) for details.

Figures 2 and 3 together show the derivation of the Yawelmani word [?ugunhun] from the underlying form /?ugn+hin/. In figure 2 an M-P construction inserted a high vowel. In figure 3 the P-F clustering circuitry has examined the P-level buffer and marked the triggers and elements. Segments that were marked as elements then have the change [+round, +back] written into their corresponding mutation slots in F-deriv. Finally, the P-F mapping matrix produces the sequence /?ugunhun/ as the F-level representation of the utterance.

5 DISCUSSION

We could not justify the extra circuitry required for clustering if it were suitable only for Yawelmani vowel harmony. The same mechanism handles a variety of other iterative phenomena, including Slovak and Gidabal vowel shortening, Icelandic umlaut, and Russian voicing assimilation. The full mechanism has some additional parameters beyond those covered in the discussion of Yawelmani. For example, clustering may proceed from right-to-left (as is the case in Russian) instead of from left-to-right. Also, clusters may be of either bounded or unbounded length. Bounded clusters are required for alternation processes, such as Gidabal shortening. They cover exactly two segments: a trigger and one element. We are making a deliberate analogy here with metrical phonology (stress systems), where unbounded feet may be of arbitrary length, but bounded feet always contain exactly two syllables. No language has strictly trisyllabic feet. We predict a similar constraint will hold for iterative phenomena when they are reformulated in parallel clustering terms, i.e., no language requires bounded-length clusters with more than one element.

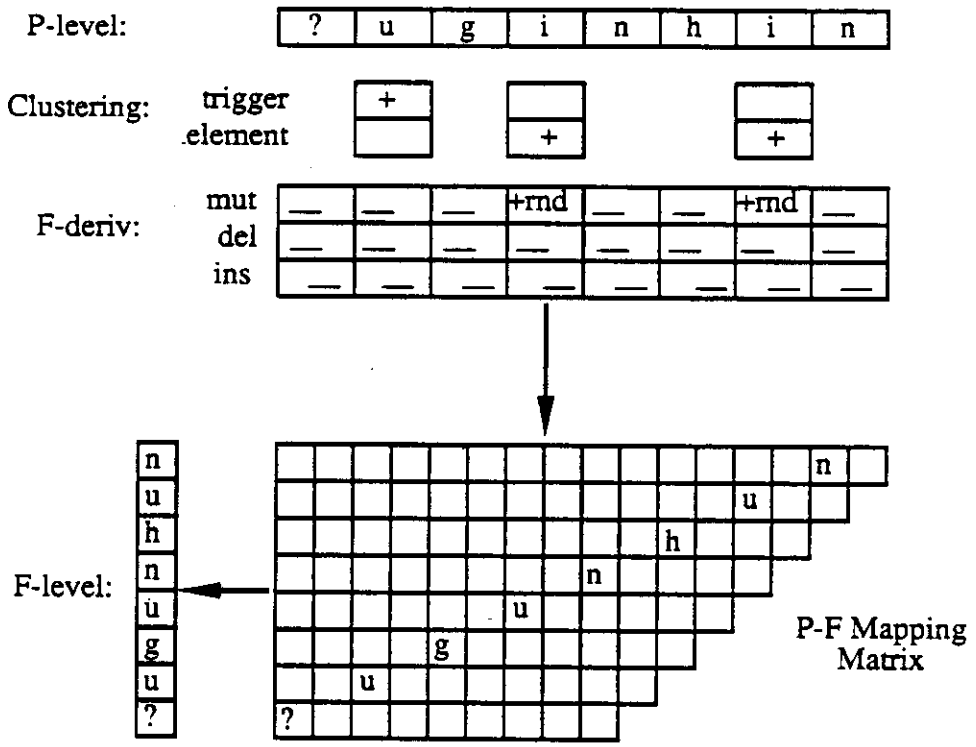

Figure 3: Clustering applied to Yawelmani vowel harmony.

Our model makes many other predictions of constraints on human phonology, based on limitations of the highly-structured "many maps" architecture. We are attempting to verify these predictions, and also to extend the model to additional aspects of phonological behavior, such as syllabification and stress.

Acknowledgements

This research was supported by a contract from Hughes Research Laboratories, by the Office of Naval Research under contract number N00014-86-K-0678, and by National Science Foundation grant EET-8716324. We thank George Lakoff for encouragement and support, John Goldsmith for helpful correspondence, and Gillette Elvgren III for implementing the simulations.

References

Goldsmith, J. (to appear) Phonology as an intelligent system. To appear in a festschrift for Leila Gleitman, edited by D. Napoli and J. Kegl.

Kenstowicz, M., and Kisseberth, C. (1979) *Generative Phonology: Description and Theory*. San Diego, CA: Academic Press.

Lakoff, G. (1988) A suggestion for a linguistics with connectionist foundations. In D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski (eds.), *Proceedings of the 1988 Connectionist Models Summer School*, pp. 301-314. San Mateo, CA: Morgan Kaufmann.

Lakoff, G. (1989) Cognitive phonology. Draft of paper presented at the UC-Berkeley Workshop on Constraints vs Rules, May 1989.

Pinker, S., and Prince, A. (1988) On language and connectionism: analysis of a parallel distributed processing model of language acquisition. In S. Pinker & J. Mehler (eds.), *Connections and Symbols*. Cambridge, Massachusetts: MIT Press.

Rumelhart, D. E., and McClelland, J. L. (1986) On learning the past tenses of English verbs. In J. L. McClelland and D. E. Rumelhart (eds.), *Parallel Distributed Processing: Explorations in the MicroStructure of Cognition*, volume 2. Cambridge, Massachusetts: MIT Press.

Smolensky, P. (1986) Information processing in dynamical systems: foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the MicroStructure of Cognition*, volume 1. Cambridge, Massachusetts: MIT Press.

Touretzky, D. S. (1989) Toward a connectionist phonology: the "many maps" approach to sequence manipulation. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pp. 188-195. Hillsdale, NJ: Erlbaum.

Wheeler, D. W., and Touretzky, D. S. (1989) A connectionist implementation of cognitive phonology. Technical report CMU-CS-89-144, Carnegie Mellon University, School of Computer Science. To appear in G. Lakoff and L. Hyman (eds.), *Proceedings of the UC-Berkeley Phonology Workshop on Constraints vs. Rules*, University of Chicago Press.

Rationale for a "Many Maps" Phonology Machine

DAVID S. TOURETZKY
School of Computer Science, Carnegie Mellon University
Pittsburgh, Pennsylvania, 15213, USA

DEIRDRE W. WHEELER
Department of Linguistics, University of Pittsburgh
Pittsburgh, Pennsylvania, 15260, USA

Abstract

M^3P, our "many maps" model of phonology, raises a number of questions about the nature of linguistic explanations and the ways in which connectionist models can contribute to the advancement of phonological theory. In this paper we attempt to answer some of the questions we and others have raised as a result of this work. We consider four sources of possible phonological constraints, and argue that articulatory and intelligibility constraints are insufficient to fully account for human phonological behavior. Computational constraints such as those suggested by our connectionist model may provide a solution.

1. Introduction

For a variety of reasons, we view phonology as an attractive starting point for cognitive scientists seeking to understand language. The domain, sequences of phonemes, is purely symbolic. The operations are familiar: chiefly insertion, deletion, and mutation of elements. The structures involved are quasi-linear. (Some theories employ limited-depth trees to represent syllables or feature hierarchies, but phonology does not admit self-similar embedded structures or objects of unbounded depth of the sort required by syntax.) True phonological processes are highly regular: they do not suffer from the plethora of special cases that complicates syntax and morphology. Even in those processes that are morphologically conditioned (and thus not purely phonological), such as the English /k/→/s/ rule that derives "electricity" from "electric" plus "-ity," the phoneme transformation itself is regular. Complexity comes only from the attachment of a morphological condition to the rule's environment.

Phonology is a unique language component for yet another reason: it is an autonomous process, not intertwined with higher levels in the way that syntax and semantics mutually interact. We acknowledge the existence of morphophonemic processes, but even here, influence flows in only one direction: morphological conditioning of phonological rules. Phonological processes do not interact with morphology. Autonomy is perhaps the most compelling reason why we see phonology as an approachable

domain for connectionist modeling. It is a domain where one may hope to achieve decisive and convincing results about human capabilities by pursuing a computation-driven approach.

Our goal in developing M^3P, our "Many Maps" Model of Phonology, is not to merely imitate human behavior or to implement a pre-existing theory. Rather, it is to investigate the ways in which adopting a particular model of computation—and its accompanying biologically-inspired constraints—can direct the development of linguistic theories, and even provide motivations for constraints on linguistic processes.

2. Overview of the Model

M^3P began as an attempt to implement George Lakoff's theory of cognitive phonology (Lakoff, 1988; Lakoff, 1989) in connectionist hardware. An early, incomplete version was described in (Touretzky, 1989). The current version is described in (Wheeler & Touretzky, 1989; Touretzky & Wheeler, 1989; Touretzky & Wheeler, in press). This version differs substantially from Lakoff's proposal—a reflection of the theoretical progress made as the model has matured.

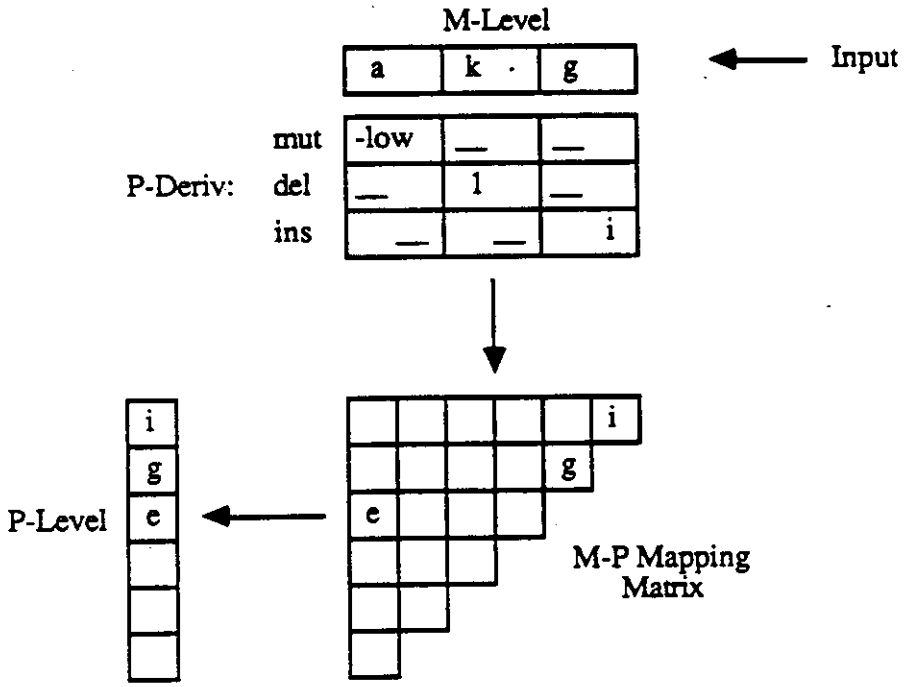

Figure 1: Example of the model's M-P map.

We cannot fully describe the details of the model here, but Figure 1 gives the flavor of our approach. This figure shows how an M (morpho-phonemic) level representation

of an example utterance, /akg/, is mapped to a P (phonemic) level representation, /egi/. In this artificial example the derivation involves three changes to the string: a mutation, a deletion, and an insertion. The changes are described in a "change buffer" called P-deriv. The M-level and P-deriv buffers both feed into an M-P mapping matrix whose job is to derive the phonemic representation, right-justified in the P-level output buffer, in one parallel step. The mapping matrix assures that there are no gaps or collisions in the output caused by multiple simultaneous insertions and deletions.

How do changes get written into the change buffer? One way is via M-P constructions (the counterpart of "rules" in traditional generative phonology) which examine the M-level representation and insert changes into P-deriv. Our model also contains a clustering mechanism that makes it possible to recognize clusters of adjacent segments sharing some property. Clustering provides an alternative to the traditional iterative accounts of phenomena such as vowel harmony, or voicing assimilation in consonant clusters, reminiscent of autosegmental representations. The M-level cluster modules are implemented using additional maps; cluster constructions read the states of these modules and write their changes into P-deriv, just as the M-P constructions do.

Most recently we have added a syllabifier to our model. Many insertion and deletion phenomena can be explained by the requirement that utterances be organized into well-formed syllables. Our syllabifier provides additional input into the mapping matrix, so that, for example, unsyllabified M-level segments will not appear at P-level, in effect causing the segment to be deleted.

The P-level representation is then fed through a second, very similar bit of mapping hardware called the the P-F map (not shown in Figure 1) to derive the F-level (phonetic) representation of the utterance. This P-F mapping is controlled by P-F constructions plus the special constructions associated with P-level clustering modules.

Our model is not capable of arbitrary string transformations. Its behavior is tightly constrained by a combination of factors: the mapping matrix can only perform insertions, deletions, and mutations of segments; there are only two levels of derivation, M-P and P-F; the clustering modules are highly specialized, and just powerful enough to model actual phonological phenomena such as vowel harmony; the syllabification mechanism is equally specialized. The key question that remains is: what is the relationship between the constraints on the model's behavior and the constraints that human beings appear to observe?

3. Sources of Phonological Constraints

In trying to account for the nature of of human phonological behavior, we find there are four sources of potential constraints. We consider them in turn.

3.1. Articulation

The first source of constraints is articulatory convenience: some sounds are simply easier to make than others. For example, affricates (the *ch* sound in "church") are relatively rare—English being an exception—while alvelar fricatives such as /s/ occur in many languages. Likewise, the basic vowels /a,i,u/ occur almost universally, while ü, ɤ̃ and ɔ are less common. Another example is consonant clusters. Many languages severely limit the ways in which consonants may adjoin; some have a strict CV structure in which consonants are always separated by vowels. (We note that insertion rules are frequently motivated by a syllabifier's desire to break up "unpronounceable" consonant clusters.) In contrast, English permits a variety of tri-consonantal clusters, such as /spl/ ("splash") and /skr/ ("scrap"). Many languages permit even more complex clusters.

Articulatory constraints are only weak constraints, because sequences that speakers of one language find unpronounceable may sound natural to other communities. However, it is still possible to objectively classify certain sounds or sound sequences as more marked than others, based on articulatory effort. If one looks at a wide spectrum of languages, the more marked sequences appear less frequently.

3.2. Computability

The second source of potential constraints on phonology is computability. Computational constraints reflect fundamental limitations imposed by the wiring of the brain's language production areas. These are the hard, universal constraints we are attempting to capture in M^3P. By its very nature, the model is unable to perform certain types of string transformations, such as reversing the order of phonemes in an utterance, or permuting the first and last consonant of a word. The model therefore predicts that no human language could possibly do these things.

We see a close relationship between our approach to phonology and the notion of parameters in (Chomsky, 1988). Chomsky suggests that languages can be characterized by particular sets of parameter values. The number of these parameters, their meanings, and their range of legal values is the province of Universal Grammar. The job of the language learner is to determine the particular parameter settings in use in his or her linguistic environment. This proposal has been put into practice by Dresher and Kaye (1990), who describe a mechanism for learning a language's metrical structure from examples. (The Dresher and Kaye model was first brought to our attention by Eric Nyberg, who suggests an alternative approach to the parameter setting problem in (Nyberg, 1989).)

The notion of a genetically-determined language machine configured by parameter values is certainly in harmony with our M^3P model. However, language universals can presumably be captured by more than one parameter scheme (and more than one machine architecture), just as there are several distinctive feature systems that adequately

characterize phonemes. Chomsky leaves open the questions of where parameter systems come from and how to choose among them when evaluating theories of universal grammar. Our work provides a way to compare alternative parameter systems by looking at the underlying machine architecture each assumes. Criteria include, for example, circuit complexity, circuit depth, and degree of parallelism.

3.3. Intelligibility

Geoffrey Hinton (personal communication) has suggested a third source of phonological constraints: intelligibility. The hearer must be able to decode the speech signal. This limits the types of transformations speakers may make. Certain types of phonological transformations might simply be too difficult for the hearer to invert to arrive back at the correct underlying form.

There is a tension between the intelligibility and computability constraints. Both can account for the fact that speakers don't invert the phoneme order of entire words. But the intelligibility constraint does not seem powerful enough to explain all the peculiarities of human phonology. For example, switching the first and last consonant of a word wouldn't seem to interfere too much with intelligibility. Also, neutralization processes such as vowel reduction in English or devoicing of word final stops in German actually work against intelligibility in favor of articulatory convenience. Therefore, intelligibility seems too weak a constraint to fully account for the structure of phonology.

On the other hand, the computational constraints we have been proposing could turn out to be too powerful. People are remarkably flexible; it would be difficult to conclusively demonstrate that they are fundamentally incapable of certain types of phonological behavior. One objection frequently raised in response to M^3P's computational constraints is Pig latin, a language game in which, for example, "games are fun" becomes "ames-gay are-way un-fay." This involves the sort of movement operation which our theory predicts is phonologically impossible. However, we doubt that competent Pig latin speakers are employing their natural phonological machinery to accomplish this task. Unlike a real language, Pig latin requires the speaker to first produce the correct English *surface* form of each word, and then transform it by one of two simple rules depending on whether the word begins with a vowel. The automaticity of Pig latin speech (after sufficient practice) does not imply that Pig latin rules have become part of the speaker's phonology. Many automatic processes, such as syntax, or the ability to play the piano, are non-phonological. What counts as "phonology" in our view are those processes that occur naturally in human languages and can be acquired automatically and *unconsciously* by children in a suitable linguistic environment. In contrast, Pig latin speakers must acquire a conscious representation of the rules of the game before they can speak it correctly.

3.4. History

A fourth source of potential phonological constraints, suggested to us by Prahlad Gupta (personal communication), is history. There is no way to know if the 5000 or so languages that have developed during the course of human history fully exercise our linguistic abilities. We must be careful to avoid turning historical accidents into universal principles. Just because a particular process has never occurred historically, this does not imply *a fortiori* that it is constrained from occurring some time in the future. Thus, one would be wise not to formulate overly-specific constraints based solely on historical evidence. The computational approach is helpful by providing additional motivation for certain types of constraints, but grey areas do remain. For example, to the best of our knowledge, no human language utilizes a quality-sensitive stress rule, such as "stress the penultimate high vowel in the word." This could be a legitimate computational constraint, but it might also be a historical artifact.

4. Relationship to Neuroscience

What is known to date about the neural basis of language comes largely from clinical studies (and subsequent autopsies) of stroke and head injury patients. Recently, some interesting new results have been obtained with radioactive imaging techniques that map metabolic activity throughout the brain during performance of language-related tasks. Along with the autopsy data, this is another valuable source of clues about how cognitive functions are distributed across different cortical regions. However, at this time there is still no detailed theory of how linguistic information is physically represented and processed in the brain.

M^3P should not be taken literally as a biological model. We don't expect to find such clean representations and regular wiring structures in real brain tissue. Furthermore, our model does not yet account for developmental processes, speech errors, or the various types of aphasias people exhibit.

We do not wish to suggest, though, that our model is completely divorced from the neural level, the way Chomsky's theories are divorced from actual computation. We are making strong claims about the functional nature of the brain's language areas. Specifically, we predict that certain types of phonological processes are impossible, because they are incompatible with the M^3P architecture. Even if the wiring of our model differs significantly from real neural circuitry, as we know it must, we assert that at the functional level the two systems may be equivalent.

Our work can be usefully contrasted with backpropagation-based approaches to phonology, such as the Rumelhart and McClelland (1986) verb learning model, or the more recent work using sequential recurrent nets by Gasser and Lee (1989). These models can in principle learn any input-output mapping, so they are unable to impose hard

constraints on phonological operations the way M^3P does. In fact, they don't appear to offer any explanation for the extremely regular and symbolic nature of phonology. At best, they may correctly mimic human phonological behavior. Often, though, their behavior is not completely correct after training.

Since rules have no independent existence in a backprop-based model other than as facets of a monolithic input-output mapping, each combination of rule interactions must be learned as a separate case. There is no mechanism requiring the individual rules to interact systematically. Backprop-based systems without internal structure, if they are powerful enough to model real human phonology, can just as easily learn many nonphonological behaviors. Unlike the Rumelhart and McClelland model, Gasser and Lee's sequential network cannot easily learn reversals and long-range metatheses, but this limitation also hinders it from modeling *any* process that examines segments from right to left, such as regressive feature spreading or assignment of stress to penultimate syllables, both of which are common in human languages.

Even if a more sophisticated backprop-based model could be constructed whose generalization behavior were completely correct, an important question would remain. Is the brain fundamentally incapable of certain types of phonological operations, as M^3P predicts, or is its phonological machinery as unrestricted as the backprop models, merely adapting itself in response to linguistic inputs? This is, of course, an instance of the classic rationalism vs. empiricism controversy. We are content to toil in the rationalist camp, enhancing our model and predicting constraints that linguists may seek to verify or refute. Some day, neurolinguistics may provide the decisive answer to the rationalist/empiricist debate. If so, we are confident that such progress will have been made possible, at least in part, by the work of connectionist modelers in both camps.

Acknowledgements

This research was supported by a contract from Hughes Research Laboratories, by the Office of Naval Research under contract number N00014-86-K-0678, and by National Science Foundation grant EET-8716324. We thank Gillette Elvgren III for his work on implementing the simulations, and George Lakoff, Geoffrey Hinton, and Prahlad Gupta for stimulating conversation.

References

[1] Chomsky, N. (1988) Language and Problems of Knowledge. Cambridge, MA: MIT Press.

[2] Dresher, B. E., and Kaye, J. D. (1990) A computational learning model for metrical phonology. To appear in Cognition.

[3] Gasser, M., and Lee, C. (1989) Networks that learn phonology. Computer Science technical report number 300, Indiana University.

[4] Lakoff, G. (1988) A suggestion for a linguistics with connectionist foundations. In D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski (eds.), Proceedings of the 1988 Connectionist Models Summer School, pp. 301-314. San Mateo, California: Morgan Kaufmann.

[5] Lakoff, G. (1989) Cognitive phonology. Draft of paper presented at the UC-Berkeley Workshop on Constraints vs Rules, May 1989.

[6] Nyberg, E. H. (1989) Weight propagation and parameter setting. Ph.D. thesis proposal, computational linguistics program, Carnegie Mellon University.

[7] Touretzky, D. S. (1989) Toward a connectionist phonology: the "many maps" approach to sequence manipulation. Proceedings of the Eleventh Annual Conference of the Cognitive Science Society, pp. 188-195. Hillsdale, NJ: Erlbaum.

[8] Touretzky, D. S., and Wheeler, D. W. (1989) Two derivations suffice: the role of syllabification in cognitive phonology. Manuscript.

[9] Touretzky, D. S., and Wheeler, D. W. (in press) A computational basis for phonology. In D. S. Touretzky (ed.), Advances in Neural Information Processing Systems 2. San Mateo, California: Morgan Kaufmann. To appear April, 1990.

[10] Wheeler, D. W., and Touretzky, D. S. (1989) A connectionist implementation of cognitive phonology. Technical Report CMU-CS-89-144, Carnegie Mellon University School of Computer Science. To appear in G. Lakoff and L. Hyman (eds.), Proceedings of the UC-Berkeley Phonology Workshop on Constraints vs, Rules, University of Chicago Press.

Rule Representations in a Connectionist Chunker

David S. Touretzky **Gillette Elvgren III**
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

We present two connectionist architectures for chunking of symbolic rewrite rules. One uses backpropagation learning, the other competitive learning. Although they were developed for chunking the same sorts of rules, the two differ in their representational abilities and learning behaviors.

1 INTRODUCTION

Chunking is a process for generating, from a sequence of if-then rules, a more complex rule that accomplishes the same task in a single step. It has been used to explain incremental human performance improvement in a wide variety of cognitive, perceptual, and motor tasks (Newell, 1987). The SOAR production system (Laird, Newell, & Rosenbloom, 1987) is a classical AI computer program that implements a "unified theory of cognition" based on chunking.

SOAR's version of chunking is a symbolic process that examines the working memory trace of rules contributing to the chunk. In this paper we present two connectionist rule-following architectures that generate chunks a different way: they use incremental learning procedures to infer the environment in which the chunk should fire. The first connectionist architecture uses backpropagation learning, and has been described previously in (Touretzky, 1989a). The second architecture uses competitive learning. It exhibits more robust behavior than the previous one, at the cost of some limitations on the types of rules it can learn.

The knowledge to be chunked consists of context-sensitive rewrite rules on strings. For example, given the two rules

R1: D → B / _ E "change D to B when followed by E"
R2: A → C / _ B "change A to C when followed by B"

the model would go through the following derivation: ADE → (Rule R1) ABE → (Rule R2) CBE. Rule R1's firing is what enables rule R2 to fire. The model detects this and formulates a chunked rule (R1-R2) that can accomplish the same task in a single step:

R1-R2: AD → CB / _ E

Once this chunk becomes active, the derivation will be handled in a single step, this way: ADE → (Chunk R1-R2) CBE. The chunk can also contribute to the formation of larger chunks.

2 CHUNKING VIA BACKPROPAGATION

Our first experiment, a three-layer backpropagation chunker, is shown in Figure 1. The input layer is a string buffer into which symbols are shifted one at a time, from the right. The output layer is a "change buffer" that describes changes to be made to the string. The changes supported are deletion of a segment, mutation of a segment, and insertion of a new segment. Combinations of these changes are also permitted.

Rules are implemented by hidden layer units that read the input buffer and write changes (via their α connections) into the change buffer. Then separate circuitry, not shown in the figure, applies the specified changes to the input string to update the state of the input buffer. The details of this string manipulation circuitry are given in (Touretzky, 1989b; Touretzky & Wheeler, 1990).

We will now go through the ADE derivation in detail. The model starts with an empty input buffer and two rules: R1 and R2.[1] After shifting the symbol A into the input buffer, no rule fires—the change buffer is all zeros. After shifting in the D, the input buffer contains AD, and again no rule fires. After shifting in the E the input buffer contains ADE, and rule R1 fires, writing a request in the change buffer to mutate input segment 2 (counting from the right edge of the buffer) to a B. The input buffer and change buffer states are saved in temporary buffers, and the string manipulation circuitry derives a new input buffer state, ABE. This now causes rule R2 to fire.[2] It writes a request into the change buffer to mutate segment 3 to a C. Since it was R1's firing that triggered R2, the conditions exist for chunk formation. The model combines R1's requested change with that of R2, placing the result in the "chunked change buffer" shown on the right in Figure 1. Backpropagation is used to teach the hidden layer that when it sees the input buffer pattern that triggered R1 (ADE in this case) it should produce via its β connections the combined change pattern shown in the chunked change buffer.

The model's training is "self-supervised:" its own behavior (its history of rule firings) is the source of the chunks it acquires. It is therefore important that the chunking

[1] The initial rule set is installed by an external teacher using backpropagation.

[2] Note that R1 applies to positions 1 and 2 of the buffer (counting from the right edge), while R2 applies to positions 2 and 3. Rules are represented in a position-independent manner, allowing them to apply anywhere in the buffer that their environment is satisfied. The mechanism for achieving this is explained in (Touretzky, 1989a).

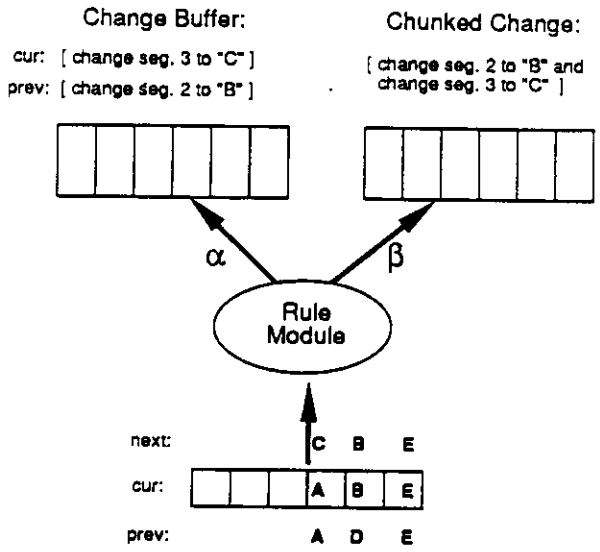

Figure 1: Architecture of the backpropagation chunker.

process not introduce any behavioral errors during the intermediate stages of learning, since no external teacher is present to force the model back on track should its rule representations become corrupted. The original rules are represented in the α connections and the chunked rules are trained using the β connections, but the two rule sets share the same hidden units and input connections, so interference can indeed occur. The model must actively preserve its α rules by continuous rehearsal: after each input presentation, backpropagation learning on a contrast-enhanced version of the α change pattern is used to counteract any interference caused by training on the β patterns. Eventually, when the β weights have been learned correctly, they can replace the α weights.

The parameters of the model were adjusted so that the initial rules had a distributed representation in the hidden layer, i.e., several units were responsible for implementing each rule. Analysis of the hidden layer representations after chunking revealed that the model had split off some of the R1 units to represent the R1-R2 chunk; the remainder were used to maintain the original R1 rule.

The primary flaw of this model is fragility. Constant rehearsal of the original rule set, and low learning rates, are required to prevent the α rules from being corrupted before the β rules have been completely learned. Furthermore, it is difficult to form long rule chains, because each chunk further splits up the hidden unit population. Repeated splitting and retraining of hidden units proved difficult, but the model did manage to learn an R1-R2-R3 chunk that supersedes the R1-R2 chunk, so that ADE mutates directly to CFE. The third rule was:

R3: $B \rightarrow F / C \_ E$ "change B to F when between C and E"

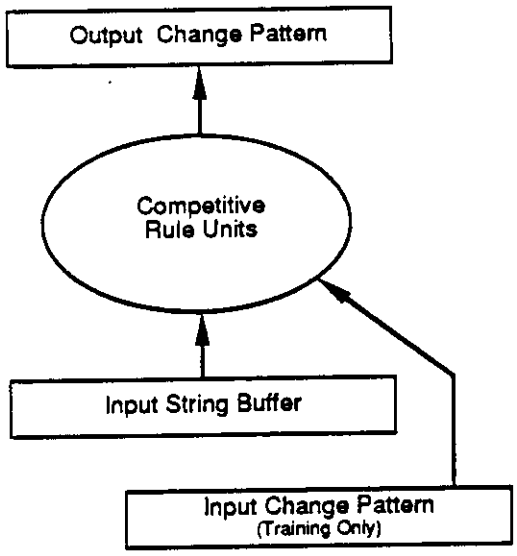

Figure 2: Architecture of the competitive learning chunker.

3 CHUNKING VIA COMPETITIVE LEARNING

Our second chunker, shown in Figure 2, minimizes interference between rules by using competitive learning to assign each rule a dedicated unit. As in the previous case, the model is taught its initial rules by showing it input buffer states and desired change buffer states. Chunks are then formed by running strings through the input buffer and watching for pairs of rules that fire sequentially. The model recruits new units for the chunks and teaches them to produce the new change buffer patterns (formed by composing the changes of the two original rules) in appropriate environments.

A number of technical problems had to be resolved in order to make this scheme work. First, we want to assign a separate unit to each rule, but not to each training example; otherwise the model will use too many units and not generalize well. Second, the encoding for letters we chose (see Table 1) is based on a Cartesian product, and so input patterns are highly overlapping and close together in Hamming space. This makes the job of the competitive learning algorithm more difficult. Third, there must be some way for chunks to take priority over the component rules from which they were formed, so that an input sequence like ADE fires the chunk R1-R2 rather than the original rule R1. As we trace through the operation of the chunker we will describe our solutions to these problems.

Rule units in the competitive layer are in one of three states: *inactive* (waiting to be recruited), *plastic* (currently undergoing learning), and *active* (weights finalized; ready to compete and fire.) They also contain a simple integrator (a counter) that is used to move them from the plastic to the active state. Initially all units are inactive and the counter

cremented. When the counter reaches a suitable value (currently 25), the unit switches from the plastic to the active state. It is now ready to compete with other units for the right to fire; its weights will not change further.

We now consider the formation of the model's first chunk. Assume that rules R1 and R2 have been acquired successfully. The model is trained by running random strings through the input buffer and looking for sequences of rule firings. Suppose the model is presented with the input string BFDADE. R1 fires, producing BFDABE; this then causes R2 to fire, producing BFDCBE. The model proceeds to form a chunk. The combined change pattern specifies that the penultimate segment should be mutated to "B," and the antepenultimate to "C." Since no plastic rule unit's change pattern weights match this change, a fresh unit is allocated and its change buffer weights are set to reproduce this pattern. The unit's input weights are set to detect the pattern BFDADE.

After several more examples of the R1-R2 firing sequence, the competitive learning algorithm will discover that the first three input buffer positions can hold anything at all, but the last three always hold ADE. Hence the weight vector will be concentrated on the last three positions. When its counter reaches a value of 25, the rule unit will switch to the active state.

Now consider the next time an input ending in ADE is presented. The network is in performance mode now, so there is nothing in the input change buffer; the model is looking only at the input string buffer. The R1 unit will be fully satisfied by the input; its normalized weight vector concentrates on just the last two positions, "DE," which match exactly. The R1-R2 unit will also be fully satisfied; its normalized weight vector looks for the sequence ADE. The latter unit is the one we want to win the competition. We achieve this by scaling the activation function of competitive units by an additional factor: the degree of distributedness of the weight vector. Units that distribute their input weight over a larger number of connections likely represent complex chunks, and should therefore have their activation boosted over rules with narrowly focused input vectors.

Once the unit encoding the R1-R2 chunk enters the active state, its more distributed input weights assure that it will always win over the R1 unit for an input like ADE. The R1 unit may still be useful to keep around, though, to handle a case like FDE → FBE that does not trigger R2.

Sometimes a new chunk is learned that covers the same length input as the old, e.g., chunk R1-R2-R3 that maps ADE → CFE looks at exactly the same input positions as chunk R1-R2. We therefore introduce one additional term into the activation function. As part of the learning process, active units that contribute to the formation of a new chunk are given a permanent, very small inhibitory bias. This ensures that R1-R2 will always lose the competition to R1-R2-R3 once that chunk goes from plastic to active, even though their weights are distributed to an equal degree.

Another special case that needs to be handled is when the competitive algorithm wrongly splits a rule between two plastic units in the same pool, e.g., one unit might be assigned the cases {A,B,C}ADE, and the other the cases {D,E,F}ADE. (In other words, one unit looks for the bit pattern 10xxx in the first position, and the other unit looks for 01xxx.)

Table 1: Input code for both chunking models.

A	1	0	1	0	0
B	1	0	0	1	0
C	1	0	0	0	1
D	0	1	1	0	0
E	0	1	0	1	0
F	0	1	0	0	1

is zero. As in any competitive learning scheme, the rule units' input weights are kept normalized to unit vectors (Rumelhart & Zipser, 1986).

When the teacher presents a novel instance, we must determine if there is already some partially-trained rule unit whose weights should be shaped by this instance. Due to our choice of input code, it is not possible to reliably assign training instances to rule units based solely on the input pattern, because "similar" inputs (close in Hamming space) may invoke entirely different rules. Our solution is to use the desired change pattern as the primary index for selecting a pool of plastic rule units; the input buffer pattern is then used as a secondary cue to select the most strongly activated unit from this pool.

Let's consider what happens with the training example DE → BE. The desired change pattern "mutate segment 2 to a B" is fed to the competitive layer, and the network looks for plastic rule units whose change patterns exactly match the desired pattern.[3] If no such unit is found, one is allocated from the inactive pool, its status is changed to "plastic," its input buffer weights are set to match the pattern in the input buffer, and its change pattern input and and change pattern output weights are set according to the desired change pattern.

Otherwise, if a pool of suitable plastic units already exists, the input pattern DE is presented to the competitive layer and the selected plsatic units compete to see which most closely matches the input. The winning unit's input buffer weights are then adjusted by competitive learning to move the weight vector slightly closer to this input buffer vector. The unit's counter is also bumped.

Several presentations are normally required before a rule unit's input weights settle into their correct values, since the unit must determine from experience which input bit values are significant and which should be ignored. For example, rule S1 in Table 2 (the asterisk indicates a wildcard) can be learned from the training instances ACF and ADF, since as Table 1 shows, the letters C and D in the second segment have no bits in common. Therefore the learning algorithm will concentrate virtually all of the weight vector's magnitude in the connections that specify "A" as the first segment and "F" as the third.

Each time a rule unit's weights are adjusted by competitive learning, its counter is in-

[3]The units' thresholds are raised so that they can only become active if their weight vectors match the input change buffer vector exactly.

This is bad because it allows the weights of each unit to be more distributed than they need to be. To correct the problem, whenever a plastic unit wins a competition our algorithm makes sure that the nearest runner up is considerably less active than the winner. If its activation is too high, the runner up is killed. This causes the survivor to readjust its weights to describe the rule correctly, i.e., it will look for the input pattern ADE. If the runner up was killed incorrectly (meaning it is really needed for some other rule), it will be resurrected in response to future examples.

Finally, active units have a decay mechanism that is kept in check by the unit's firing occasionally. If a unit does not fire for a long time (200 input presentations), its weights decay to zero and it returns to the inactive state. This way, units representing chunks that have been superseded will eventually be recycled.

4 DISCUSSION

Each of the two learning architectures has unique advantages. The backpropagation learner can in principle learn arbitrarily complex rules, such as replacing a letter with its successor, or reversing a subset of the input string. Its use of a distributed rule representation allows knowledge of rule R1 to participate in the forming of the R1-R2 chunk. However, this representation is also subject to interference effects, and as is often the case with backprop, learning is slow.

The competitive architecture learns very quickly. It can form a greater number of chunks, and can handle longer rule chains, since it avoids inteference by assigning a dedicated unit to each new rule it learns.

Both learners are sensitive to changes in the distribution of input strings; new chunks can form any time they are needed. Chunks that are no longer useful in the backprop model will eventually fade away due to non-rehearsal; the hidden units that implement these chunks will be recruited for other tasks. The competitive chunker uses a separate decay mechanism to recycle chunks that have been superseded.

This work shows that connectionist techniques can yield novel and interesting solutions to symbol processing problems. Our models are based on a sequence manipulation architecture that uses a symbolic description of the changes to be made (via the change buffer), but the precise environments in which rules apply are never explicitly represented. Instead they are induced by the learning algorithm from examples of the models' own behavior. Such self-supervised learning may play an important role in cognitive development. Our work shows that it is possible to correctly chunk knowledge even when one cannot predict the precise environment in which the chunks should apply.

Acknowledgements

This research was supported by a contract from Hughes Research Laboratories, by the Office of Naval Research under contract number N00014-86-K-0678, and by National Science Foundation grant EET-8716324. We thank Allen Newell, Deirdre Wheeler, and Akihiro Hirai for helpful discussions.

Table 2: Initial rule set for the competitive learning chunker.

S1:	A*F	→ B*F
S2:	BD	→ BF
S3:	{D,E,F}*E	→ {A,B,C}*A
S4:	{B,E}B	→ CB
S5:	{A,D}C	→ {C,F}C

Table 3: Chunks formed by the competitive learning chunker.

Chunk	(Component Rules)
EA*F → CB*F	(S1,S4)
ABD → CBF	(S1,S2,S4)
AADF → CBFF	(S1,S2,S1,S4)
BE*E → CB*A	(S3,S4)
DEB → FEB	(S4,S5)

References

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987) Soar: An architecture for general intelligence. *Artificial Intelligence* 33(1):1-64.

Newell, A. (1987) The 1987 William James Lectures: Unified Theories of Cognition. Given at Harvard University.

Rumelhart, D E., and Zipser, D. (1986) Feature discovery by competitive learning. In D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Cambridge, MA: MIT Press.

Touretzky, D. S. (1989a) Chunking in a connectionist network. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pp. 1-8. Hillsdale, NJ: Erlbaum.

Touretzky, D. S. (1989b) Towards a connectionist phonology: the "many maps" approach to sequence manipulation. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pp. 188-195. Hillsdale, NJ: Erlbaum.

Touretzky, D. S., and Wheeler, D. W. (1990) A computational basis for phonology. In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2.* San Mateo, CA: Morgan Kaufmann.