DTIC FILE COPY

(2)

AD-A219 028

RULES AND MAPS IN CONNECTIONIST SYMBOL PROCESSING

Technical Report AIP -79

David S. Touretzky

Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213-3890

The Artificial Intelligence and Psychology Project

Departments of
Computer Science and Psychology
Carnegie Mellon University

Learning Research and Development Center
University of Pittsburgh

DTIC
ELECTE
MAR 14 1990
S B D

Approved for public release; distribution unlimited.

90 03 12 084

(2)

RULES AND MAPS IN CONNECTIONIST SYMBOL PROCESSING

Technical Report AIP -79

David S. Touretzky

Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213-3890

June 1989

DTIC
ELECTE
MAR 14 1990
S B D

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

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 the Office of Naval Research, the National Science Foundation, the U.S. Government, or Hughes Research Laboratories.

Reproduction in whole or in part is permitted for any purpose of the United States Government. Approved for public release; distribution unlimited.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AIP - 79	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Carnegie-Mellon University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Computer Sciences Division Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) Department of Psychology Pittsburgh, Pennsylvania 15213	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, Virginia 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Same as Monitoring Organization 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0678
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS p4000ub201/7-4-86 PROGRAM ELEMENT NO. N/A; PROJECT NO. N/A; TASK NO. N/A; WORK UNIT ACCESSION NO N/A

11. TITLE (Include Security Classification)

Rules and Maps in Connectionist Symbol Processing

12. PERSONAL AUTHOR(S)

Touretzky, David S.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 86Sept15 TO 91Sept14	June, 1989	16

16. SUPPLEMENTARY NOTATION Will appear in the Proceedings of the Eleventh Annual Conference of the Cognitive Science Society, Ann Arbor, August 1989

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

cognitive phonology, connectionist symbol processing
linguistics,
many-maps model, Voice Communication

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

SEE REVERSE SIDE

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Alan L. Meyrowitz	22b. TELEPHONE (Include Area Code) (202) 696-4302 22c. OFFICE SYMBOL N00014

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

SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

ABSTRACT

This report contains two papers to be presented at the Eleventh Annual Conference of the Cognitive Science Society. The first describes a simulation of chunking in a connectionist network. The network applies context-sensitive rewrite rules to strings of symbols as they flow through its input buffer. Chunking is implemented as a form of self-supervised learning using back-propagation. Over time, the network improves its efficiency by replacing simple rule sequences with more complex chunks.

The second paper describes the first implementation of Lakoff's new theory of cognitive phonology. His approach is based on a multilevel representation of utterances to which all rules apply in parallel. Cognitive phonology is free of the rule ordering constraints that make classical generative theories computationally awkward. The connectionist implementation utilizes a novel "many maps" architecture that may explain certain constraints on phonological rules not adequately accounted for by more abstract models.

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Chunking in a Connectionist Network

David S. Touretzky

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Incremental performance improvement with accumulated experience has been measured in human beings for a wide variety of cognitive, perceptual, and motor tasks (Newell, 1987). "Chunking" produces similar performance improvements in symbolic computer programs, such as the SOAR production system (Laird et al., 1987). Chunking takes place in SOAR by observing the working memory trace associated with a sequence of rule firings, and abstracting from this trace a chunk which in the future will produce the same results in a single step.

This paper presents a rule-following connectionist system that also improves its efficiency through chunking. It differs from symbolic production systems in several respects. Although connectionist networks may exhibit rule-following behavior, they do not necessarily contain explicit symbolic rules (Rumelhart & McClelland, 1986; Smolensky, 1988; Pinker & Prince, 1988). The system reported here learns its initial set of behaviors by back propagation from examples. Chunks are then created by a mechanism that observes input/output behavior as the network runs. The chunker is not told which features of the input were responsible for a particular output. In SOAR terminology, it has no access to a working memory trace.

The task the connectionist network is performing is string manipulation based on an abstract version of generative phonology. It was while working on a connectionist approach to phonology that I hypothesized chunking might play a role in the linguistic development of humans. Some speculations on the interaction between a chunker and the Language Acquisition Device appear at the end of this paper.

A Rule-Following Connectionist Network

Figure 1 shows part of a connectionist network that manipulates strings according to context-sensitive rewrite rules. The rewrite rules are an abstract version of classical generative phonology rules, and are shown here using classical notation. Rule R1 below says "change C to E in environments where it precedes a D." Similarly, rule R2 says "change A to B when it precedes an E."

```
R1:  C  -->  E  /  _  D
R2:  A  -->  B  /  _  E
```

Application of R1 to the string ABCD yields ABED. Figure 1 shows how this is accomplished. The input buffer, rule module, and change buffer form a three-layer feed-forward network. Symbols are sequentially shifted into the input buffer. Rule units read the buffer state and generate an output pattern in the change buffer describing the changes that are to be made to the input. (Each input buffer segment has a corresponding change buffer segment.) Three types of changes are possible: *mutation* of input tokens, *deletion* of input tokens, and *insertion* of new tokens. A String Editing Network, not shown, reads the input and change buffer patterns and generates an updated input pattern in which the specified changes have been made. The design of the String Editing Network is explained in (Touretzky, 1989).

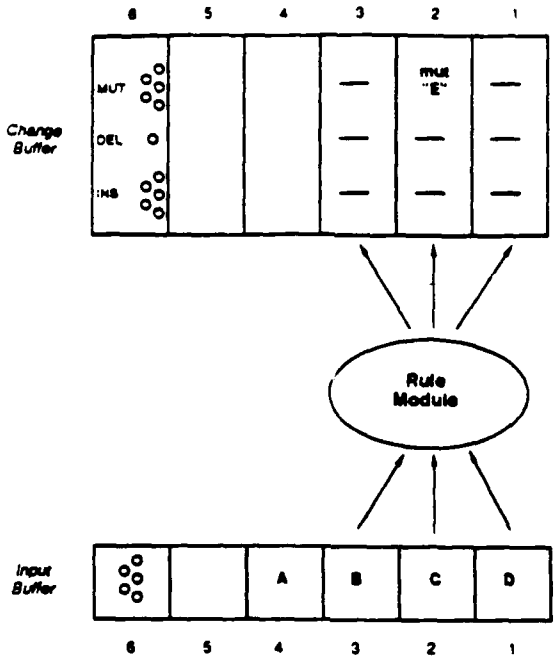

Figure 1: Part of a connectionist network for applying rewrite rules to strings.

The symbols from which strings are composed are binary feature vectors. The experiment reported here uses a representation with five "phonetic" features organized as one group of two features and one of three features. (In real phonology there are many more features; they encode the place and manner of articulation of sounds.) Features within a group are mutually exclusive. There are a total of six legal symbols, labeled A through F. The change buffer patterns use an eleven-element code for each segment: one for signaling deletion, five for describing a mutation, and five for specifying an insertion. Symbols are always inserted to the right of the corresponding input buffer segment.

Change buffer patterns are tri-state: 0 means "no change," +1 means "turn the corresponding bit in the input buffer on," and −1 means "turn the corresponding bit off." For deletion and insertion operations, −1 is treated like zero. The use of tri-state patterns causes the change buffer units to adopt the "no change" case as the default in the absence of input. Tri-state outputs are obtained using the symmetric sigmoid activation function $\sigma(x) = 2/[1 + \exp(-x)] - 1$.

The initial rules are installed by applying backpropagation to a training set of input pattern/change pattern pairs. The rule module serves as the hidden layer during learning. Once the initial rule set has been acquired, there is no supervised learning in the model. To acquire chunks, sequences of typical inputs are run through the input buffer. As it applies its rewrite rules, the model formulates chunks when two rules fire in succession, and trains itself using backprop to predict a chunked action in the appropriate context. Chunking may therefore be regarded as "self-supervised" learning, since the model is serving as its own teacher. The chunking mechanism is explained further after the next section.

Position-Independent Rules

Rules are always learned in "standard position," where the rightmost element of the rule's environment is

the rightmost element of the input buffer. However, downstream feeding relationships may require rules to apply in other positions. Consider what happens when the string ACD is shifted into the input buffer one segment at a time. The network does nothing with the initial substrings A and AC. After shifting in the D, ACD is converted to AED by rule R1. R2 should then apply to produce BED, but the AE environment for R2 is not aligned with the right edge of the input buffer; it is one segment downstream.

To allow rules to apply independent of position, we make several downstream copies of the primary rule module and constrain the link weights in each copy to be equal to the corresponding primary module weights, as shown in Figure 2. This way rules need only be learned in standard position, but they can apply anywhere they are needed. The reason for using a change description as the output representation should now be clear: the outputs of all the rule modules can be superimposed by addition at the change buffer units. If each rule module were to directly map the input string to an updated string, the outputs of multiple rule modules could not be combined.

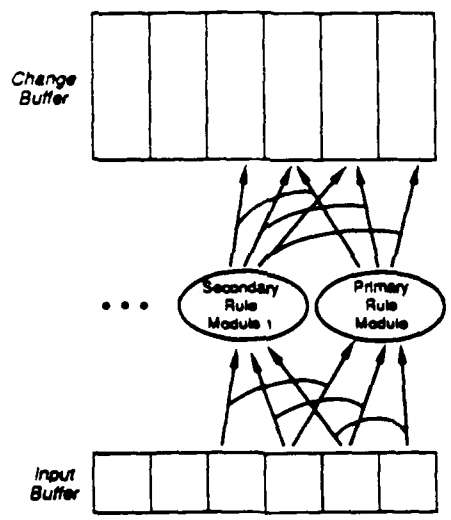

Figure 2: Link-equality constraints cause secondary rule modules to replicate the behavior of the primary module at various positions downstream.

The Chunking Mechanism

Figure 3 shows how chunking is accomplished. The model has two change buffers. The α connections, which control the Current Change Buffer, are created by back propagation learning on an initial training set supplied by the teacher. The β connections control the Chunked Change Buffer, which the network uses to teach itself new chunks.

Chunking occurs continuously as the network processes patterns flowing through its input buffer. Each time a symbol is shifted into the input buffer and a forward pass is performed, the α connections produce a Current Change Buffer pattern. If the pattern is all zeros, meaning no α rule fired, the β connections are taught to produce the same result. If the pattern is non-zero, meaning some α rule did fire, the chunker makes a note of the change buffer pattern, and the string editing network makes the requested change

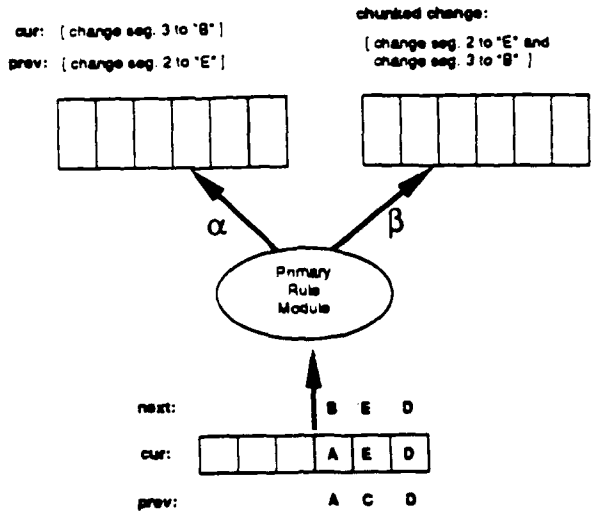

Figure 3: Chunking of rules R1 and R2 by training β connections to produce the composition of the two rules' change buffer patterns.

and updates the input buffer. After a second forward pass, if no more α rules fire, there is no sequence to be chunked. In this case the β connections are taught to imitate the change pattern produced by the single α rule. If an α rule does fire on the second forward pass, a chunk can be composed from the remembered change buffer pattern of the first rule plus the change buffer pattern of the second rule. The β connections are then taught to output this composite change pattern in the context that caused the first α rule to fire.

This training regimen ensures that the β rules will be an essential superset of the α rules. The only α rules not duplicated on the β side will be those that never fire in isolation, but only to feed another rule or as a result of a feeding rule. These non-essential α rules will be replaced by chunks. More commonly, chunked and unchunked versions of rules coexist on the β side.

A number of fine points in the training of the model need to be explained. In a chunking network the connections to rule and change buffer units should remain plastic. Plasticity can be lost if units are allowed to get too far out on the tails of the sigmoid, where the derivative goes to zero. Several steps are taken to prevent loss of plasticity. In standard back propagation the error signal of an output unit is defined to be the difference between the actual and desired outputs multiplied by the derivative of the output function (Rumelhart et al., 1986). In the chunking network the derivative term is omitted for output units.

In addition, weights must not be allowed to grow too large during training, as this can also hinder future learning. To keep the weights small and the rule units from getting too far out on the tails of the sigmoid, the model uses output training targets of +0.5 and −0.5 rather than +1 and −1. When updating the input buffer, any change buffer value greater than +0.3 is treated as +1, and any value less than −0.3 is treated as −1.

Although change buffer units use a symmetric sigmoid, rule units use the standard sigmoid. I conjecture

that rules may be learned more easily this way. Rule units are feature detectors, so when a feature is not present the unit's output should be zero. This is easily achieved with the standard sigmoid by supplying a substantial negative bias that can be counteracted only by an appropriate pattern of input features. With tri-state units it is not possible hold the output steady at zero over the entire set of inputs that aren't supposed to trigger a rule.

Finally, it should be noted that in order to learn the environments in which new chunks apply, rule units must modify not only their β output connections, but also their input connections. But this alters the rule unit's response to subsequent inputs, so it may interfere with the continued production of correct patterns in the Current Change Buffer. To prevent the model from leading itself astray, it is programmed to continually rehearse its α behaviors as it trains the β connections. Rehearsal is another instance of self-supervised learning. Each pattern the α units generate in the Current Change Buffer is "idealized" by treating all values greater than +0.3 as +0.5, all values less than −0.3 as −0.5, and all other values as 0. The difference between the actual α outputs and the idealized outputs generates an error signal that helps to readjust the input weights on each presentation, countering the disruptive effect training the β units has on the input weight pattern. The α and β sides of the model are thereby forced to compromise on an input weight pattern that allows each side to do its job.

Complex Rule Interactions

Composing a chunk from two mutation rules is easy: one simply inclusive-or's the change buffer patterns (using tri-state logic), giving the second rule priority in the case of a +1/ − 1 conflict. Composing chunks from other types of rules is slightly more complex. If the first rule inserts or deletes a segment, some portion of the second rule's change buffer pattern will need to be shifted to take this change into account before inclusive-oring the two together. If the second rule mutates a segment that was inserted by the first, the second rule's mutation pattern must be combined (with priority) with the first rule's insert pattern, not its mutation pattern. If the second rule deletes a segment that was inserted by the first rule, the first rule's insertion must be suppressed in the composed chunk. This can be accomplished by setting the insertion bits to zero.

In the simulation, chunked change buffer patterns were composed by a Lisp version of the above algorithm. However, it would be easy to construct a connectionist network to do the same task. The input would be the current and previous change buffer patterns; the output would be the composed change.

A limitation of this particular rewrite-rule architecture is that only one symbol can be inserted between each pair of symbols in the input buffer. Therefore one cannot chunk two rules if they both insert something at the same input position. In practice this situation does not seem to come up in segmental phonology, although there are multi-segment insertions at the morphological level.

Experimental Results

The initial chunker simulation used an input buffer of length six, and three rule modules, each of which looked at three adjacent input segments. The primary rule module was taught rules R1 and R2 by backpropagation on a small training set. (The training set consisted of some environments in which the rules should apply, plus some additional environments in which no rule should fire.) The following example shows the results of this training. R2 and then R1 applies, independently, in standard position, as the string AEFCD is shifted through the input buffer. Underscores denote null segments (all zeros.)

```
* (demo '(a e f c d))
    Shift A into input buffer:      _ _ _ _ _ A
    Shift E into input buffer:      _ _ _ _ A E
      Change due to rule firing:    _ _ _ _ B E        (rule R2)
    Shift F into input buffer:      _ _ _ B E F
    Shift C into input buffer:      _ _ B E F C
    Shift D into input buffer:      _ B E F C D
      Change due to rule firing:    _ B E F E D        (rule R1)
```

We next consider an example of downstream feeding of R2 by R1, which never occurred in the training data. Note that after the last symbol is shifted in, the input buffer changes twice. This is the condition allowing a chunk to be composed.

```
* (demo '(a c d))
    Shift A into input buffer:      _ _ _ _ _ A
    Shift C into input buffer:      _ _ _ _ A C
    Shift D into input buffer:      _ _ _ A C D
      Change due to rule firing:    _ _ _ A E D        (rule R1)
      Change due to rule firing:    _ _ _ B E D        (rule R2)
```

Running the network on sequences such as ACD allows it to learn chunks in self-supervised mode, by observing its own behavior. The chunk for turning ACD into BED consists of R1 plus a shifted version of R2, since R2 is applying one segment downstream. The rule units must learn to pay attention to the third segment of the buffer, whereas for R1 and R2 in isolation only the first two segments are important.

The result of chunking is shown below for the string ACDCD. (To actually use the learned chunks we replace the α weights with the learned β weights.) The ACD to BED portion of the example demonstrates the existence of the R1-R2 chunk; the CD to ED portion that follows demonstrates the preservation of R1 on the β side as an independent rule. Other inputs verified that R2 was also preserved.

```
* (demo '(a c d c d))
    Shift A into input buffer:      _ _ _ _ _ A
    Shift C into input buffer:      _ _ _ _ A C
    Shift D into input buffer:      _ _ _ A C D
      Change due to rule firing:    _ _ _ B E D        (chunk R1-R2)
    Shift C into input buffer:      _ _ B E D C
    Shift D into input buffer:      _ B E D C D
      Change due to rule firing:    _ B E D E D        (rule R1)
```

Additional experiments confirm that the network can chunk insertion and deletion rules as well as mutations. It can also combine a learned chunk with another rule to form a bigger chunk.

As long as the model's behavior is governed solely by the α connections, it will not be able to apply the chunks it has learned. An initial, brute-force solution to this problem is to simply copy the β weights to the α connections whenever the β training error is low enough. But such a drastic, global weight change

is admittedly unnatural. We are currently exploring more fluid ways of exchanging knowledge between the α and β sides. One scheme we have tried is to maintain running confidence levels for each side, and with each new input symbol, stochastically choose either the α or β change-buffer pattern based on relative confidence values. Initially the β confidence is low. When the α side has successfully trained the β side, the network begins to execute a mix of α and β actions, including some learned chunks. As the β side in turn tries to teach new chunks to the α side, the α confidence level drops and the β rules take over until the new α chunks have been learned.

Interesting Chunking Phenomena

A number of interesting questions are raised by this work. One is the order in which larger chunks should be formed. Consider the feeding rule chain R1-R2-R3-R4. If the model builds at most one chunk before shifting a new symbol into its buffer, the chain will be chunked in the order (((R1 R2) R3) R4). This approach is compatible with the power law of practice cited by Newell. If the model builds a chunk whenever any pair of unchunked rules fire in sequence, the order of chunk creation will be ((R1 R2) (R3 R4)). It is not yet known which order is more compatible with the way the learning algorithm creates rule representations.

A second question is what representation the model will develop for rules that participate in multiple feeding chains. Consider a case where, for one class of inputs there is a chunk R1-R2-R3, and for another class a chunk R1-R4-R5. Since R1 is shared by both chunks and may also apply in isolation, the representations of the two chunks and the original rule should be similar, and will probably share units.

A related issue is the formation of variable-length chunks from self-feeding rules, such as this deletion rule: `R6: E --> Ø / _ F`

R6 applies three times in succession to the string BEEEF to derive BF. After chunking, BF should be obtained in a single rule firing. If the chunker is exposed to sequences of form $\{E\}^{+}F$ of varying length, it should build a collection of related chunks. The degree and nature of the overlap in representations of these chunks is worth investigating.

Finally there is the issue of variables appearing in rules. Variables serve either to narrow the domain of application of a rule (when the same variable appears twice on the left hand side), or to copy a value from one place to another (when the variable appears once on the left and at least once on the right hand side.) In phonology it is not too expensive to expand a variable-containing rule into a set of variable-free rules, because variables can take on only a few values. In more general symbol processing tasks this may not be feasible. It may be possible to teach a backpropagation network to implement rules with variables by encoding the value in the hidden layer activation pattern. Such a scheme would probably require a more complex hidden layer than in the present model.

Chunking and Language

The segmental phonology of any human language can be expressed by sequences of simple rewrite rules on strings. These rules are highly constrained, so that, for example, reversing the segments of a word is not possible in human phonology (Pinker & Prince, 1988). Another constraint is that there is no metathesis (switching) of non-adjacent segments. The regularity and degree of constraint of phonological processes is striking, and cries out for scientific explanation. The hypothesis motivating the work reported

here is that the Language Acquisition Device may only be able to hypothesize very simple rules. The rules can interact to produce lengthy derivations, and they are extensively chunked during development to arrive at adult linguistic performance. But chunking is the only source of complex rules; they cannot be created *de novo* by the LAD.

Why have rules at all in a connectionist theory? Rules separate *policy* (what Chomsky calls linguistic "switch settings") from *mechanism* (the fundamental ability to do insertions, deletions, and mutations.) If a mechanism such as the String Editing Network is universal and genetically determined, then the LAD's job is tremendously easier: it can concentrate on learning just the policies of the speaker's language.

This paper makes no assumption that policies require explicit symbolic representations in speakers' heads. Rather, it shows that chunking can occur even when there is no working memory trace available and new rules cannot be constructed symbolically. The connectionist chunker acquires its rules incrementally, through self-supervised backpropagation and rehearsal of prior knowledge. Further experiments are planned to analyze the representations the chunker develops.

Acknowledgments

This work was supported by a contract with Hughes Research Laboratories, by National Science Foundation Grant EET-8716324, and by the Office of Naval Research under contract number N00014-86-K-0678. I thank Deirdre Wheeler, Marco Zagha, Michael Witbrock, and George Lakoff for discussions that contributed to this work, and Gillette Elvgren for help with the simulations.

References

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

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

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*. MIT Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland and (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press.

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

Smolensky, P. (1988) On the hypotheses underlying connectionism. *Behavioral and Brain Sciences* 11(1).

Touretzky, D. S. (1989) Towards a connectionist phonology: the "many maps" approach to sequence manipulation. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates.

Towards a Connectionist Phonology: The "Many Maps" Approach to Sequence Manipulation

David S. Touretzky

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract: Lakoff's new theory of cognitive phonology appears to be free of the rule ordering constraints that make generative rules computationally awkward. It uses a multilevel representation for utterances, to which multiple rules may apply in parallel. This paper presents the first implementation of Lakoff's proposal, based on a novel "many maps" architecture. The architecture may also explain certain constraints on phonological rules that are not adequately accounted for by more abstract models.

Linguists established long ago the value of describing phonological processes in terms of formal symbolic rules, but they have steadfastly refrained from speculating about the nature of representations in speakers' heads. Rumelhart and McClelland (1986) argue against the neuropsychological reality of rules. Pinker and Prince (1988) offer persuasive counterarguments. The current reaction against rule-based accounts of low-level cognitive phenomena, phonology in particular, is no doubt strengthened by the computational awkwardness of classical generative phonological rules. Constraints on their order of application force the rules to act sequentially and, in some cases, cyclically. In contrast, Rumelhart and McClelland's PDP model of the phonology of English past tense formation maps input patterns to output patterns directly, in one parallel step. It is, despite its weaknesses, computationally sleek.

In 1988 George Lakoff published a new theory of "cognitive phonology" in which parallel rules apply everywhere simultaneously (Lakoff, 1988a, 1988b). Cognitive phonology is therefore free of the cycles and rule ordering constraints that mar earlier, generative theories. Lakoff described his theory as founded on connectionist principles, but did not specify how it should be implemented. The solution is non-obvious, because cognitive phonology relies on a multi-level mapping representation in which insertions, deletions, and mutations all take place at once.

This paper presents the first working implementation of Lakoff's theory. It uses a novel "many maps" architecture to manipulate sequences of phonemes at multiple levels, and to support abstractions such as the "vowel tier" required by some rules. I will begin by reviewing Lakoff's analysis of a Mohawk problem posed in (Halle & Clements, 1983), and then show how the "many maps" model implements Lakoff's solution. Finally I address the question of why one would want to have rules in a connectionist model. I will argue that the simplicity and highly constrained nature of phonology may be a consequence of humans' using a sequence manipulation architecture similar to the one described here.

Six Rules for Mohawk Speakers

Halle and Clements give six generative rules for deriving the Mohawk word /yɔkrege?/. ("I will push it") from its underlying form /ye + ʌk + hrek + ?/. (It may aid understanding to look at Lakoff's solution first; see Figure 1.) We will consider four of these rules here:

Epenthesis: $\emptyset$ --> e / C _ ? #
Stress: V --> [+stress] / _ C_0 V C_0 #
Vowel omission: V --> $\emptyset$ / _ + V
Intervocalic voicing: C --> [+voice] / V _ V

The epenthesis rule inserts /e/ between a consonant (/k/ in this example) and a word-final glottal stop /?/. The stress rule assigns stress to the penultimate vowel in a word. Notice that in the example /ʌ́/ is penultimate in underlying form but antepenultimate at the surface, due to epenthesis. Thus, the stress assignment rule must be applied prior to epenthesis in order to stress the correct vowel. The intervocalic voicing rule voices a consonant if it appears between two vowels; it changes /eke/ to /ege/. But the second /e/ was inserted by epenthesis; therefore intervocalic voicing must not be applied until after epenthesis. The vowel omission rule deletes the leftmost /e/ in the underlying form, since it precedes another vowel. Evidence from other Mohawk words shows that vowel omission applies before stress assignment. (If it didn't, the rules could perhaps assign stress to a vowel and then delete it, leaving no vowel stressed.)

In the classical account these four rules are totally ordered: vowel omission precedes stress assignment, which precedes epenthesis, which precedes intervocalic voicing. Each of these rewrite rules modifies the "current" derivation, producing a new one. When all the rules have applied, what's left is the surface form of the word.

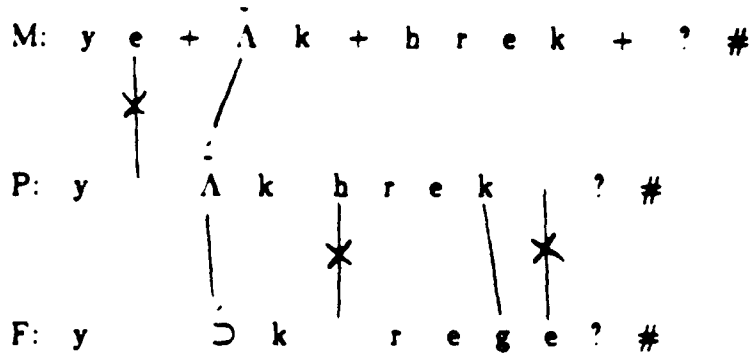

Figure 1: Lakoff's cognitive phonology derivation of the Mohawk word "I will push it."

Lakoff's analysis replaces the sequential rewrite rules with mapping constraints that all apply in parallel. There are three levels of representation: M (morphemic), P (phonemic), and F (phonetic). Sequences at M level are by default simply copied to P level. But M-P constraints can alter the mapping, causing changes in the P-level representation. Intra-level constraints may also affect the representation at P level. The combined effect of M-P and P-level constraints can be seen in the middle line of Figure 1: the first /e/ has been deleted and the penultimate vowel has been stressed. A second mapping takes P-level representations to F-level representations via a combination of P-F and F constraints. At F level we see that the epenthetic /e/ has been inserted and, consequently, the /k/ has been voiced.

Lakoff's solution elegantly answers a number of phonological questions which, unfortunately, we cannot afford to raise here. Elegance aside, though, its implementation in connectionist hardware is problem-

atic. The major problems that arise are: how to efficiently implement insertion, deletion, and mutation operations when several occur in parallel; how associations between corresponding segments at different levels can be maintained, since levels may have varying numbers of elements; and how rules can apply everywhere at once in the input buffer. The "many maps" architecture provides solutions to these problems.

How to Build a Map

As a prelude to discussing the full "many maps" implementation I will describe the workings of a single map. Figure 2 shows the P-level map in the context of the Mohawk example. The input to this map comes from two buffers: M-level and P-deriv (P derivation). The output is the P-level representation of the utterance. The M-level buffer, which is read-only, contains the underlying form of the utterance. Segments are shifted into the buffer from the right, and discarded when they reach the left edge. M-level segments are by default mapped to identical segments at P-level. However, each M-level segment has a slot in P-deriv for describing changes that can be made to it if some rule requests. Three types of changes are supported: mutation, deletion, and insertion. Deletion of an M-level segment means blocking its appearance at P-level. Mutation maps the segment to a segment with slightly different features at P-level. Insertion causes a new segment to appear at P-level to the right of the M-level segment. (Insertion to the left could also be supported, but was omitted to simplify the simulation.) In the figure, the M-level /e/ is marked in P-deriv for deletion, and the /ʌ̄/ is to be mutated by adding stress. Thus the M-level sequence /yeʌk/ appears as /yʌ̄k/ at P-level.

The upper-diagonal matrix in the figure represents an array of connectionist mapping units. When one of these units is active (shown by a segment appearing inside it), the segment in that input column is copied to the corresponding output row. At the same time, any mutations to the segment that were requested in P-deriv are made.

The units in the mapping matrix are subject to lateral inhibition. At most one unit can be on in each row and each column. The inhibition is asymmetric, so that when choosing which row an input segment should map to, the model prefers to fill higher rows first. In addition, when choosing which segment should appear in a row, the model prefers to select the rightmost segment available. This ensures that the ordering of M-level segments is preserved at P-level, and that the P-level representation always appears right-justified in the buffer with no gaps where M-level segments are deleted, and no collisions where new segments are inserted.

Consider first the fate of the M-level /k/. The active square in the first row of the matrix shows that this segment is mapped to the rightmost position in the P-level buffer. Since this unit is fully active, no other unit can come on in that row or in that column. The /ʌ̄/ is mapped to the second row; simultaneously it is stressed, as specified in the mutation part of P-deriv. The /e/ is marked for deletion in P-deriv. Deletion is accomplished by inhibiting all the units in that column of the matrix, thereby preventing the segment from mapping to any row of P level. Thus the /y/, which is the fourth M-level segment counting from the right, appears as the third segment at P-level. The M-P mapping is computed in parallel (in fact, in constant time), independent of the number of segments in the buffer or the number of insertions and deletions to be performed.

Note that M-level segments are positioned over every other mapping column. The intervening columns are reserved for insertions. If an insertion is specified in P-deriv, the segment to be inserted will be mapped to the next available row, just as an M-level segment would be.

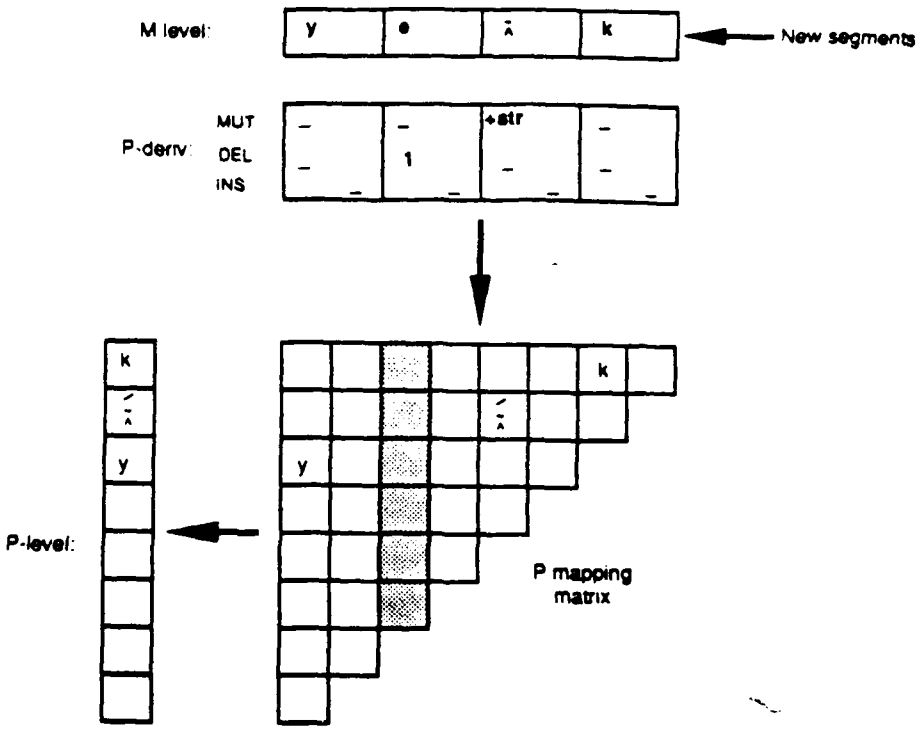

Figure 2: The map that derives P-level representations.

How Rules Work

One of the strengths of cognitive phonology is that rules may locate their environments at one level and their actions at another. Thus the application of an M-P rule does not affect the environments of other M-P rules. It does, however, affect the environments of P and P-F rules.

M-P rules are implemented by connectionist units that take their inputs from the M-level buffer and have output connections to P-deriv. After a new segment is shifted into M-level, M-P rules may cause some P-deriv units to change state, thereby recording a change the rules wish to make in the mapping. P-deriv units maintain their states indefinitely unless disturbed by rule units, thereby serving as a memory of accumulated changes. Each time P-deriv is modified by some M-P rule, the mapping matrix re-derives the P-level representation from the M-level and P-deriv buffers. When the M-level buffer is shifted left to accomodate the next incoming segment, the contents of P-deriv are also shifted left to maintain registration with the M level.

Pure P-level rules are trickier to implement than M-P rules, because they take their inputs from the *output* of the mapper. For example, suppose a purely P-level rule wanted to devoice the /y/ in Figure 2. This segment appears at position three at P level, but it is in position four at M level due to the deletion of a preceding segment. In order for P-level rules to record their changes in the correct P-deriv segment, they must invert the M-P mapping to align their changes with the M-level segments. The circuitry for this is straightforward. The state of the mapping matrix used to produce the current P-level representation provides the necessary information to invert the map.

Since P rules apply to their own outputs, they can feed each other, and there is even a possibility of long rule chains. Here is a simple example. In this implementation of Lakoff's Mohawk solution, epenthesis and intervocalic voicing are both implemented as F-level rules. Even though the rules are unordered, the former feeds the latter, so they will have to fire sequentially. The existence of rule chains appears to prevent the sort of parallel processing that cognitive phonology strives for. However, chunking can be used to automatically collapse a chain of intra-level rules into one complex rule, and thus regain the parallelism. This has been demonstrated for abstract phonological rules in (Touretzky, 1989).

Rules cannot be tied to fixed buffer positions because of feeding relationships. Suppose the intervocalic voicing rule were aligned with the right edge of the buffer, i.e., it looked at the rightmost three segments. When it saw /eke/ it would produce /ege/. But if the buffer initially holds /ek?/, epenthesis will produce /eke?/, and so the first appearance of the /eke/ fragment will not be aligned with the right edge of the buffer. It will be "downstream" of its standard position.

To make rules position-independent, we hypothesize that all rules are independently motivated and hence can be learned in standard (right-aligned) position by a primary rule module. Secondary rule modules are introduced at successive positions downstream. Their input and output connections are forced by link equality constraints to mimic the behavior of the primary module. All the rule modules operate in parallel, and their requests for changes are combined and recorded in P-deriv. In this way we achieve position-independence without having to supply examples of every rule firing in every position.

The "Many Maps" Architecture

At a minimum, cognitive phonology requires two maps: one for P-level representations and one for F-level. The F-level map is similar to the P-level discussed above, except it takes input from three buffers: M-level, P-deriv, and F-deriv. F-level representations are derived directly from M-level by merging the P-deriv and F-deriv changes at the input to the F mapping matrix. In the case of conflict, F-deriv changes are given priority. See Figure 3. This approach allows the model's multiple maps to operate independently instead of increasing the latency with each new level of representation. (This idea is due to Gillette Elvgren.)

Two types of rules influence the contents of F level. P-F rules have their environments at P and their actions at F; their actions are recorded in F-deriv by first inverting the mapping specified by the P matrix to align them properly with M-level segments. (The M-level segments and P-deriv and F-deriv changes are all kept in strict registration.) Since F-level rules have their environment at F, their actions are recorded in F-deriv by inverting the mapping specified by the F matrix. Figure 4 shows the relationships between the various rule types.

In Mohawk, the stress rule is most easily implemented by placing its environment at yet another level: a P-level vowel tier containing only vowels and word boundary markers. This allows the stress rule to look for the pattern VV# in the vowel tier and stress the penultimate vowel. (This solution was suggested by Deirdre Wheeler. Other evidence for an independent vowel tier is cited in (Goldsmith, 1989).) The map that extracts the P vowel tier takes inputs from the same M-level and P-deriv buffers as the regular P-level map, but only vowels appear in its output; consonants are left unmapped. The P vowel tier map operates completely in parallel with the regular P-level map.

We have successfully applied the "many maps" architecture to additional examples Lakoff chose from Slovak, Gidabal, and Lardil. Other languages will require other specialized maps. We expect, though,

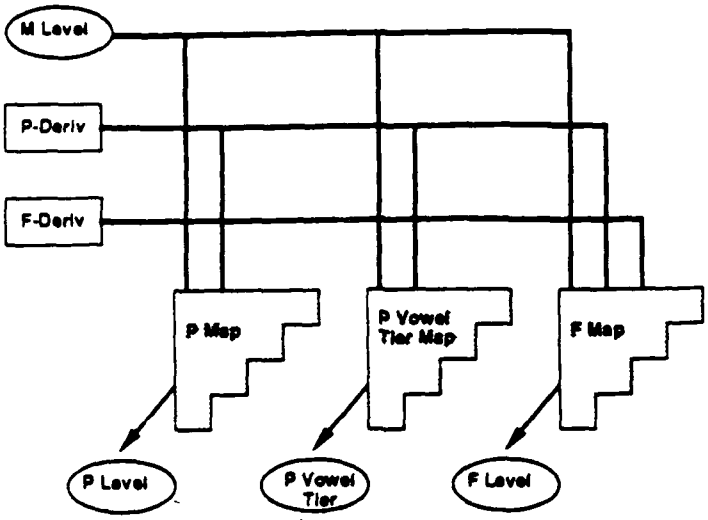

Figure 3: Inputs and outputs of the P-level, F-level, and P-vowel-tier maps. All maps operate independently, and in parallel.

that these can all be built from similar hardware. Perhaps language learners are born with a collection of such maps at their disposal, which are then trained to extract whatever features are *salient in the linguistic* environment.

Discussion

Phonology continues to be a rich and promising domain for connectionist investigations of language. It is simpler and less plagued by the special cases and exceptions that complicate syntax and morphology, so there is a better chance of finding a complete solution. Another advantage of phonology is its quasi-linear structure, which facilitates experimentation with parallel distributed processing techniques. The PDP approach isn't currently as well suited to manipulating hierarchical structures such as syntactic trees.[1]

The present model is not without limitations. It deals only with segmental phonology; no attempt is made to include morphology. (In contrast, the Rumelhart and McClelland verb learning model combines morphological and phonological processing in a single layer of weights.) Also, currently the model does not represent syllable structure. Certain types of phonological rules therefore cannot be expressed. This is an area where further work is in order.

The mapping architecture does not permit more than one segment to be inserted between segments adjacent at the previous level. Morphology sometimes requires multi-segment insertions, but it appears that phonology does not. If this observation holds true, it is a significant constraint on phonological machinery. The model provides an achitectural explanation for it, unlike more abstract phonological models which ignore implementation issues. Finally, the mapping matrix does not support metathesis (switching of segments) as a primitive operation. Considering the controversial and still unresolved status of metathesis in linguistic theory, we are in no rush to add it.

[1]However, Touretzky (1986), Hinton (1988), and Pollack (1988) offer some hope for handling hierarchical structures.

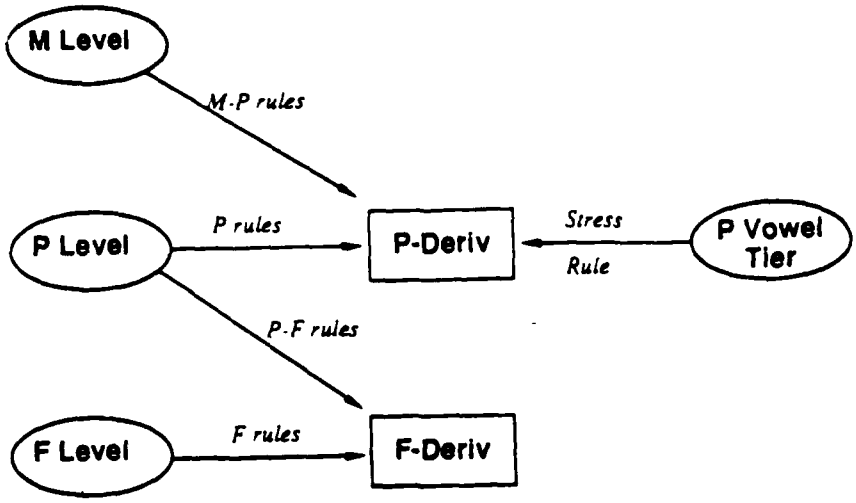

Figure 4: The different types of cognitive phonology rules that relate representations at various levels.

The "many maps" model might be improved by switching to an autosegmental representation with separate phonemic, skeletal, and tonal tiers, as in (Goldsmith, 1989). In fact, the original inspiration for the mapping matrix came from wondering how inter-tier association lines in autosegmental phonology could be represented in a connectionist network. A mapping matrix representation seems particularly appropriate for inter-tier rules, such as the tone-shift rules Goldsmith describes in various Bantu languages.

On the Reality of Rules

Why should human phonology be so regular and tightly constrained? It is amazing that this level of language can be described by classical generative rules which affect only a single segment each. This mode of description is effective, but it remains computationally inelegant. On the other hand, as Pinker and Prince point out, a connectionist architecture that directly maps input sequences to output sequences can perform outlandish transformations never seen in human language, such as reversing all the phonemes of a word.

There appear to be more modest sorts of transforms that are absent from the human repertoire. For example, no language methathesizes non-adjacent segments. Consonants are never changed to vowels, and vice versa. And harmony and assimilation phenomena always spread features from one edge of a cluster to the other, never from the interior outward. To be successful, a connectionist theory of phonology should motivate these constraints by providing computational explanations for them.

We can begin to account for constraints on phonology by adopting a universal, genetically-specified sequence manipulation machine that, like the "many maps" model, operates in parallel but can perform only a limited set of transformations. The function of linguistic rules is to operate this machine — to "press the right buttons at the right times." A speaker's linguistic knowledge does not directly modify sound sequences as in the Rumelhart and McClelland model; it modifies sequences only indirectly, by controlling this built-in machinery.

An input representation plus a discrete set of symbol manipulation primitives defines a *rule system*. If such a system underlies human phonology, then even if speakers do not have symbolic rule representations in their heads, they truly do use rules, as opposed to merely saying their behavior can be described by rules. Classical phonology concerns itself with the regularities of this rule system. Connectionist phonology attempts to ground the system in the design of the sequence manipulation machine, for it is from there that the rule system emerges.

Acknowledgements

I am most grateful to George Lakoff for sharing his work on cognitive phonology with me, and for taking the time to answer so many questions. I also thank Deirdre Wheeler for stimulating conversations, and for the linguistic guidance she has provided during our weekly meetings. Gillette Elvgren programmed the Mohawk simulation, and has made valuable refinements to the mapping architecture.

This work was sponsored 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.

References

Goldsmith, J. (1989) *Autosegmental and Metrical Phonology: A New Synthesis*. Basil Blackwell.

Halle, M., & Clements, G. N. (1983) *Problem Book in Phonology*. MIT Press.

Hinton, G. E. (1988) Representing part-whole hierarchies in connectionist networks. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, pp. 48-54.

Lakoff, G. (1988a) A suggestion for a linguistics with connectionist foundations. In D. Touretzky, G. Hinton, and T. Sejnowski (eds.), *Proceedings of the 1988 Connectionist Models Summer School*. Morgan Kaufmann Publishers.

Lakoff, G. (1988b) Cognitive phonology. Manuscript draft; presented at the LSA annual meeting.

Pinker, S., & 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*. MIT Press.

Pollack, J. (1988) Recursive auto-associative memory: devising compositional distributed represesnations. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, pp. 33-39

Rumelhart, D. E., & McClelland, J. L. (1986) On learning the past tenses of English verbs. In J. L. McClelland & D. E. Rumelhart (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press.

Touretzky, D. S. (1986) BoltzCONS: reconciling connectionism with the recursive nature of stacks and trees. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 522-530.

Touretzky, D. S. (1989) Chunking in a connectionist network. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates.