A Connectionist Context-Free Parser Which is not Context-Free, But Then It is ot Really Connectionist Either[1]

Eugene Charniak

Eugene Santos

Department of Computer Science
Brown University
Providence, RI 02912

ABSTRACT

We present a distributed connectionist architecture for parsing context free grammars. It improves earlier attempts in that it is not limited to parse trees of fixed width and height (i.e. fixed length sentences). The memory limitations inherent in connectionist architectures comes out in an inability to parse center-embedded sentences.

Key Words: connectionist parsing, distributed connectionism, context-free grammars.

1. Introduction

This paper describes a context-free parser designed in a "connectionist" architecture.

Connectionism has attracted considerable attention of late because it offers a new way of looking at old problems in Artificial Intelligence and Psychology - a way which also has considerable neurophysiological plausibility. Unfortunately, connectionism, or at least the "distributed" connectionism we will assume in this paper, has been hard to apply to high-level cognitive tasks. Connectionist parsing has been of interest because of parsing's intermediate role between higher and lower cognitive abilities.

Indeed, there have been several previous attempts at a connectionist context-free parser [1,2,3]. These, like the present parser, have a major failing – they are strictly speaking, only capable of parsing regular grammars. This is inevitable. Connectionism assumes a large, but bounded, number of units. Thus any connectionist scheme must be finite state.

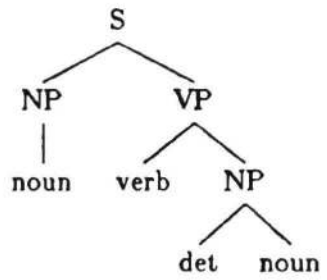

Figure 1. Parse tree on a white-board

[1]This research was supported in part by the Office of Naval Research under contract N00014-79-C-0592, the National Science Foundation under contracts IST-8416034 and IST-8515005. and by the Defense Advanced Research Projects Agency under ARPA Order No.4786. Thanks to James McClelland for some encouragement.

Nevertheless one can ask if such parsers could approach context-free in the limit (as the number of units is increased) and since the answer for all is "yes" it seems reasonable to call them all (including ours) "context-free."

More interesting though is to ask how the memory limitations show up in the parsing process. The failings of previous parsers have been distinctly "un-human." They are limited to fixed length sentences. Ours is more promising in this regard. While it does fine for right branching structures, it is limited in its ability to parse center embedded constructions, a point we will return to later.

Ours does, however, have a distinct failing of its own – it is not a "true" connectionist architecture. But let us say what it *is* first.

2. Representing a Parse Tree

The easiest way to visualize the parser, and how it represents a parse tree, is to imagine a parse tree drawn on a white board, as shown in Figure 1. Naturally since we are talking about a distributed connectionist scheme, we will break the white board up into many distinct units, each one of which may have one of a small number of distinct values - S, NP, VP, PP, Noun, etc. This is shown in Figure 2.

The scheme shown in Figure 2 has several representational inadequacies. For one, the representation does not indicate which constituents dominates which and as such does not distinguish the two PP attachments in Figure 3. Furthermore the same parse tree can have many distinct representations. Figure 4 denotes the same tree as Figure 2, but is quite different.

Thus we indicate a parse tree by including the entire path, from leaf to root, in every column as shown in Figure 5. This means that a single constituent, for example, the S in Figure 5. will be represented as several units. Nothing indicates that these all denote the same S constituent. There are

	S		
NP	VP		
		NP	
noun	verb	det	noun

Figure 2. Segmented parse-tree array

		S			
NP			VP		
			NP		
				PP	
					NP
noun	verb	det	noun	prep	noun

Figure 3. Ambiguous parse array

S			
NP		VP	
			NP
noun	verb	det	noun

Figure 4. Another representation of Figure 2

		S	S
S	S	VP	VP
NP	VP	NP	NP
noun	verb	det	noun

Figure 5. Tree-path parse array

different ways this could be handled. The one we have chosen is to introduce "binding" units. Each unit denoting a non-terminal (or the lack thereof, represented internally by the symbol e) has a sister unit which states which, if any, of the units in the column to the right is the same constituent. In our figures we will use lines to indicate such bindings. See Figure 6. Internally these are values from 0 to the height of the array plus a special value, b, for a "boundary" indicating that we have a right boundary of a constituent and thus there is no corresponding constituent to the right.

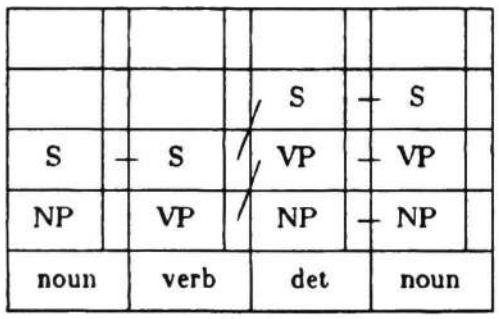

Figure 6. Actual representation of a parse

As has been implicit in our diagrams so far, the actual input to the parser is not words, but rather their parts of speech, since that is all that is relevant to our task. Thus parts of speech are the terminal symbols in the grammar. We have distinguished the bottom row of the parser, where the parts of speech appear, from the other rows, since all of the others have non-terminals only. Also the terminal row does not have binding units since we do not admit discontinuous terminal constituents. In what follows we will pretty much ignore the terminal units. So, for example, most rules which apply to non-terminals have special cases for terminals. These are ignored.

3. How the Parser Works

The basic idea is that words (really parts of speech) are read into the parser on the lower right and then shifted to the left as each new word comes in. Thus the columns are numbered from 0 to N starting from the right. See Figure 7. Initially all NT units are set to e, and all B units are set to b. At the first word the lower right unit, TR(0), then has its value "clamped" to, say, noun, and all of the NT(i,j) units, recompute their values synchronously (that is, all based upon the values in the previous iteration). Then the B(i,j)'s compute their values (using the new NT values but the old B values). This repeated is for a total of five times before the next word is read in. (Five iterations was chosen arbitrarily. In fact, it seems likely that a much lower number, like two or three, would work as well. It is one of many things we have not yet had time to test.) Figure 7 shows the starting configuration and that after three iterations. Since a unit will typically have some probability for several different values, each unit in Figure 7 has four values for it. If all four are the same it means that value has probability $> .75$. Three the same indicate a probability $> .5$, etc. This is true for lines as well, but we only indicate the most probable to reduce the jumble. In point of fact, however, all the lines in all of the examples go the same place. At this point the entire network is shifted to the left, leaving NT(0,j) = e and TR(0) is clamped to the next part of speech. Figure 8 shows a noun and verb following on the heals of the input in Figure 7. This processing, and all of the other examples, have been done using the following grammar:

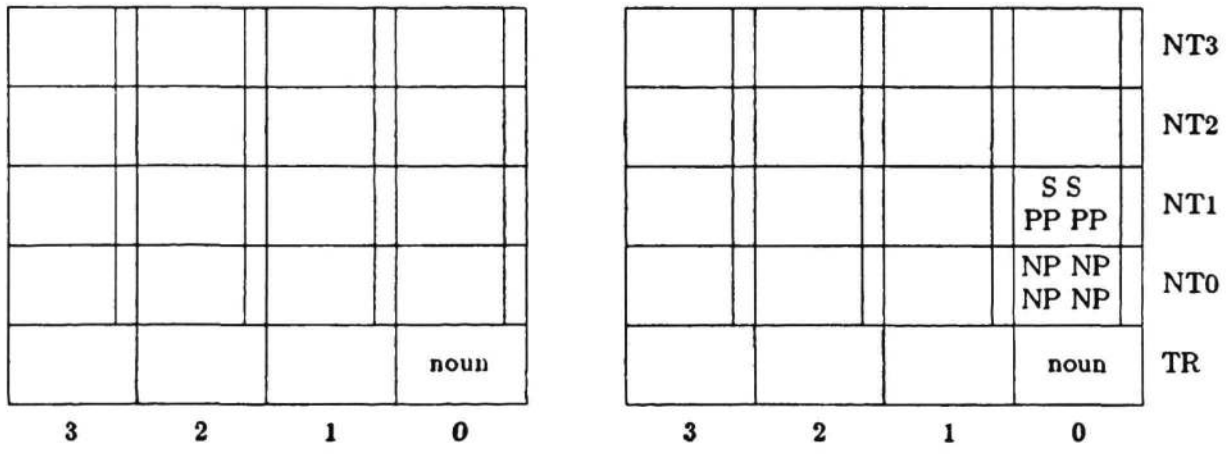

Figure 7. Start of sentence 'noun verb noun'

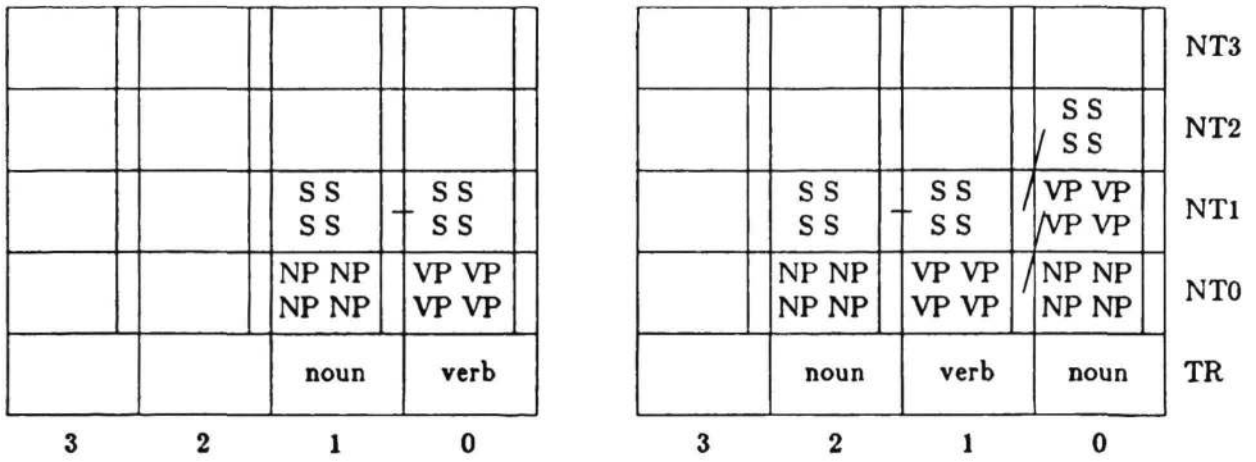

Figure 8. End of sentence 'noun verb noun'

S → NP VP
VP → verb NP PP*
PP → prep NP
NP → (det) noun PP*
NP → pronoun
NP → propnoun.

Shifting words left (and eventually out the left-hand side) allows the parser to handle sentences of unbounded length, at least in principle. Since such shifting is not within the traditional connectionist repertoire, it makes our architecture slightly suspect to a "traditionalist." It also means that sentences longer than the parser's width will not be completely represented at any one time. In such cases we take the total parse tree to be the "obvious" combination of the trees created during the parse.

4. Rules

Next we describe the actual rules applied at each unit. These will take us still further from connectionist orthodoxy because we allow each unit calculation to be much more complex than the typical summing plus threshold. In particular a unit decides what value to adopt by summing the influence of several sub-rules which apply to it. The sub-rules fall into two broad types, "housekeeping" rules, and grammar rules.

4.1. Housekeeping Rules

Housekeeping rules enforce the basic expectations of how the system is to work and as such remain constant over all grammars. There are three such rules.

Binding consistency. If a binding unit B(i,j) has value k, then NT(i,j) should denote the same constituent as NT(i-1,k). More formally, the system tries to preserve the following constraint.

$$\{NT(i,j) = NT(i\text{-}1,k)\} + \{B(i,j) = k\}$$

This rule is applied in all ways. That is, NT(i,j) tries to make itself equal to NT(i-1,k), and vice versa, while B(i,j) tries to pick a value which will make its left and right NT's the same.

Tree-consistency. If B(i,j) = k then B(i,j+1) wants to be k+1, and vice versa.

$$\{B(i,j) = k\} + \{B(i,j+1) = k+1\}$$

Again this is used both ways, with one exception. When a higher unit, B(i,j+1), uses it to determine the value of a lower unit, B(i,j), if B(i,j+1) = k+1, then B(i,j) = k or b.

Blank-edge. If there is no constituent at a position NT(i,j) then NT(i,j) = e, for empty. In such cases no value of B(i,j) would have any real meaning. It turns out that the machine works better if such B(i,j)'s are assigned to be boundaries, so this rule encourages

$$\{NT(i,j) = e\} + \{B(i,j) = b\}$$

4.2. Grammar Rules

The program does not learn a grammar from examples, but must be given the grammar, which is then compiled into four rules, each of which form one aspect of the function computed it each NT and B units (though some only apply to one or the other).

Up-Down. Given a rule like S → NP VP, the presence of an NP at position i,j should encourage the presence of an S at i, j+1, and conversely. More generally assume a rule of the form A → ... B ...then the following combination is encouraged:

$$\{NT(i,j) = A\} + \{NT(i,j\text{-}1) = B\}$$

Left-hand Side Start Rule. Given the rule S → NP VP the presence of a starting NP should encourage a starting S.

$$\{NT(i,j) = A\} + \{NT(i,j\text{-}1) = B\} +$$
$$\text{forall}(k)[\{B(i\text{-}1,k) \neq j\} + \{B(i\text{-}1,k\text{-}1) \neq j\text{-}1\}]$$

This rule is only applied to the B units, since it is a more restricted version of the up-down rule when applied to NT's.

Left-hand side finish. Given the rule S → NP VP the presence of a VP which is ending should encourage the S above it to end.

$$\{NT(i,j) = A\} + \{NT(i,j\text{-}1) = B\} + \{B(i,j) = b\} + \{B(i,j\text{-}1) = b\}$$

Right-hand side start finish. The rule S →NP VP encourages a finishing NP with an S above it to be followed by a starting VP with an S above it. Assuming the rule A → ...B_1 B_2 ...

$$\{NT(i,j) = A\} + \{NT(i,j\text{-}1) = B_1\} + \{B(i,j\text{-}1) = b\} + \{NT(i\text{-}1,j\text{-}1) = B_2\}$$

5. Limitations and Psychological Relevance

As already noted, the parser escapes the fixed sentence length limitation because it shifts in input into the registers, constructs a parse tree for the section in question, and eventually shifts the input out on the left.

However, parse trees are bounded not only in width, but in height. Thus long sentences will tend to have high trees, and so it is possible that some of the tree will shift up "over the top" and be lost. If we are willing to agree that the system's parse tree is the collection of the partial trees created before they shifted either left and out, or up and out, then it is nevertheless possible for the program to construct arbitrarily wide and high trees. Figure 9 shows the system parsing a modest right embedded sentence which due to the very small parser width and height still overflows the buffer in both directions. (The height and width are parameters one sets.)

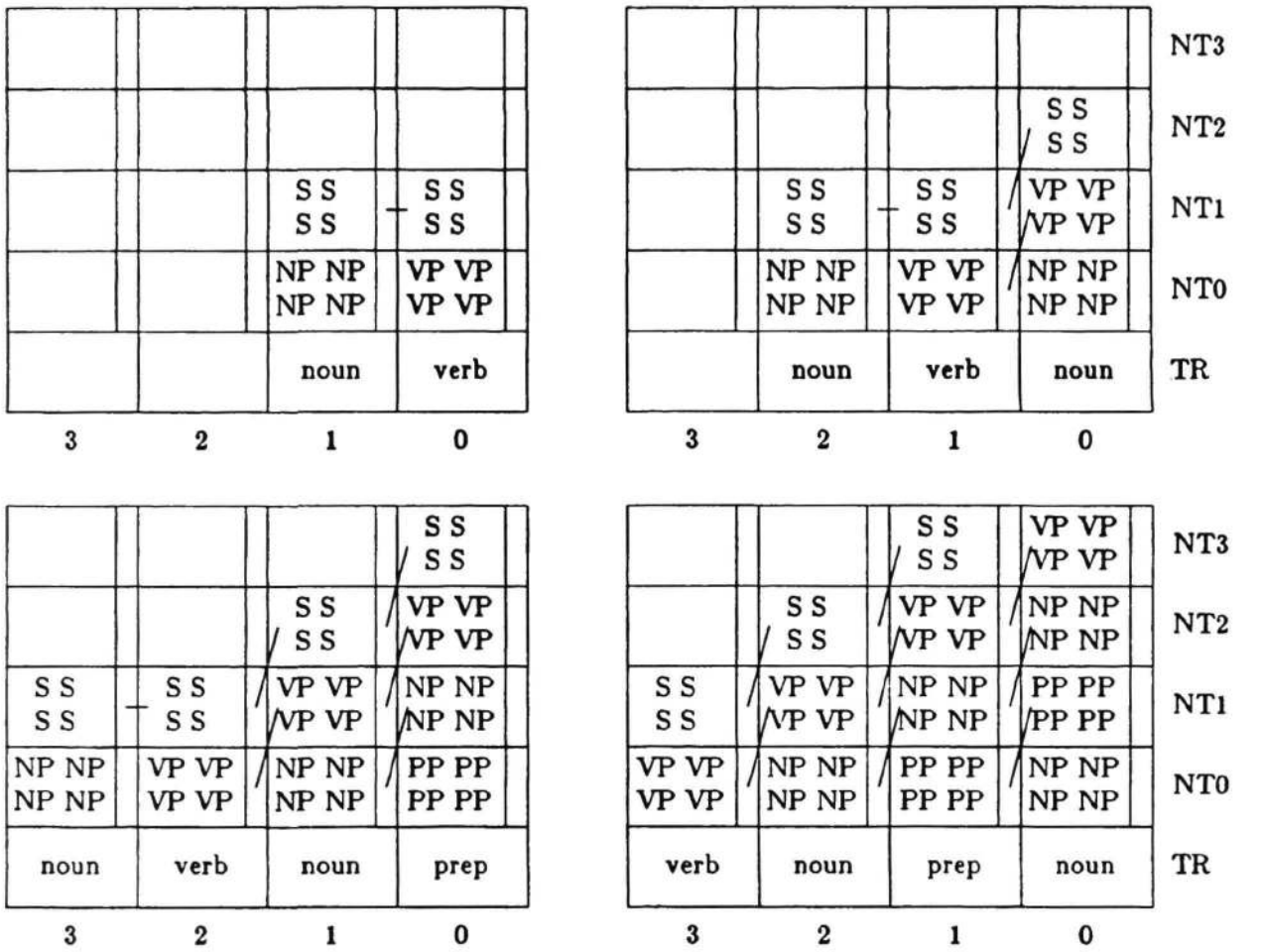

Figure 9. Overflow on 'noun verb noun prep noun'

However, the shifting has effect on what can be parsed correctly. In particular, the depth limitation means that right embedding is fine, but center embedding will be difficult. Although the front of the tree is continually being shifted left and out, and the top of the tree is going over the top, if the sentence is right embedded then the portion of the tree to which the input should attach is always present. For center embedding this will not be true, and thus the parser cannot parse such examples correctly.

6. How the Parser Really Works

We have described how the parser works in terms of the contributions of rules which we expressed as constraints to be satisfied. We have not specified exactly how the constraint satisfaction works. The actual rules are formalized in terms of array multiplication. We will consider only one of the simplest rules here, the up-down rule.

First we need to modify our notation slightly. So far we have denoted unit values with equations like NT(1,2) = S. However, as we have already noted, NT(1,2) may be, say .7 = S, .2 = NP, and .1 = PP, or some such. Therefore, it would make more sense to say NT(1,2) = (.7,.2,.1,0) where we use S = 0, NP = 1, PP = 2, and VP = 3 to indicated vector positions. Alternatively we could use a 3-D array and

say NT(1,2,1) = .2. (Arrays start with position 0.) In fact, it proves to be most convenient to represent the situation as $NT_1(2,0) = .7$ $NT_1(2,1) = .2$ etc. Here the column number becomes a subscript picking out different two dimensional arrays. Next, we can represent the grammatical information needed by the up-down rule as an array itself.

$$UD(i,j) = \begin{cases} 1 \text{ iff the ith and jth nonterminals are} \\ \quad \text{found in a rule of the form } i \rightarrow \ldots j \ldots \\ 0 \text{ otherwise} \end{cases}$$

Now consider how we apply the UD rule downward. We look at each position, and consider to what degree the entity at position j suggest what should be at j-1. We can consider the NT_i array then to be this: $NT_i(k,LHS)$. We then perform the array multiplication

$$NT_i(K,LHS) \times UD(LHS,RHS) = NT_i(K,RHS)$$

The resulting array has at position K the values that the LHS at K induces on the RHS. To get this into the format we want, we then shift down K by 1, so that the values of the RHS for K-1 appear in the K-1th position. Thus the up down rule looks like this:

$$NT_i(K\text{-}1,RHS) = \text{shift-K-down-1}(NT_i(K,LHS) \times UD(LHS,RHS))$$

The other rules are similar, but often more complicated.

7. Problems and Future Research

There are many ways in which the parser could use improvement. It currently predicts that center embedding is more difficult than right or left embedding, but does not explain why some forms of center embedding (of S's) are worse than others (of PP's). If we used a grammar which represented NP's as deep trees, rather than flat ones, this issue would be even more critical.

We have only tried the parser on the simple context-free grammar given earlier, plus a few minor extensions. We have no knowledge of how it works on more complicated grammars. In particular we would like to handle extended phrase-structure grammars. For this we would expand our NT units to be many units, each for a particular feature in the extended phrase structure approach. It looks like a natural, but whether it will work is an open question.

The parser does not learn its grammars from examples. Given how far our architecture is from traditional distributed connectionism it seems unlikely that any of the standard learning algorithms will apply. Perhaps those algorithms be extended or alternatively our architecture could be made more conventional.

8. Conclusion

We have presented a distributed connectionist architecture for parsing context free grammars. It improves earlier attempts in that it is not limited to parse trees of fixed width and height (i.e. fixed length sentences). The memory limitations inherent in connectionist architectures comes out in an inability to parse center-embedded sentences, although right-embeddeding works out fine.

9. References

1. Fanty, M., "Context-free parsing in connectionist networks," TR 174, Univeristy of Rochester Computer Science Department (1985).
2. Selman, Bart, "Rule-based processing in a connectionist system for natural language understanding," Technical Report CSRI-168, Computer Systems Research Institute, University of Toronto (1985).
3. Waltz, David L. and Pollack, Jordan B., "Massively parallel parsing: a strongly interactive model of natural language interpretation," *Cognitive Science* 9 pp. 51-74 (1985).