META-PARSING IN NEURAL NETWORKS

Anton Nijholt
Faculty of Computing Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

ABSTRACT

Constructing a connectionist network for context-free language parsing can be compared with parsing sentences in such a way that the actual (pre-) terminals or words become irrelevant. Only the structure of possible sentences is important. We introduce the notion of meta-parsing to denote this way of constructing a network and apply the idea to connectionist CYK and connectionist Earley parsing.

1 Introduction

Connectionist networks are strongly interconnected groups of very simple processing units. Such networks are studied in natural language processing since their inherent parallelism and distributed decision making allows an integration of syntactic, semantic and pragmatic processing for language analysis. See, e.g., [Waltz88] and [Cottrell89]. By isolating the syntactic component – without abandoning the connectionist paradigm – it becomes possible to study context-free parsing in environments where we can make different assumptions about types of networks, learning rules and representations of concepts. Only few authors have considered context-free language parsing in connectionist or neural networks. It is possible to distinguish a dynamic programming approach based on the CYK algorithm [Fanty85], a Boltzmann machine approach [Selman87] and an interactive relaxation approach [Howells88]. We follow Fanty's connectionist approach to CYK parsing. In this paper the emphasis is on the building of the network rather than on the parsing. Moreover, we discuss properties such a network will have. The method is generalized to connectionist Earley parsing and also in this case it is the construction of the network which gives rise to interesting observations about patterns of node configurations in the network.

2 Fanty's Connectionist CYK Parser

For convenience we recall the standard CYK algorithm. As usual, assume that the grammar is in Chomsky Normal Form (CNF). For any string $x = a_1 a_2 \cdots a_n$ to be parsed an upper-triangular $(n+1)\times(n+1)$ recognition table T is constructed. Each table entry $t_{i,j}$ with $i < j$ will contain a subset of N (the set of nonterminal symbols) such that $A \in t_{i,j}$ if and only if $A \Rightarrow^* a_{i+1} \cdots a_j$. String x belongs to the language if and only if the start symbol S is in $t_{0,n}$ when the construction of the table is completed. Initially, entries $t_{i,j}$ with $i < j$ are empty. The input string is available on the matrix diagonal (cf. Fig. 1).

(1) Compute $t_{i,i+1}$, as i ranges from 0 to $n-1$, by placing A in $t_{i,i+1}$ exactly when there is a production $A \rightarrow a_{i+1}$ in P, the set of productions.

(2) In order to compute $t_{i,j}$, $j-i>1$, assume that all entries $t_{k,l}$ with $l \leq j$, $k \geq i$ and $kl \neq ij$ have been computed. Add A to $t_{i,j}$ if, for any k such that $i<k<j$, $B \in t_{i,k}$, $C \in t_{k,j}$ and $A \rightarrow BC$ is a production rule and A is not already present in $t_{i,j}$.

a_1	0,1	0,2	0,3	0,4	0,5
	a_2	1,2	1,3	1,4	1,5
		a_3	2,3	2,4	2,5
			a_4	3,4	3,5
				a_5	4,5
					$

Fig. 1 The upper-triangular CYK-table.

We now discuss the connectionist version of this algorithm. The table's diagonal will be used for representing the input symbols. For each nonterminal symbol each entry in the table which is not on the diagonal will represent a configuration of nodes. These nodes allow top-down and bottom-up passing of activity. We first explain the bottom-up pass. Consider a particular entry, say $t_{i,j}$ with $j-i \geq 2$, of the upper-triangular matrix. In the traditional algorithm, a nonterminal symbol X is added to the set of nonterminal symbols associated with the entry if there are symbols $Y \in t_{i,k}$ and $Z \in t_{k,j}$ such that $X \rightarrow YZ$ is in P. In the connectionist adaptation we already have a node for each nonterminal symbol in entry $t_{i,j}$. Rather than adding a symbol, here node X at position $t_{i,j}$ is made active if node Y at position $t_{i,k}$ and node Z at position $t_{k,j}$ are active. In general there will be more ways to have a realization of the production $X \rightarrow YZ$ at position $t_{i,j}$. For example, a node for X at entry $t_{1,5}$ can be made active for a production $X \rightarrow YZ$ if there is an active node for Y at $t_{1,2}$ and for Z at $t_{2,5}$, or for Y at $t_{1,3}$ and for Z at $t_{3,5}$, or for Y at $t_{1,4}$ and for Z at $t_{4,5}$. This separation is realized with the help of match nodes in the configuration of each entry of the table. The use of match nodes is illustrated in Fig. 2 for a node for X at position t_{15} of a CYK-table. The three match nodes, one for each possible realization of $X \rightarrow YZ$, for this node at this particular position are shown. For match nodes to become active all of their inputs must be on. The node for X becomes active when at least one of its inputs is on.

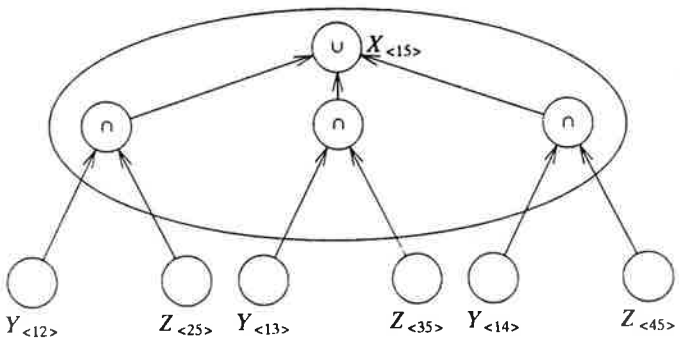

Fig. 2 Bottom-up passing of activity.

In the figure only match nodes for separate realizations of the same production are included. Match nodes should also be included at this position for all possible realizations of the other

productions with lefthand side X. In this way all the inputs that can make the node for X at this particular position active can be received in a proper way.

In our explanation the assumption $j-i \geq 2$ for entry $t_{i,j}$ was made. We assume that there is a node for each terminal symbol in each position at the diagonal of the matrix. Since the grammar is in CNF we have realizations of productions of the form $X \rightarrow a$ in the entries $t_{i,j}$ with $j-i=1$. Also in these entries match nodes are needed since different terminal symbols can have the same nonterminal as lefthand side. We assume that there is a node for each terminal symbol in each position at the diagonal of the matrix. Recognition starts by activating the nodes which correspond with the input symbols. Then activation passes bottom-up through the network, first with realizations of productions of the form $X \rightarrow a$, next with realizations of productions of the form $X \rightarrow YZ$. The input is accepted as soon as the node for the start symbol in the topmost entry of the column of the last input symbol becomes active.

A straightforward construction of the network for a particular grammar can be performed as follows. We have to choose an m, the maximum length of the strings that can be processed by the network. For each entry $t_{i,j}$ with $i \leq j$ of an upper-triangular $(m+1)\times(m+1)$ table T we have to introduce the nodes and the connections which allow the passing of activity we discussed above. This can be done diagonal by diagonal:

(0) For each $a \in \Sigma \cup \{\$\}$ introduce a node for a in entry $t_{i,i}$, $0 \leq i \leq m$.

(1a) For each production rule $A \rightarrow a$ and for each entry $t_{i,j}$ of T with $j-i=1$ introduce a match node for A in $t_{i,j}$ and connect it with the node for a in $t_{i,i}$.

(1b) For each collection of match nodes in an entry $t_{i,j}$ with $j-i=1$ that correspond with the same nonterminal introduce a node for that nonterminal in $t_{i,j}$ and connect it with its match nodes.

(2a) For each k, $i<k<j$, such that there is a node for B in $t_{i,k}$, a node for C in $t_{k,j}$ and a production rule $A \rightarrow BC$ in P introduce a match node for A in $t_{i,j}$ and connect it with the nodes for B and C in $t_{i,k}$ and $t_{k,j}$, respectively.

(2b) For each collection of match nodes in an entry $t_{i,j}$ that correspond with the same nonterminal introduce a node for that nonterminal in $t_{i,j}$ and connect it with its match nodes.

A more global look on the network learns us that in each entry of a particular diagonal the same collection of nodes and match nodes will be introduced. Since the set of nonterminals is finite we obtain a 'regular' pattern of diagonals. A node for a nonterminal A is introduced in an entry $t_{i,j}$ if and only if A can generate a string of length $j-i$. An appropriate name for building the network for a particular grammar should therefore be **meta-CYK-parsing**. Although we obtain in this way diagonals with entries that have nodes for the same set of nonterminals they can differ in the number of match nodes. As an example of meta-parsing consider the following context-free grammar in CNF. The symbol | separates the different alternatives of a nonterminal symbol.

$$\begin{array}{ll} S \rightarrow AB \mid AC & C \rightarrow AB \\ A \rightarrow a & D \rightarrow AB \\ B \rightarrow b \mid DE & E \rightarrow b \end{array}$$

In Fig. 3 we show the nonterminals in the entries of the upper-triangular table for which nodes and match nodes will be introduced. On the second diagonal $(j-i=1)$ we have nodes for nonterminals with the property that they can generate strings of length 1, at the third diagonal $(j-i=2)$

those for strings of length 2, etc.

a b	A B E	S C D	S B	S C D	S B	S C D
	a b	A B E	S C D	S B	S C D	S B
		a b	A B E	S C D	S B	S C D
			a b	A B E	S C D	S B
				a b	A B E	S C D
					a b	A B E
						a b

Fig. 3 Meta-CYK-Parsing.

Until now we have discussed a network which accepts (or rejects) a string. In order to obtain a representation of the parse tree(s) a second, top-down, pass of activity is necessary. To perform this top-down pass we assume that each node mentioned so far consists of a bottom-up and a top-down unit. The bottom-up units are used as explained above. In Fig. 4 both bottom-up and top-down passing of activity is illustrated in a configuration of nodes for an entry $t_{i,j}$ with $j-i \geq 2$. Each node is represented as consisting of a leftmost or bottom-up and a rightmost or top-down unit. A top-down unit becomes active when it receives input from its bottom-up counterpart and at least one external source. In order to activate the top-down unit of the node for the start symbol in the upper right corner of the table we assume that it receives input from its bottom-up counterpart and from the node at position $t_{n,n}$, where n is the length of the input, which is used to represent endmarker $\$$ of the input and which is made active when parsing starts. Hence, when the input is recognized this unit becomes active and it passes activity top-down. All top-down units which receive this activation and activation from their bottom-up counterparts become active. Hence, activity is passed down to the terminal nodes and the active top-down nodes of the network represent the parse tree(s).

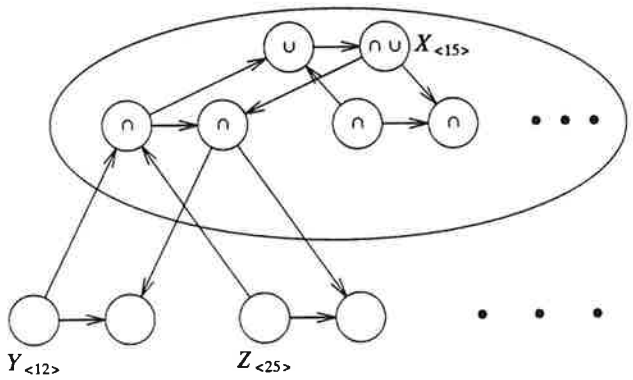

Fig. 4 Top-down and bottom-up passing of activity in CYK parsing.

If the grammar is not in CNF, and if we forget about details on the distinction between terminal and nonterminal symbols, then, rather than looking for only two symbols in the table which may constitute a righthand side of a production, it now becomes necessary to consider combinations of two, three, or more symbols. In general, if the longest righthand side has length p, then $O(n^{p-1})$ matches are necessary for each of the $O(n^2)$ elements of the CYK recognition table. The $O(n^{p-1})$ matches translate into $O(n^{p-1})$ possible realizations of productions for each of the $O(n^2)$ entries of the CYK-table. Therefore the number of connections for each node will be proportional with n^{p-1} and the total number of nodes in the network will be proportional with n^{p+1}. The parse in the connectionist network completes in $O(n)$ time.

3 A Connectionist Earley Parser

We recall the (tabular version of the) Earley context-free parsing algorithm. For any string $x = a_1 a_2 \cdots a_n$ to be parsed an upper-triangular $(n+1)\times(n+1)$ recognition table T is constructed. Each entry $t_{i,j}$ will contain a set of items of the form $A \rightarrow \alpha\cdot\beta$, where $A \rightarrow \alpha\beta$ is a production rule and the dot $\cdot$ is a symbol not in $N \cup \Sigma$. The computation of the table entries goes column by column. The following two functions will be useful. Function PREDICT:$N \rightarrow 2^D$, where $D = \{A \rightarrow \alpha\cdot\beta \mid A \rightarrow \alpha\beta \in P\}$, is defined as

$$\text{PREDICT}(A) = \{B \rightarrow \alpha\cdot\beta \mid B \rightarrow \alpha\beta \in P,\ \alpha \Rightarrow^* \varepsilon \text{ and } \exists\ \gamma \in V^* \text{ with } A \Rightarrow^* B\gamma\}.$$

Function PRED:$2^N \rightarrow 2^D$ is defined as $\text{PRED}(X) = \bigcup_{A \in X} \text{PREDICT}(A)$.

Initially, $t_{0,0} = \text{PRED}(\{S\})$ and all other entries are empty. Suppose we want to compute the elements of column j, $j > 0$. In order to compute $t_{i,j}$ with $i \neq j$ assume that all elements of the columns of the upper-triangular table to the left of column j have been computed and in column j the elements $t_{k,j}$ for $i < k < j$ have been computed.

(1) Add $B \rightarrow \alpha a\beta\cdot\gamma$ to $t_{i,j}$ if $B \rightarrow \alpha\cdot a\beta\gamma \in t_{i,j-1}$, $a = a_j$ and $\beta \Rightarrow^* \varepsilon$.

(2) Add $B \rightarrow \alpha A\beta\cdot\gamma$ to $t_{i,j}$, if, for any k such that $i < k < j$, $B \rightarrow \alpha\cdot A\beta\gamma \in t_{i,k}$, $A \rightarrow \omega\cdot \in t_{k,j}$ and $\beta \Rightarrow^* \varepsilon$.

(3) Add $B \rightarrow \alpha A\beta\cdot\gamma$ to $t_{i,j}$ if $B \rightarrow \alpha\cdot A\beta\gamma \in t_{i,i}$, $\beta \Rightarrow^* \varepsilon$ and there exists $C \in N$ such that $A \Rightarrow^* C$ and $C \rightarrow \omega\cdot \in t_{i,j}$.

After all elements $t_{i,j}$ with $0 \le i \le j-1$ of column j have been computed then it is possible to compute $t_{j,j}$.

(4) Let $X_j = \{A \in N \mid B \rightarrow \alpha\cdot A\beta \in t_{i,j}, 0 \le i \le j-1\}$. Then $t_{j,j} = \text{PRED}(X_j)$.

It is not difficult to see that $A \rightarrow \alpha\cdot\beta \in t_{i,j}$ if and only if there exists $\gamma \in V^*$ such that $S \Rightarrow^* a_1 \cdots a_i A\gamma$ and $\alpha \Rightarrow^* a_{i+1} \cdots a_j$. Hence, in $t_{0,n}$ we can read whether the sentence was correct. The algorithm can be extended in order to produce parse trees.

Various parallel implementations of Earley's algorithm have been suggested in the literature. The main problem is the parallel computation of the diagonal elements $t_{i,i}$, for $0 \le i \le n$. The solution is simple. Initially all elements $t_{i,i}$, $0 \le i \le n$, are set equal to PRED(N), where N is the set of nonterminal symbols. The other entries are defined according to the steps (1), (2) and (3). As a consequence, we now have $A \rightarrow \alpha\cdot\beta \in t_{i,j}$ if and only if $\alpha \Rightarrow^* a_{i+1} \cdots a_j$. Computation can now be done diagonal by diagonal:

(1) Set $t_{i,i}$ equal to PRED(N), $0 \le i \le n$.

(2) Set $d = 0$. Assuming $t_{i,i+d}$ has been formed for $0 \le i \le n-d$, increase d with 1 and compute $t_{i,j}$ for $0 \le i \le n-d$ and $j = i+d$ according to:

(2.1) Add $B \rightarrow \alpha a \beta \cdot \gamma$ to $t_{i,j}$ if $B \rightarrow \alpha \cdot a \beta \gamma \in t_{i,j-1}$, $a = a_j$ and $\beta \Rightarrow^* \varepsilon$.

(2.2) Add $B \rightarrow \alpha A \beta \cdot \gamma$ to $t_{i,j}$ if, for any k such that $i < k < j$, $B \rightarrow \alpha \cdot A \beta \gamma \in t_{i,k}$, $A \rightarrow \omega \cdot \in t_{k,j}$ and $\beta \Rightarrow^* \varepsilon$.

(2.3) Add $B \rightarrow \alpha A \beta \cdot \gamma$ to $t_{i,j}$ if $B \rightarrow \alpha \cdot A \beta \gamma \in t_{i,i}$, $\beta \Rightarrow^* \varepsilon$ and there exists $C \in N$ such that $A \Rightarrow^* C$ and $C \rightarrow \omega \cdot \in t_{i,j}$.

We are now sufficiently prepared to introduce a connectionist Earley parser. The network that is built defines again a limit on the lengths of the strings it can process. Hence, we have to choose an m and for strings longer than m the network should be extended. To obtain connections according to the rows and columns of the table, input string $a_1 \cdots a_n$ will be offered to the network on the positions $t_{1,1}$ to $t_{n,n}$. The nodes and connections can be introduced diagonal by diagonal:

(0) For each entry $t_{i,i}$, $0 \le i \le m$, introduce nodes in $t_{i,i}$ for each $a \in \Sigma \cup \{\$\}$ and for each item in PRED(N).

(1) Introduce a match node for $B \rightarrow \alpha a \beta \cdot \gamma$ in $t_{i,j}$ if there is a node for $B \rightarrow \alpha \cdot a \beta \gamma$ in $t_{i,j-1}$ and $\beta \Rightarrow^* \varepsilon$. Connect it with this node and with the node for a in $t_{j,j}$.

(2a) For each k, $i < k < j$, such that there is a node for $B \rightarrow \alpha \cdot A \beta \gamma$ in $t_{i,k}$ and a node for $A \rightarrow \omega \cdot$ in $t_{k,j}$ and $\beta \Rightarrow^* \varepsilon$, introduce a match node for $B \rightarrow \alpha A \beta \cdot \gamma$ in $t_{i,j}$ and connect it with these two nodes.

(2b) For each collection of match nodes in an entry $t_{i,j}$ that correspond with the same item introduce a node for that item in $t_{i,j}$ and connect it with its match nodes.

(3a) Introduce a match node for $B \rightarrow \alpha A \beta \cdot \gamma$ in $t_{i,j}$ if there is a node for $B \rightarrow \alpha \cdot A \beta \gamma$ in $t_{i,i}$, $\beta \Rightarrow^* \varepsilon$ and there exists $C \in N$ such that $A \Rightarrow^* C$ and $t_{i,j}$ has a node for $C \rightarrow \omega \cdot$. Connect the match node with these two nodes.

(3b) Finally, do once more step (2b), but now to obtain nodes for the match nodes introduced in step (3a).

We now use the name **meta-Earley parsing** for building this network. In each entry of a particular diagonal the same collection of nodes and match nodes are introduced and we obtain a periodically returning pattern of diagonals. A node for an item $A \rightarrow \alpha \cdot \beta$ is introduced in entry $t_{i,j}$ if and only if α can generate a string of length $j-i$. That is, on the first diagonal ($j-i=0$) we have nodes for items of the form $A \rightarrow \alpha \cdot \beta$ where α has the property that it can generate the empty string, on the second diagonal ($j-i=1$) α of item $A \rightarrow \alpha \cdot \beta$ can generate strings of length 1, etc. For each item we can determine in advance to what entries it will belong. Although we obtain in this way diagonals with entries that have nodes for the same set of items, they can differ in the number of match nodes. As an example consider the following grammar

$$S \rightarrow A \mid B \qquad A \rightarrow aAa \mid \varepsilon \qquad B \rightarrow bbBb \mid \varepsilon$$

In Fig. 5 we show the items in the entries of the upper-triangular table for which nodes and match nodes will be introduced during meta-parsing. The sets of nodes that are displayed in the entries from $t_{0,3}$ to $t_{0,8}$ constitute the pattern that repeats itself in the table. For example grammar G

with rules

$$S \rightarrow NP\ VP \mid S\ PP \qquad PP \rightarrow *prep\ NP$$
$$NP \rightarrow *n \mid *det\ *n \mid NP\ PP \qquad VP \rightarrow *v\ NP$$

a simple analysis shows that all table entries $t_{i,j}$ with $j-i>3$ will contain a node (and match nodes) for each of the following items:

$$S \rightarrow NP \cdot VP \mid NP\ VP \cdot \mid S \cdot PP \mid S\ PP \cdot$$
$$NP \rightarrow NP \cdot PP \mid NP\ PP \cdot$$
$$PP \rightarrow *prep\ NP \cdot$$
$$VP \rightarrow *v\ NP \cdot$$

Hence, apart from the number of match nodes (a number that increases when sentences become longer) the structure is obtained from only one subset of the set of all items.

PRED(N) ΣU {$}	$A \rightarrow a \cdot Aa$ $B \rightarrow b \cdot bBb$ $A \rightarrow aA \cdot a$	$A \rightarrow aAa \cdot$ $B \rightarrow bb \cdot Bb$ $B \rightarrow bbB \cdot b$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbBb \cdot$ $S \rightarrow B \cdot$	$A \rightarrow aAa \cdot$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbB \cdot b$	$A \rightarrow aAa \cdot$ $B \rightarrow bbBb \cdot$ $S \rightarrow A \cdot$ $S \rightarrow B \cdot$	$A \rightarrow aA \cdot a$	$A \rightarrow aAa \cdot$ $B \rightarrow bbB \cdot b$ $S \rightarrow A \cdot$
	PRED(N) ΣU {$}	$A \rightarrow a \cdot Aa$ $B \rightarrow b \cdot bBb$ $A \rightarrow aA \cdot a$	$A \rightarrow aAa \cdot$ $B \rightarrow bb \cdot Bb$ $B \rightarrow bbB \cdot b$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbBb \cdot$ $S \rightarrow B \cdot$	$A \rightarrow aAa \cdot$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbB \cdot b$	$A \rightarrow aAa \cdot$ $B \rightarrow bbBb \cdot$ $S \rightarrow A \cdot$ $S \rightarrow B \cdot$	$A \rightarrow aA \cdot a$
		PRED(N) ΣU {$}	$A \rightarrow a \cdot Aa$ $B \rightarrow b \cdot bBb$ $A \rightarrow aA \cdot a$	$A \rightarrow aAa \cdot$ $B \rightarrow bb \cdot Bb$ $B \rightarrow bbB \cdot b$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbBb \cdot$ $S \rightarrow B \cdot$	$A \rightarrow aAa \cdot$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbB \cdot b$	$A \rightarrow aAa \cdot$ $B \rightarrow bbB \cdot b$ $S \rightarrow A \cdot$ $S \rightarrow B \cdot$
			PRED(N) ΣU {$}	$A \rightarrow a \cdot Aa$ $B \rightarrow b \cdot bBb$ $A \rightarrow aA \cdot a$	$A \rightarrow aAa \cdot$ $B \rightarrow bb \cdot Bb$ $B \rightarrow bbB \cdot b$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbBb \cdot$ $S \rightarrow B \cdot$	$A \rightarrow aAa \cdot$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbB \cdot b$
				PRED(N) ΣU {$}	$A \rightarrow a \cdot Aa$ $B \rightarrow b \cdot bBb$ $A \rightarrow aA \cdot a$	$A \rightarrow aAa \cdot$ $B \rightarrow bb \cdot Bb$ $B \rightarrow bbB \cdot b$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbBb \cdot$ $S \rightarrow B \cdot$	$A \rightarrow aAa \cdot$ $S \rightarrow A \cdot$
					PRED(N) ΣU {$}	$A \rightarrow a \cdot Aa$ $B \rightarrow b \cdot bBb$ $A \rightarrow aA \cdot a$	$A \rightarrow aAa \cdot$ $B \rightarrow bb \cdot Bb$ $B \rightarrow bbB \cdot b$ $S \rightarrow A \cdot$	$A \rightarrow aA \cdot a$ $B \rightarrow bbBb \cdot$ $S \rightarrow B \cdot$
						PRED(N) ΣU {$}	$A \rightarrow a \cdot Aa$ $B \rightarrow b \cdot bBb$ $A \rightarrow aA \cdot a$	$A \rightarrow aAa \cdot$ $B \rightarrow bb \cdot Bb$ $B \rightarrow bbB \cdot b$ $S \rightarrow A \cdot$
							PRED(N) ΣU {$}	$A \rightarrow a \cdot Aa$ $B \rightarrow b \cdot bBb$ $A \rightarrow aA \cdot a$
								PRED(N) ΣU {$}

Fig. 5 Meta-Earley-parsing.

The network that is built is a network of 'possible realizations'. An actual sentence will cause the selection of those items from the network that are relevant for the structure of that sentence. In order to recognize a string $a_1 \cdots a_n$ we activate in $t_{i,i}$, $0 \le i \le n$, the elements of PRED(N)

and the node for a_i, $1 \leq i \leq n$. Activation passes bottom-up with realizations of items of the form $A \rightarrow \alpha \cdot \beta$. For a match node to become active all its inputs must be on. A node with match nodes becomes active when at least one of its inputs is on. The input is accepted as soon as a node for an item of the form $S \rightarrow \omega \cdot$ in the topmost entry of the column of the last input symbol becomes active. In order to distinguish parse trees top-down parsing of activity is required. This can be done by introducing bottom-up and top-down units of a node, in a similar way as has been discussed for connectionist CYK parsing. In Fig. 6 both bottom-up and top-down passing of activity is illustrated in a configuration of nodes for an entry $t_{i,j}$ with $j - i > 1$.

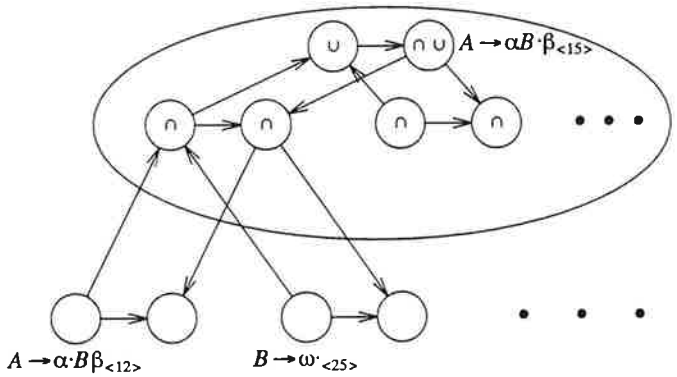

Fig. 6 Top-down and bottom-up passing of activity in Earley parsing.

A top-down unit becomes active when input is received from the bottom-up counterpart and from an external source. In order to initiate top-down passing of activity we have a node for symbol \$ in each entry $t_{i,i}$ which is connected to each top-down unit associated with an item of the form $S \rightarrow \omega \cdot$ in $t_{0,i}$. For input string $a_1 \cdots a_n$ both the node for a_n and the node for \$ will be made active in entry $t_{n,n}$ when parsing starts. When the input is recognized a top-down unit associated with an item of the form $S \rightarrow \omega \cdot$ becomes active and it passes activity top-down. All top-down units which receive this activation and activation from their bottom-up counterparts become active. Hence, activity is passed down to the terminal nodes. From the active top-down units of the network the parse tree(s) can be reconstructed.

4 Conclusions

The main contribution of this paper is the emphasis on the fact that constructing a connectionist network along the lines suggested by Fanty can be compared with parsing sentences in such a way that the actual (pre)terminals or words of the sentence do not play a role. The properties of the regular patterns that show themselves in the structures of the networks have been discussed in [Sikkel90]. In the same report, the methods discussed here are extended to include a connectionist version of Rytter's algorithm [Gibbons88] for context-free language parsing [Sikkel90], [Sikkel91]. An obvious disadvantage of the approach discussed here is the inability to parse sentences of arbitrary length. Alternatives are discussed in [Howells88] and [Nakagawa88]. A survey of non-connectionist parallel context-free parsing methods can be found in [Nijholt99].

Acknowledgements:
I am grateful to K. Sikkel for his comments on a first draft of this paper.

References

[Cottrell89] G.W. Cottrell, *A Connectionist Approach to Word Sense Disambiguation.* Research Notes in Artificial Intelligence, Morgan Kaufmann Publishers, 1989.

[Fanty85] M.A. Fanty, "Context-free parsing in connectionist networks." Report TR 174, Computer Science Department, University of Rochester, 1985.

[Gibbons88] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms.*, Cambridge University Press, 1988.

[Howells88] T. Howells, "A connectionist parser." Proceedings of the tenth Annual *Conference of the Cognitive Science Society*, 1988, 18-25.

[Nakagawa88] H. Nakagawa and T. Mori, "A parser based on connectionist model." In: *COLING 88*, Budapest, 454-458.

[Nijholt91] A. Nijholt, "Parallel approaches to context-free language parsing." In: *Parallel Models of Natural Language Computation*, U. Hahn and G. Adriaens (eds.), Ablex Publishing Corporation, Norwood, New Jersey, to appear in 1991.

[Selman87] B. Selman and G. Hirst, "Parsing as an energy minimization problem." Chapter 11 in *Genetic Algorithms and Simulated Annealing.* Research Notes in A.I., Morgan Kaufmann Publishers, Los Altos, California, 1987.

[Sikkel90] K. Sikkel, "Connectionist parsing of context-free grammars." Memoranda Informatica 90-30, Computer Science Department, University of Twente, 1990.

[Sikkel91] K. Sikkel and A. Nijholt, "Connectionist parsing of context-free grammars." In: Proceedings *Int. Workshop on Parsing Technologies*, Cancun, Mexico, 1991, 117-126.

[Waltz88] D. Waltz and J.B. Pollack, "Massively parallel parsing: A strongly interactive model of natural language interpretation." In: D.L. Waltz and J.A. Feldman (eds.), *Connectionist Models and their Implications*, Readings from Cognitive Science, Ablex Publishing Co., Norwood, NJ, 1988, 181-204.