A Structured Neural Network Approach to Robust Parsing

Eugene Santos Jr.

Department of Computer Science
Brown University
Providence, Rhode Island 02912

Technical Report No. CS-91-14
February 1991

A Structured Neural Network Approach to Robust Parsing*

Eugene Santos Jr.
Department of Computer Science
P.O. Box 1910, Brown University
Providence, RI 02912
e-mail: esj@cs.brown.edu

Abstract

Practical natural language interfaces must exhibit a robustness in dealing with ungrammatical data. Existing approaches considered ungrammaticality as some form of "noisy" data. Extensions were then made to existing language systems to handle some particular type or class of ungrammaticality. Most of the original systems were built from traditional symbolic approaches which are poorly suited for handling noise. The resulting augmented language systems were generally inefficient and cumbersome. Connectionism offers an alternative approach to this problem with its natural ability at handling noise. However, very few successful connectionist systems have been built which can adequately model complex tasks such as parsing. In this paper, we present a robust parser built from the precepts of structured neural networks which combines the desirable properties of traditional symbolic approaches and of the connectionist approaches.

1. Introduction

In the real world of human natural language processing, we often encounter speech utterances which deviate from our normal grammatical and semantic expectations. For practical purposes, natural language interfaces must exhibit a robustness when attempting to process these deviations. Many researchers have made this observation and have attempted to construct these robust interfaces [1, 2, 3, 4, 5, 6, 7, 8, 9]. Most of these approaches have been involved with extending existing language systems to cover some specified subset of these

* This work has been supported by the National Science Foundation under grant IRI-8911122 and by the Office of Naval Research, under contract N00014-88-K-0589.

deviations (or ungrammaticalities). With a few exceptions[1], the majority of these techniques treated ungrammaticality as some form of "noisy" data. However, the resulting systems have been generally inefficient and cumbersome making them impractical as natural language interfaces.

The existing systems modified by these approaches were built with traditional symbolic techniques. Without careful consideration of the possibility of handling ungrammaticality in the original system, most were not well suited to handling noise. Thus, the difficulties experienced in the previous work were not unexpected.

Our approach to designing a robust natural language interface centers around using the concepts found in neural networks [10]. The neural network approach naturally handles noise. However, the resulting robustness found in most of these systems has usually just been a byproduct. Very little effort has been made at seriously exploiting this property. Our work focuses on carefully understanding this robustness in a "structured network." We will see that by integrating the representative power of probabilistic grammars into neural networks, a natural and powerful approach to robust parsing can be developed.

In section 2, we present a brief discussion of the relative merits of structured neural networks and its applicability to our problem. In section 3, we present the definitions and notations of probabilistic grammars which serves as the mathematical foundations of our approach. In section 4, we define the notion of parsing probabilistic grammars. In section 5, we implement a parser for probabilistic grammars using a feedforward neural network. In section 6, we consider ungrammaticality and how it can be solved with neural networks. Finally, we conclude our discussion in section 7.

2. Structured Neural Networks

"Structured" neural networks offer an approach with the best properties from both traditional symbolic approaches and "standard" neural network approaches. Structured networks can implement properties such as variable binding and direct symbol manipulation natural to symbolic systems while retaining the massive parallelism and "relaxation" properties of neural networks. A wider range of problems can be solved by using structured networks.

Most of the successful standard neural network approaches in NLP have been in solving the relatively low-level cognitive processes such as word order [11] and simple sentence "completion" [12]. These are easily understood and well-defined. Higher order processes, on the other hand, present an increased complexity and an added uncertainty. This

[1] [8] suggested that certain forms of ungrammaticalities in natural language can be modeled with regular structures. If this is the case, then these form are no longer treated as ungrammaticalities.

makes the encoding of such processes in the simple networks rather difficult. Also, with the increase in the complexity of the problem, the expected increase in the complexity of the encoding often results in overly large networks which are impractical to maintain.

Symbolic approaches in NLP often do not suffer the problems of the simple networks in dealing with these higher order processes. Successful implementations of different processes are relatively easy to understand and interpret due to the symbolic nature of these systems. However, as we mentioned above, problems such as "noise" are often difficult to deal with in these approaches.

When considering other problems such as language acquisition and ungrammaticality, both the symbolic approach and the standard neural network approach often fail to arrive at a "desirable" solution. Yet, each has to offer some property available to one but not the other.

In dealing with our problem of ungrammaticality, symbolic approaches present us with a natural framework for modeling parsing but not in its extension to robust parsing. On the other hand, the problems of noisy data seem more amenable to the "relaxation" property inherent in neural networks.

Through structured networks, a new type of system can be constructed which can combine the best of both approaches. For many problems, structured networks may provide straightforward solutions not readily obtainable from the other two approaches. Successful structured network approaches in NLP include syntactic parsing systems [12, 13, 14, 15] and reasoning systems [16, 17].

Although structured networks seem promising, the major problem with the approach is attempting to apply learning which is natural to the standard approaches. With the structured nature of these networks, existing learning algorithms may not be applicable. However, we feel that the usefulness of this approach will outweigh not having a standard learning algorithm.

3. Probabilistic Grammars

Although the ultimate solution in processing ungrammaticality lies in the proper integration of semantic and syntactic information, much insight can be gained in exploring the limits of purely syntactic parsing.

Integrating the inherent representational power of probabilistic grammars into structured networks provides a natural means for constructing a robust syntactic parsing system.

The basis of our formulation lies in extending the concept of formal languages. In this section, we present a class of languages called the weighted languages. The properties of this language will form the foundations of our system. We shall define the basic concepts and notations which will be needed in subsequent sections.

Notation. Γ^* is the collection of all strings over Γ, and $|x|$ is the length of the string x, and Λ is the null string. Moreover, $\Gamma^+ = \Gamma^* - \{\Lambda\}$.

Definition. Given a finite alphabet Γ, a *weighted language* **L** (*over* Γ) is characterized by a function $\chi_{\mathbf{L}}$ from Γ^* into [0,1], the closed unit interval.

Remarks. $\chi_{\mathbf{L}}(x)$ is the characteristic function of **L**, and may be interpreted as the probability that x is in **L** or as the probability that x is "grammatical". Clearly, if $\chi_{\mathbf{L}}(x) = 0$ or 1 for all $x \in \Gamma^*$, then **L** is an ordinary formal language.

Definition. A *weighted phrase structure grammar* G (or simply *weighted grammar*) is a 4-tuple (Γ, N, P, S) where Γ is a finite set of terminals, N is a finite set of nonterminals disjoint from Γ, $S \in N$ is the start symbol, and P is a function from $V^*NV^* \times V^*$ into [0,1], where $V = \Gamma \cup N$. Moreover,

1) G is *finitary* iff $\{ (x, y) \mid P(x, y) > 0 \}$ is finite.
2) G is *uniform* iff $P(x, y) > 0$ implies $x \in N^+$ and $y \in N^+ \cup \Gamma^*$.
3) G is a *weighted context-sensitive grammar* iff G is finitary and $P(x, y) > 0$ implies $x = w_1Aw_2$ and $y = w_1zw_2$ for some $A \in N$, $w_1, w_2 \in V^*$, and $z \in V^+$.
4) G is a *weighted context-free grammar* iff G is finitary and $P(x, y) > 0$ implies $x \in N$.

If, in addition, $y \neq \Lambda$, then G is a *Λ-free weighted context-free grammar.*

Remark. In the above definitions, iff stands for if and only if.

Weighted, stochastic[2], and, probabilistic grammars and languages have been examined extensively in the literature [18, 19, 20].

If $G = (\Gamma, N, P, S)$ is a finitary weighted grammar, then P is completely characterized by a finite set of weighted productions of the form

$$x \xrightarrow{p} y$$

where $p = P(x, y) > 0$. In this case, p is the probability associated with the weighted production. If no ambiguity is likely to arise, we shall use the same symbol P to denote the set of weighted productions associated with P.

Notation. Lub stands for least upper bound while glb stands for greatest lower bound. In addition, we assume that lub $\emptyset$ = glb $\emptyset$ = 0 where $\emptyset$ is the null set.

Notation. Let $x, y, u, v \in (\Gamma \cup N)^*$. $(x, u) \rightarrow (y, v)$ iff $x = x_1ux_2$ and $y = y_1vy_2$ for some $x_1, x_2, y_1, y_2 \in (\Gamma \cup N)^*$.

Definition. Let $G = (\Gamma, N, P, S)$ be a weighted grammar, and $x, y \in (\Gamma \cup N)^*$.

1) $P_1(x, y) = \text{lub}\{ P(u, v) \mid (x, u) \rightarrow (y, v) \}$.
2) $P_k(x, y) = \text{lub}\{ P_{k-1}(x, z) \otimes P(z, y) \mid z \in (\Gamma \cup N)^* \}$ for $k > 1$.
3) $P_\infty(x, y) = \text{lub}\{ P_k(x, y) \mid k \geq 1 \}$.

In the above definitions, $\otimes$ is a binary operator over [0,1] which is commutative, asso-

---

[2] **L** is a stochastic language if $\sum_{x \in \Gamma^*} \chi_{\mathbf{L}}(x) = 1$.

ciative, distributive with respect to lub and glb, and $a \otimes b \leq a$ for every $a, b \in [0,1]$.

Remark. The multiplication operator and the Min operator satisfy the conditions required of $\otimes$.

Definition. The *weighted language* $L(G)$ *generated by* the weighted grammar $G = (\Gamma, N, P, S)$ is defined by

$$\chi_{L(G)}(x) = P_\infty(S, x) \text{ for all } x \in \Gamma^*.$$

In the rest of the paper, unless otherwise specified, Γ will represent a finite set of terminals, N a finite set (disjoint from Γ) of nonterminals, and $V = \Gamma \cup N$. Moreover, unless otherwise stated, G will represent the Λ-free weighted context-free grammar (Γ, N, P, S).

4. Weighted CYK Parser

Weighted context-free grammars, like their conventional counterparts, are powerful formalisms for representing languages in a compact and precise form. However, to properly utilize this formalism, conventional grammars must have associated parsers to, in essence, decode them. In this section, we shall present a parser for weighted context-free grammars.

Notation. Let T be a labeled tree.

1) If B is a node of T, then Label(B) is the label of B.
2) Root(T) is the label of the root of T.
3) Leaves(T) is the string consisting of all the labels of the leaves of T, ordered from left to right.

Notation. Let a be a symbol. Tree(a) will denote the degenerate tree with the single node labeled a.

Notation. Let A be a symbol and $T_1, T_2, \ldots, T_n$ be a sequence labeled trees. $\vartheta(A, T_1, T_2, \ldots, T_n)$ is the labeled tree T where Root(T) = A and the subtrees of the root of T are $T_1, T_2, \ldots, T_n$ ordered from left to right according to the sequence.

Definition. An *admissible tree* T over (Γ, N) is a labeled tree which satisfies the following two conditions:

i) If B is a leaf of T, then Label(B) $\in \Gamma$.
ii) If B is an internal node of T, then Label(B) $\in N$.

The collection of all admissible trees over (Γ, N) will be denoted by $\mathcal{T}(\Gamma, N)$.

Remark. In formal languages, $\mathcal{T}(\Gamma, N)$ can represent any parse trees formed from Λ-free context-free grammars over (Γ, N).

Definition. Let $T \in \mathcal{T}(\Gamma, N)$. The G-*derivational probability of* T, denoted by $DP_G(T)$, is defined recursively as follows:

i) If T = Tree(a) for some $a \in \Gamma$, then $DP_G(T) = 1$.
ii) Let $T = \vartheta(A, T_1, T_2, \ldots, T_n)$ for some $A \in N$ and $T_1, T_2, \ldots, T_n \in \mathcal{T}(\Gamma, N)$. For $i = 1, 2, \ldots, n$, let $A_i = \text{Root}(T_i)$ and $p_i = DP_G(T_i)$. Then

$$DP_G(T) = p_0 \otimes p_1 \otimes p_2 \otimes \ldots \otimes p_n \text{ where } p_0 = P(A, A_1A_2\ldots A_n).$$

The collection of all $T \in \mathcal{T}(\Gamma, N)$ such that $DP_G(T) > 0$ will be denoted by $\mathcal{T}(G)$.

Theorem 4.1. If $T = \vartheta(A, T_1, T_2, \ldots, T_n)$, then

$$DP_G(T) \leq DP_G(T_i) \text{ for all } i = 1, 2, \ldots, n.$$

Proof. Follows from condition (ii) of the definition of G-derivational probability. ♦

Definition. Let $A \in N$ and $x \in \Gamma^*$.

1) Let $T \in \mathcal{T}(G)$. T is a G-*derivation tree for* x *from* A iff Root(T) = A and Leaves(T) = x. Moreover, T is a *maximal* G-*derivation tree for* x *from* A iff $DP_G(T) \geq DP_G(T')$ for all G-derivation tree T' for x from A.

2) The G-*derivational probability of* x *from* A, denoted by $DP_G(A, x)$, is defined by
$$DP_G(A, x) = \text{lub}\{\, DP_G(T) \mid T \in \mathcal{T}(G), \text{Root}(T) = A, \text{Leaves}(T) = x \,\}.$$

For simplicity, we shall write $DP_G(x)$ for $DP_G(S, x)$.

Theorem 4.2. Let $A \in N$ and $x \in \Gamma^*$.

1) $DP_G(A, x) > 0$ iff there exists a G-derivation tree for x from A.
2) $DP_G(A, x) = DP_G(T)$ iff T is a maximal G-derivation tree for x from A.
3) There exists a G-derivation tree for x from A iff there exists a maximal G-derivation tree for x from A

Proof. (1) follows immediately from the definition of $DP_G(A, x)$. (2) and (3) follow from the fact that the set $\{\, DP_G(T) \mid T \in \mathcal{T}(G), \text{Root}(T) = A, \text{Leaves}(T) = x \,\}$ is bounded above. ♦

Theorem 4.3. Let $G = (\Gamma, N, P, S)$ be a Λ-free weighted context-free grammar.

$$\chi_{L(G)}(x) = DP_G(x) \text{ for all } x \in \Gamma^*.$$

Proof. Follows from Theorem 4.2 and the fact that the set $\{\, P(u, v) \mid (x, u) \rightarrow (y, v) \,\}$ is finite. ♦

The above theorems provide a basis for constructing parsers for weighted context-free grammars. However, unlike ordinary formal grammars, finding a derivation tree for a string in a weighted grammar is not sufficient. We must also find one that is maximal.

Various methods for parsing restricted weighted grammars are available. However, the technique we will use in this paper is similar to the dynamic programming method used in the CYK algorithm for Chomsky Normal Form context-free grammars [21]. We choose this approach because it is efficient and it admits a massively parallel implementation as well as a straight forward implementation using feedforward neural networks. From the neural network, we can consider questions of relaxation and noise.

Definition. A weighted context-free grammar $G = (\Gamma, N, P, S)$ is in *Chomsky Normal Form* if all weighted productions of G are of the forms

$$A \xrightarrow{p} BC \quad \text{or} \quad A \xrightarrow{p} a,$$

where $A, B, C \in N$ and $a \in \Gamma$.

Theorem 4.4. (Chomsky Normal Form) For every weighted context-free grammar G, there exists a weighted context-free grammar G' in Chomsky Normal Form such that L(G) = L(G').

Notation. Z will denote the set of all nonnegative integers and Z^+ the set of all positive integers.

Definition. An *index* is an ordered pair (i, j), where $i, j \in Z^+$. The collection of all indices will be denoted by IDX. If $x \in V^*$, then IDX(x) will denote the set of indices (i, j) where $i + j \leq |x| + 1$.

Notation. If $x = a_1a_2...a_n$ where $a_k \in \Gamma$ for $k = 1, 2, \ldots, n$, then $x(i, j) = a_ia_{i+1}...a_{i+j-1}$ for all $(i, j) \in IDX(x)$.

Input: A weighted context-free grammar $G = (\Gamma, N, P, S)$ in Chomsky Normal Form and an input string $x \in \Gamma^*$.
Output: W(A, i, j), together with its tags, for all $A \in N$ and $(i, j) \in IDX(x)$.
Method:

```
1)   set W(A, i, j) to 0 for all A ∈ N and (i, j) ∈ IDX(x).
     set all associated tags to NULL.
     set n to |x|.
2)   for j := 1 to n do
3)      for all A ∈ N such that P(A, x(j, 1)) = p > 0 do
4)         W(A, 1, j) := p
5)   for i := 2 to n do
6)      for j := 1 to n - i + 1 do
7)         for all A, B, C ∈ N such that P(A, BC) = p > 0 do
8)            for m := 1 to i - 1 do
                 begin
9)                  W(A, i, j) := p ⊗ W(B, m, j) ⊗ W(C, i-m, j+m)
10)                 if W(A, i, j) > 0 do
11)                       set tags for W(A, i, j) to point to
                             W(B, m, j) and W(C, i-m, j+m)
                 end
```

Algorithm 4.1. Weighted CYK algorithm for weighted context-free grammars in Chomsky Normal Form.

Remark. In the loop from lines 7 to 11, we assume the weighted production rules are ordered in some arbitrary manner. This ordering will determine the tags unambiguously.

Once Algorithm 4.1 is completed, we simply follow the tags from the designated start symbol, W(S, n, 1), to form the maximal G-derivation tree for x together with its associated probability, if it exists.

Example 4.1. Consider the weighted context-free grammar:

S	→	AB	.7
S	→	BC	.8
A	→	BA	.4
A	→	a	.6
B	→	CC	.3
B	→	b	.7
C	→	AB	.9
C	→	a	1.0

and the input string *baaba* [21]. Figure 4.1 shows the state of the parsing matrix after a complete run of Algorithm 4.1.

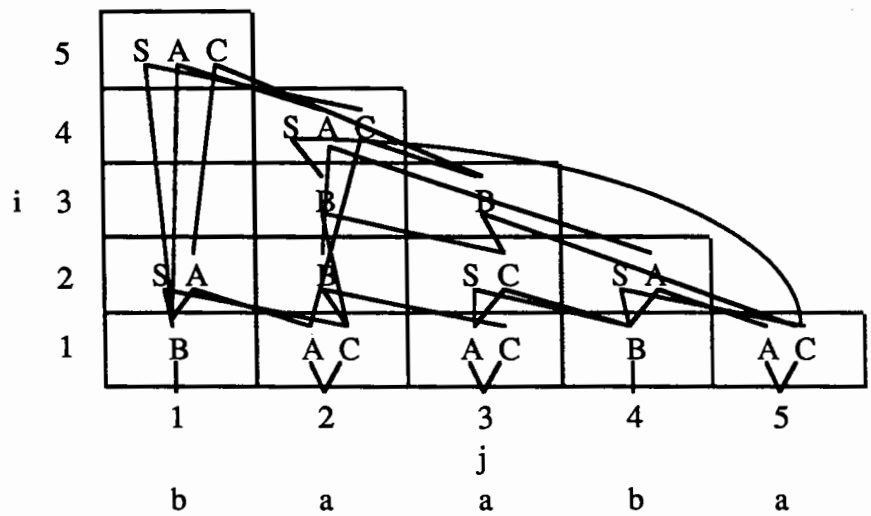

Figure 4.1. Weighted CYK parsing with tags and K = 1.

Matrix value :

B_{11}	=	.7			
A_{12}	=	.6	C_{12}	=	1.0
A_{13}	=	.6	C_{13}	=	1.0
B_{14}	=	.7			

A_{15}	=	.6	C_{15}	=	1.0			
S_{21}	=	.56	A_{21}	=	.168			
B_{22}	=	1.0						
S_{23}	=	.294	C_{23}	=	.378			
S_{24}	=	.56	A_{24}	=	.168			
B_{32}	=	.1134						
B_{33}	=	.1134						
S_{42}	=	.09072	A_{42}	=	.0672	C_{42}	=	.061236
S_{51}	=	.30429216	A_{51}	=	.018816	C_{51}	=	.01714608

We now briefly prove the correctness of the above algorithm.

In Theorems 4.5 to 4.8 below, $G = (\Gamma, N, P, S)$ is a weighted context-free grammar in Chomsky Normal Form and $x \in \Gamma^*$.

Theorem 4.5. If Algorithm 4.1 is applied to G, x, and K, then upon termination, for all $A \in N$ and $(i, j) \in IDX(x)$, W(A, i, j), together with its tags, if exists, constitute the G-derivation tree T for x(i, j) from A with the maximal G-derivational probability.

Proof. Follows from lines 7 to 11 in Algorithm 4.1. ♦

Remark. After executing our weighted CYK algorithm on some string x, if we choose some W(A, i, j) and build a tree by following its tags, we have shown that the resulting tree is the maximal G-derivation tree for x(j, j+i-1) from A. We will call the forest of all such trees, a G-derivation forest for x.

Theorem 4.6. If Algorithm 4.1 is applied to G and x, then upon termination,

$$DP_G(A, x(i, j)) = W(A, i, j)$$

for all $A \in N$ and $(i, j) \in IDX(x)$.

Proof. Follows from Theorems 4.2 and 4.5. ♦

Theorem 4.7. If Algorithm 4.1 is applied to G and x, then upon termination,

$$\chi_{L(G)}(x) = W(S, n, 1).$$

Proof. Follows from Theorems 4.3 and 4.6. ♦

The CYK algorithm is called an all-paths parsing algorithm because it explores all possible paths in parallel without backtracking making it amenable to massive parallelization as we shall see in the next section.

5. Feedforward Neural Network Implementation

Although many approaches are available for building our weighted CYK algorithm in a parallel manner, we are particularly interested in using structured neural networks because of the useful perspective provided by the approach and its inherent relaxation properties.

We begin by simply building portions of the network to correspond closely to struc-

tures used in our algorithm. For example, production rules for the target grammar will correspond closely to a type of node called *rule units*. This structuring allows an easier understanding of the operation of the network and can also supply a useful visual interpretation. Its from these observations that we are able to extend our system to handle ungrammaticality as we shall see later in the next section.

Our network consists of four type of units: *terminal units, nonterminal units, tag units,* and *rule units.*

Terminal units serve as the input to our system since they are activated by an outside source. We associate with each column in our CYK matrix a set of these units. Given some set, each unit belonging in the set represents a single terminal symbol. Its activation value (from 0 to 1) will denote the likelihood that the terminal it represents occurs in this position. In its simplest application, only one unit from each set will have an activation value of 1 and all others will be at 0.

Nonterminal units are arranged in a tabular format identical to the CYK matrix. Each table entry will be represented by a collection of nonterminal units each representing an individual nonterminal's likelihood of appearing in this entry.

Tag units correspond to the tags for our weighted CYK algorithm. The tag units will explicitly represent the final parsing structure and thus serves as the output for our system. Each tag unit will denote the combination of three nonterminal units -- a designated root unit and two designated subroot units. This corresponds to the root of a tree and the roots of its two subtrees. Whenever two subtrees maybe joined to form a single subtree according to our grammar, we allocate a tag unit to represent this phenomenon. Thus, for each legal combination or joining of nonterminals, we have a representative tag unit. Tag units do not physically connect nonterminal units but abstractly represent the joining.

The above units are now connected together via the *rule units*. Whenever two nonterminal units can be combined or joined to form a larger tree, we allocate a rule unit. The rule units will take input from the nonterminal units it wishes to join. It combines its inputs by performing the operator $\otimes$ on them. The unit will then output this value to the nonterminal unit that is to be the new root. Since each rule unit is formed from some production rule, the output of the unit will be further modified by the weight of the production rule. Also, the unit will output to the appropriate tag unit representing this particular joining.

Since each nonterminal unit will receive input from several rule units, it final activation value is simply the maximum of all its inputs.

Given a string x, we activate the appropriate terminal units. Then we allow the information to propagate through the network until it stabilizes. This usually requires only $|x|$ iterations of the network where an iteration is a single synchronous firing of all units. Now, we can easily determine the probability of membership for x by examining the value of the start symbol at the apex of the table.

From each group of tag units with identical root units, we choose the largest tag unit. All the tag units chosen will represent the maximal derivation forest for x. Since there maybe two or more tag units in a group which are largest due to ambiguity, we can simply modify the weights of the rule units as follows: if two rule units which output to the same nonterminal unit have the same weight, then decrease the weight of one of the rule units by an arbitrarily small amount. This guarantees a single largest choice for tag units and will not affect the order of extraction by much for further parsings of x.

It is easy to show that this feedforward neural network is equivalent to the weighted CYK algorithm. Much of the structure and operation of the algorithm has been directly mapped into the network. As we can easily see, this network is a highly structured network which was constructed in a very straightforward manner.

An earlier parser built with connectionist networks based on the conventional CYK algorithm [12] could parse strings for a Chomsky Normal Form CFG. However, that parser will not work with weighted grammars. Another limitation of Fanty's system is its inability to handle both ambiguity and ungrammaticality. As we shall see later, the use of weighted grammars will enable us to better formulate and handle both ill-formed sentences and ambiguities.

6. Ill-formed Sentence Processing

In the CYK algorithm, one attempts to parse a string by building derivation trees for its smaller substrings and then combining them together to form derivation trees for larger substrings. If a string belongs to the language generated by the given grammar, we will eventually end up with a complete derivation tree for it. Otherwise, there will be no derivation tree. However, since ill-formed sentences are not grammatical, CYK would essentially ignore them.

Ill-formed strings, by definition, do not belong in the language. Thus, no derivation tree will exist for these types of strings. Our strategy is based on viewing ill-formed strings as strings that would have been grammatical except for some errors occurring during its derivation.

Example 6.1. (Ill-formed string derivation) Given the following CFG:

S	→	NP	VP
S	→	NP1	VP
S	→	NOUN	VP
S	→	NP	VP1
S	→	NP1	VP1
S	→	NOUN	VP1
NP	→	DET	NP1

NP	→	DET	NOUN
NP	→	ADJ	NP1
NP	→	ADJ	NOUN
NP1	→	ADJ	NP1
NP1	→	ADJ	NOUN
VP	→	AUX	VP1
VP1	→	VERB	NP
DET	→	the | a	
NOUN	→	dog | man | cat | hat	
ADJ	→	red | big | orange	
AUX	→	has	
VERB	→	bitten | worn	

We will now attempt to derive "the orange cat has bitten a big dog":

S → NP VP → DET NP1 VP → the NP1 VP

→ the ADJ NOUN VP → the orange man NOUN VP

At this point in the derivation, we have inadvertently introduced an extra word "man" before introducing "cat" as the replacement for NOUN.

→ the orange man cat AUX VP1 → the orange man cat hat VP1

Here, we have made the mistake of substituting "has" for "hat" for the nonterminal AUX.

→ the orange man cat hat VERB NP

→ the orange man cat hat bitten NP

→ the orange man cat hat bitten DET NP1

→ the orange man cat hat bitten a NP1

Finally, we have made the mistake of deleting NOUN from the production NP1 → ADJ NOUN.

→ the orange man cat hat bitten a ADJ

→ the orange man cat hat bitten a big

Our resulting sentence is "the orange man cat hat bitten a big" instead of "the orange cat has bitten a big dog."

Remark. In our example, we used the three most basic types of errors: insertion, deletion, and substitution for illustrative purposes only. We will be considering methods for modeling other more complicated types of errors such as reversal, phrasal substitution, and so on.

A way to view our weighted CYK parser's inability to handle ungrammaticality is in terms of the system's "rigidness." Like any standard grammar parser, all actions must strictly conform to the set of production rules provided. By loosening this "rigidness", we hope to provide a mechanism to handle ill-formed strings.

By looking at the neural network implementation of our weighted CYK parser, we can gain insight into the nature of the system's rigidness and how to go about its loosening.

The neural network provides us with a strong visualization of operation and structure.

The most obvious location of rigidness is in the rule units. Each rule unit corresponds to a proper application of some production rule. By allowing some "improper" applications, we are essentially attempting local repairs. Another potential area of repair is at the nonterminal and terminal units. Such repairs can be done with the addition of new nodes called *repair units* to the network.

In the case of insertion, we make the following observation from the neural network: To form a new constituent from two or more existing constituents, the constituents must represent adjacent substrings. By changing the restriction that they must be adjacent to just being consecutive, we allow substrings to occur between two or more constituents and treat them as extra terminals to be ignored. This can be easily modeled by a repair unit which mimics a rule unit but can take inputs for consecutive but not necessarily adjacent nonterminal units. The weight on the output of this new repair unit will usually be a percentage of the original rule unit it was meant to repair.

In the case of reversal, we observe that to form a new constituent from two existing constituents, the two existing constituents must be in the prescribed order. By loosening the restriction that they must be in the right order, we allow the substrings to occur in reversed order. Again, a repair unit can be simply introduced to model this loosening.

In the case of deletion, we have a somewhat difficult problem as we shall see. A constituent has been removed and must somehow be recovered. When forming a new constituent, there must exist two or more constituents. When one or more of them are missing, we must somehow guess which one or ones are missing from those that remain. Our repair strategy entails such a guess and allows the system to continue.

Since the missing constituent(s) may represent any number of possible substrings, it would be inefficient to try to choose one over the other. Also, if it were possible to choose a likely candidate, this implies that we must somehow accommodate this new substring into our representation. If we did this every time we took a guess, it would be very difficult, if not impossible, to maintain all of them. Instead, we assume that the new constituent need only be represented by its root node.

If the new constituent is formed by only one old constituent, then since no new substring is introduced, the index of the new constituent will be the same as the old constituent, that is, in terms of the CYK matrix, the constituents occupy the same entry. Looking at our weighted CYK algorithm, when a new tree is created after applying a weighted production, the index of the new tree reflects the combination of the indices of those trees from which the new tree is created. Thus, the question comes up on where should this new constituent formed by the unit production be placed. One possible solution is to simply rely strictly on the meaning of the positions in the CYK matrix. Since this new constituent will only contain a single substring, the new constituent will appear in the same

position with the existing constituent. For the new constituent to appear anywhere else would violate representation of the matrix. In terms of tags, tags can now also point to constituents which occur at the same location.

The biggest pitfall of this solution is that the process can now have loops occurring inside each matrix entry. However, it can be proven that even with such loops the network will converge and provide a predictable answer [22].

Many other types of ungrammaticalities can be modeled through the use of additional repair units. For example, dyslexia, phrasal substitution, zero-copula, context-sensitive errors, and combinations of simpler error types.

7. Conclusion

Structured neural networks integrate symbolic and subsymbolic approaches creating an interesting alternative approach for NLP. High level symbolic processes as well as lower level robustness makes these networks a promising candidate for work on problems such as ungrammaticality as we have just seen. The easy visual understanding of the process at hand also permits different and possibly new perspectives to the problem.

With one such perspective, we find that ungrammaticality can be naturally approached from the viewpoint of noisy data using neural networks. The capability to handle relaxation and to represent uncertainty is obviously evident in neural networks and can be easily exploited as we saw above.

Combined with probabilistic grammars, this approach to ungrammaticality can also be fully modeled mathematically [22] and presents an alternative framework for robust parsing. Certain properties such as convergence, correctness, and the possible use of different classes of repair techniques can be proven and predicted.

In conclusion, using structured neural networks to handle the problem of ungrammaticality seems very promising. Current experiments using subsets of the English language have shown the system to be efficient and practical at handling ill-formed sentences. In comparison to existing systems, there are indications that our approach may be faster and more capable of handling a larger assortment of deviations overall.

Further work on this approach should consider questions such as learning, unbounded length sentences, and psychological plausibility.

References

[1] Aho, A. V. and Peterson, T. G. (1972), A minimum distance error-correcting parser for context-free languages, *SIAM J Comput* **4**, 305-312.

[2] Hendrix, G. C. (1977), Human engineering for applied natural language process-

ing, *Proc. of the 5th IJCAI*, 183-191.

[3] Tanaka, E. and Fu, K. S. (1978), Error-correcting parsers for formal languages, *IEEE Trans. Comput.* **C-27**, 605-616.

[4] Eastman, C. M. and McLean, D. S. (1981), On the need for parsing ill-formed input, *AJCL* **7**.

[5] Kwasny, S. C. and Sondheimer, N. K. (1981), Relaxation techniques for parsing ill-formed input, *AJCL* **7**, 99-108.

[6] Carbonell,J. G. and Hayes, P. J. (1983), Recovery strategies for parsing extragrammatical language, *AJCL* **9**:3-4, 123-146.

[7] Weischedel, R. M. and Sondheimer, N. K. (1983), Meta-rules as a basis for processing ill-formed input, *AJCL* **9**, 161-177.

[8] Linebarger, M. C. , Dahl, D. A., Hirschman, L. and Passonneau, R. J. (1988), Sentence fragments regular structures, *Proc. 26th Annual Meeting of the ACL*, 7-16.

[9] Mellish, C. S. (1989), Some chart-based techniques for parsing ill-formed input, *Proc. 27th Annual Meeting of the ACL*, 102-109.

[10] Rumelhart, D. E. and McClelland, J. L. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Volumes 1 and 2*," Bradford Books / MIT Press, Cambridge, MA.

[11] Servan-Schreiber, D., Cleeremans, A. and McClelland, J. L. (1988), Encoding sequential structure in simple recurrent networks, Computer Science Department, Carnegie-Mellon University, CMU Tech Report **CMU-CS-88-183**, Pittsburgh, PA.

[12] Hanson, S. J. and Kegl, J. (1987), PARSNIP: a connectionist network that learns natural language grammar from exposure to natural language sentences, *Proc. Ninth Annual Cognitive Science Soc. Conf.*, 106-119, Seattle, WA.

[12] Fanty, M. (1988), Learning in structure connectionist networks, Computer Science Department, University of Rochester, Tech Report **#252**, Rochester, NY.

[13] Waltz, D. L. and Pollack, J. B. (1985), Massively parallel parsing: a strongly interactive model of natural language interpretation, *Cog Sci* **9**, 51-74.

[14] Charniak, E. and Santos, E. Jr. (1987), A connectionist context-free parse which is not context-free, but then it is not really connectionist either, *Proc. Ninth Annual Cog. Sci. Conf.*, 70-77, Seattle, WA.

[15] Santos, E. Jr. (1989), A massively parallel self-tuning context-free parser, *Advances in Neural Information Processing Systems 1* (Ed. D. Touretzky), 573-544, Morgan Kaufmann, San Mateo, CA.

[16] Lange, T. E. and Dyer, M. G. (1989), Dynamic, non-local role bindings and inferencing in a localist network for natural language understanding, Computer

Science Department, University of California, Technical Report **UCLA-AI-89-01**, Los Angeles, CA.

[17] Barnden, J. A. (1988), The right of free association: Relative-position encoding for connectionist data structures, *Proc. of the Tenth Annual Conf. of the Cog. Sci. Soc.*

[18] Salomaa, A. (1969), Probabilistic and weighted grammars, *Inform. Contr.* **15**, 529-544.

[19] Santos, E. S. (1972), Probabilistic grammars and automata, *Inform. Contr.* **21**, 27-47.

[20] Fu, K. S. (1972), *Syntactic Methods in Pattern Recognition*, Academic Press, New York / London.

[21] Hopcraft, J. E. and Ullman, J. D. (1979), *Introduction to Automata Theory, Languages, and Computation*, Addision-Wesley, Reading, MA.

[22] Santos, E. Jr. (1991), A mathematical framework for robust parsing, Department of Computer Science, Brown University, Technical Report, Providence, RI.