Competition and Learning in a Connectionist Deterministic Parser[1]

Stan C. Kwasny

Kanaan A. Faisal

Department of Computer Science
Washington University

ABSTRACT

Deterministic parsing promises to (almost) never backtrack. Neural network technology promises generalization, competition, and learning capabilities. The marriage of these two ideas is being investigated in an experimental natural language parsing system that combines some of the best features of each. The result is a deterministic parser that learns, generalizes, and supports competition among structures and lexical interpretations.

The performance of the parser is being evaluated on predicted as well as unpredicted sentence forms. Several mildly ungrammatical sentences have been successfully processed into structures judged reasonable when compared to their grammatical counterparts. Lexical ambiguities can create problems for traditional parsers, or at least require additional backtracking. With the use of neural networks, ambiguities can be resolved through the wider syntactic context. The results have shown the potential for parsing using this approach.

INTRODUCTION

Any plausible model of language processing should permit alternative linguistic structures to compete while inputs are processed left-to-right. Computer models based on backtracking (e.g., Augmented Transition Networks (ATNs) or Definite Clause Grammars (DCGs)) do not adequately capture the competitive nature of sentence processing. Furthermore, there is no evidence from human experiments that any conscious re-processing of inputs is routinely performed. The lone exception is perhaps for "garden path" sentences.

A good example of competition can be found in the TRACE model of speech perception (McClelland & Elman, 1986). In that work, competing interpretations of the pseudo-speech feature vectors are proposed and activation levels rise or fall as each potential interpretation is supported or contradicted. Parsers should permit syntax and other levels of processing to aid in resolving lexical ambiguities just as ambiguous phonemes were resolved in TRACE.

In most neural network or connectionist parsers, grammar rules are processed into a network of units connected with excitatory and inhibitory links. The number of units required to realize a given grammar is a function of the maximum input sentence length and the complexity of the grammar. Hence, a limitation is introduced on the number of elements that can be present in the input. Sentences are processed within such a framework by presenting them, possibly in a simulated left-to-right fashion, at the input side of the network and activations are permitted to spread through the network (Cottrell, 1985; Fanty, 1985; Waltz & Pollack, 1985). Alternatively, a stochastic method, such as simulated annealing, is used (Selman & Hirst, 1985).

---

[1] Partial support for this work was received from the Center for Intelligent Computer Systems at Washington University.

Classically, parsers process inputs iteratively from an unbounded stream of input. Neural network parsers typically do not work iteratively and have limits imposed artificially on the length of the sentence. There is, however, work underway on neural network iteration mechanisms that could be used in parsers of natural language. (Servan-Schreiber, Cleeremans, & McClelland, 1988; Williams & Zipser, 1988).

In classic approaches, natural language processing by computer is performed under the direction of a set of grammar rules. These are often executed as if following instructions in a program. If the intent is to model human sentence processing, then this method is incorrect. Rules should be permitted to play an advisory role only — that is, as descriptions of typical situations and not as prescriptions for precise processing. Control in the application of a rule or variant of a rule should be determined jointly as a data-driven and expectation-driven process.

Symbolic rules are an essential part of most linguistic accounts at virtually all levels of processing, from speech signal to semantics. But systems based literally on rules tend to be brittle since there is no direct way to process linguistic forms that do not strictly adhere to the pre-conceived rules. If a complete set of rules for all meaningful English forms existed, then this might be satisfactory. But no such set of rules exists, nor does it seem desirable or even possible to construct such a set. Furthermore, the rules would have a difficult time capturing "degrees of grammaticalness" (Chomsky, 1965)[2].

Another consequence of a rule-based grammar is that acquisition of new grammar rules often require tedious re-tuning of existing rules. Rarely can a rule be added to the grammar without it affecting and being affected by other rules in the grammar. To the credit of their creators, some grammars have been continually refined over a period of years, even decades, in an attempt to more accurately depict the processing requirements of English. The only solution to this problem in a practical and realistic manner is through learning.

APPROACH

Our connectionist parser supports competition among sentence structures and performs sequentially over an unbounded input stream. In addition to parsing well-formed sentences, the parser is capable of parsing some types of ill-formed sentences and resolving some lexical ambiguities using syntactic context. Our model is based on a multi-level neural network, trained through backward propagation (Rumelhart, et al., 1986). It combines both symbolic and non-symbolic processing with actions of the rules carried out symbolically and decision-making carried out non-symbolically. Rules of the grammar are presented as training patterns of processing strategies, not as packets of infallible advice to be memorized and followed literally. Our design is based on deterministic parsing (Marcus, 1980) and iteration is an integral feature of the design.

Experimentation with a medium-size grammar has produced results which have been encouraging. Once trained, the network is quick, robust, and permits competition among processing alternatives. Training sequences are derived from two sources: (1) the rules of a rule-based deterministic grammar; and (2) traces of sentence processing steps from actual sentences. The former training is deductive while the latter training is inductive.

Deterministic Parsing

Deterministic, or "wait-and-see" parsing (WASP)[3] requires in the worst case that several (3 to 5) constituents of the input sentence be in view before deciding on the appropriate structure for the current constituent. Once this decision has been reached, it cannot be reversed and once structures have been constructed, they are never thrown away. Deterministic parsers are also rule-based in that their actions are

---

[2] There have been several expressions of this idea in the literature. Several psycholinguistic studies have attempted to measure the reality of this notion, both from a use as well as an interpretation perspective. Chomsky was selected as an important reference and one that illustrates a classic viewpoint.

[3] Waltz & Pollack (1985) characterize this option as one based on "delay" as opposed to one based on backtracking.

controlled by a collection of rules. The rules are partitioned into rule packets which aid in conflict resolution.

A single processing step in a traditional deterministic parser consists of selecting a rule to be fired from the appropriate rule packet and firing the rule to alter the structure and positions of constituents in the model. As with most rule-based systems, rules whose left-hand sides are found to match the state of the system correctly are eligible to be fired. Rule packets are activated as a consequence of which portion of the structure is being built and, within the packet, conflicts are resolved through a pre-assigned numeric priority and from the static ordering of rules within each priority value. Once selected, the rule is fired and its actions are performed. The action effects changes on the stack and buffer. After a series of processing steps, a termination rule fires, and the final parse structure is left on the top of the stack[4].

LEARNING A RULE-BASED GRAMMAR

Training proceeds by presenting patterns to the network and teaching it to respond with an appropriate action. The input patterns represent encodings of the buffer positions and the top of the stack. The output level of the network contains a series of units representing actions to be performed during processing and judged in a winner-take-all fashion. The training data are derived as "rule templates" from rules in a deterministic grammar. These rule templates are instantiated once in each epoch of training. Network convergence is observed once the network can achieve a perfect score on the rule templates themselves and the error measure has decreased to an acceptable level (set as a parameter). Once the network is trained, the weights are stored in a file so that various experiments can be performed with the network.

Network Architecture

Patterns in the pattern/action rules of the grammar consists of a list of syntactic features, divided into four groups to match the three buffer positions and the top of the stack. These are represented in a localist manner in the network with each syntactic feature being represented by a unit. The choice of a localist representation allows the grammar to be represented in a very straightforward manner and permits experimentation with sentence processing in a direct way.

In the set of experiments described here, the network has a three-layer architecture with 37 input units, 20 hidden units, and 20 output units. The input layer consists of four pools of input units, the first three pools represent the buffer, with each containing the features of a buffer item, and the fourth pool represents the top of the stack including the current node of the parse tree. One hidden layer has proven sufficient in all of our experiments. The output layer represents the 20 actions that can be performed on each iteration of processing.

During sentence processing, the network is presented with encodings of the buffer and the top of the stack. The network produces the action to be taken. If the action creates a vacancy in the buffer and if more of the sentence is left to be processed then the next sentence component is moved into the buffer. Iteration is achieved in this fashion.

Sentences

The grammar used is capable of processing a variety of simple sentence forms which end with a final punctuation mark. Simple declarative sentences, yes-no questions, imperative sentences, and simple passives are permitted by the grammar. What the model actually sees as input is not the raw sentence but a canonical representations of each word in the sentence, in a form that could be produced by a simple lexicon. Such a lexicon is not part of the model in its present form.

For test purposes, several sentences were coded that would parse correctly by the rules of the deterministic parser. Also, several mildly ungrammatical sentences were coded to determine if the network was

[4] This is an over-simplification of the processing involved, but accurately reflects accounts in many texts. A more accurate view, including a discussion of attention-shifting (AS) rules, rule priorities, etc., can be obtained from Allen (1987).

generalizing in any useful way. Finally, sentences containing ambiguously coded lexical items were presented to test if the context could aid in resolving such ambiguities.

Coding of Grammar Rules

In the canonical input format of a rule template, word forms are represented as a list of syntactic features. The set of possible features was chosen as necessitated by the grammar. In general each word form is represented by an ordered feature vector in which one or more values is ON(**+1**) for features of the form and all other values are either OFF(**-1**) or DO NOT CARE (**?**). A rule template is instantiated by randomly changing **?** to **+1** or **-1**.

Each grammar rule has the following format:

{ <Stack> <1st Item> <2nd Item> <3rd Item> → Action on Stack}

For example, a rule for Yes/No questions would be written as:

{ < S node > < Aux Verb > < NP > < > → Switch 1st and 2nd items }

A grammar rule is coded as a training template, which is a list of feature values. Each template represents many training patterns. On each training epoch every template is instantiated once yielding a specific training case. Thus, each training epoch is slightly different. Further details are available in Kwasny (1988a; 1988b).

Each input pattern consists of three feature vectors from the buffer items and one stack vector. Each vector activates 15 input units in a pattern vector representing a word or constituent of the sentence. The stack vector activates seven units representing the current node on the stack. In our simplified version of the grammar, only two items are coded from the buffer and thus 37 input units are sufficient.

Training from Rule Templates

Training consists of the presentation of 200,000 epochs of 23 training cases generated from 23 grammar rule templates[5]. The templates are not organized into rule packets nor grouped in any way as in the deterministic grammar. The probability of a **?** becoming a **+1** or **-1** is equal and set at 0.5. All weights in the network are initialized to random values between -0.3 and +0.3. After the presentation of each pattern, an error signal is derived from comparing activation on the output layer (the network's prediction) with the desired output pattern. That error signal is back-propagated through all the connections and the weights adjusted before presenting the next pattern[6].

Each rule template containing n **?**'s can generate up to 2^n training cases. Some rule templates have over 30 **?**'s which means they represent approximately 10^9 unique training cases. It is obviously impossible to test the performance of all these cases, so testing from rule templates involves substituting a zero value for each **?**. Zero is used since it represents the mean of the range of values seen.

PERFORMANCE

Each sentence receives a score representing the overall average strength of responses during processing. The score for each processing step is computed as the reciprocal of the error for that step. The error is computed as the Euclidean distance between the actual output and an idealized output consisting of a **-1** value for every output unit except the winning unit which has a **+1** value. The errors for each step are summed and averaged over the number of steps[7]. The average strength is the reciprocal of the average

---

[5] A slightly modified version of the grammar from appendix C of Marcus (1980) was used for all experiments in this paper. This appendix contains the list of rules specifically discussed in his thesis and can be taken to represent illustrations of the basic mechanisms. These have been coded into rule templates within our system for training.

[6] A slightly modified version of VICE, a program developed by John Merrill, was used for all simulations reported in this paper. Values of learning rate and momentum (*eta* and *alpha* in Rumelhart, et al. (1986)) were chosen sufficiently small to avoid large oscillations and were generally in the range of 0.01 to 0.02 for learning rate and 0.5 to 0.9 for momentum over a range of test runs.

[7] This sum is just the total-sum-of-squares (tss) used, for example, in the PDP software (McClelland & Rumelhart, 1988).

TABLE 1: Grammatical Sentences Used in Testing

	Sentence Form	Average Strength
(1)	John should have scheduled the meeting.	283.3
(2)	John has scheduled the meeting.	240.8
(3)	Has John scheduled the meeting?	132.2
(4)	John is scheduling the meeting.	294.4
(5)	Schedule the meeting.	236.2
(6)	The boy did hit Jack.	298.2
(7)	John is kissing Mary.	294.4
(8)	Mary is kissed.	276.1
(9)	Tom hit Mary.	485.0
(10)	Tom will hit Mary.	547.5
(11)	They can(v) fish(np).	485.0
(12)	They can(aux) fish(np).	598.2

error per step.

Parsing Grammatical Sentences

Grammatical sentences, by our definition, are those which parse correctly in the rule-based grammar from which we derived the training set. Table 1 shows several examples of grammatical sentences which are parsed successfully along with their response strengths. Each example shows a relatively high average strength value, indicating that the rules used in training have been learned.

During parsing, the input sentence is presented in the input buffer from left to right. On each iteration, the network is presented with constituents from the input buffer and the entry from the top of the stack. The action specified by the network is performed and the buffer and stack are updated as required. New input items replace empty buffer positions as needed. The process then repeats until a stop action is performed, usually when the buffer becomes empty.

Parsing Ungrammatical Sentences

An important test of the system's generalization capabilities is its response to ungrammatical sentences. This is strictly dependent upon its experience since no relaxation rules were added to the original grammar to handle such ungrammatical cases. This experiment consisted of testing a few ungrammatical sentences that were close to the training data and within the scope of our encoding. Table 2 contains examples that have produced reasonable structures when presented to our system. Note that overall average strength is lower for ungrammatical sentences when compared to similar grammatical ones.

In sentence (13), the structure produced resembled that produced while parsing sentence (1). The only difference was that the two auxiliary verbs, *have* and *should*, were in reverse order. Sentence (14) contains a disagreement between the auxiliary *has* and the main verb *schedule* and yet the comparable grammatical sentence (3) parsed identically, but with more strength in the network's responses. Similarly, sentence (15) is comparable to sentence (4) in its processing steps. For sentence (16), Charniak (1983) reports that his system, PARAGRAM, produces a nonsensical parse structure. In our parser, this sentence succeeds and produces a structure which resembles one for sentence (6).

Lexical Ambiguity

In a final set of experiments, the parser was tested for its ability to aid in the resolution of lexical ambiguity. Grammatical sentences were presented, except that selected words were coded ambiguously to represent an ambiguously stored word from the lexicon. These examples are shown in Table 3. Several of these examples come from Milne (1986).

TABLE 2: Ungrammatical Sentences Used in Testing

	Sentence Form	Average Strength
(13)	*John have should scheduled the meeting.	25.1
(14)	*Has John schedule the meeting?	38.1
(15)	*John is schedule the meeting.	4.7
(16)	*The boy did hitting Jack.	26.6

TABLE 3: Lexically Ambiguous Sentences Used in Testing

	Sentence Form (Words in <> are presented ambiguously)	Average Strength
(17)	<Will> he go?	83.6
(18)	Tom <will> hit Mary.	118.7
(19)	Tom <hit> Mary.	39.0
(20)	They <can> fish.	4.5
(21)	They can <fish>.	172.2

Sentence (17) contains the word *will* coded ambiguously as an NP and an auxiliary, modal verb. In the context of the sentence, it is clearly being used as a modal auxiliary and the parser treats it that way. A similar result was obtained for sentence (18). In sentence (19), *hit* is coded to be ambiguous between an NP (as in playing cards) and a verb. The network correctly identifies it as the main verb of the sentence. Sentence (20) presents *can* ambiguously as an auxiliary, modal, and main verb, while *fish* is presented uniquely as an NP. *Can* is processed as the main verb of the sentence. Compare this example with sentence (11) of Table 1. Here, each word is presented unambiguously with *can* coded as a verb and *fish* coded as an NP. The same structure results in each case, with the average strength level much higher in the unambiguous case. By coding *fish* ambiguously as a verb/NP and coding *can* uniquely as an auxiliary, the result obtained is as shown for sentence (21), which is comparable to sentence (12).

In the cases shown, the lexically ambiguous words were disambiguated and reasonable structures resulted. Note that the overall average strengths were lower than comparable grammatical sentences discussed, as expected.

DISCUSSION

Robust language processing has been demonstrated in our model for selected, mildly ungrammatical sentence forms as well as for some types of lexical ambiguity. A network model of language processing has been trained on an encoded set of rules and tested on a variety of problematic forms. Results have been good with expected sentence forms evoking higher response strengths in general than unexpected forms.

The robust property of our parser is one of the most important reason for considering this approach. Attempts to process ill-formed inputs using conventional (symbolic) means, though successful in limited ways, have generally resulted in somewhat ad hoc methodologies that are tedious to use and have their own "sharp edges" in performance[8]. As mentioned earlier, Charniak (1983) attempted to provide for

[8] For further discussion of symbolic approaches, see Kwasny & Sondheimer, (1981); Weischedel & Sondheimer, (1983); Weischedel & Ramshaw, (1987).

parsing ungrammatical sentences in a deterministic grammar. His method is to score each possible test from the pattern portion of a rule and execute the rule with the best score. Our network provides its own scoring mechanism refined during learning.

Competition in our network among sentence processing alternatives has been observed. In our winner-take-all network, there can be only one action taken on each step. In ambiguous situations, however, there are often two or more competing actions which reflect alternative processing sequences. This is true in the ungrammatical and lexically ambiguous examples which often have multiple grammatical counterparts. This feature of the processing is a necessary part of parsing. With the absence of outside influences in our parser, e.g., semantics or the context of a dialogue, the network provides a choice based solely on its training experiences.

A single neural network-based parser trained on a deterministic grammar without rule packets has been shown to generalize to some cases not acceptable to the grammar. The grammar is therefore being used in an advisory role. Indeed, sentence forms which fall under the jurisdiction of the grammar parse with minimal error and thus universally earn a higher strength score than its ungrammatical counterpart.

In a brief experiment on inductive learning, the network was trained on the grammatical sentences used in our tests and its performance was tested on the rule templates. For those rule instances that were represented in the training data, the system did well, but overall exhibited less generalization due to the lack of extensive training cases. Overall strength was low, except for the precise sentences for which it was trained. As these experiments are continued and more sentence examples are used, better generalization is expected.

FUTURE DIRECTIONS

There are several directions in which our work is progressing. Some of the recent work on recurrent networks is being examined with the hope of improving the iteration properties of our system. Ultimately, it should be sufficient to present a final encoded structure as teaching data for a sentence and permit the system to organize itself into the appropriate number and kind of processing steps necessary to build it. Although achieving this will not happen soon, this work is expected to move away from the present very strong dependence on the organization associated with classic deterministic parsing.

Our choice of encoding was based on its simplicity and directness. Now that our experiments have shown how generalization can be achieved, our representation of the structures being built and the stack being used should be improved (Pollack, 1988). Our coding scheme is also being expanded to include a more complete set of features, for example, person and number as well as other labels that can appear in final structures. Eventually, the output layer should produce an updated encoding of the input and not require that the action be performed externally. As our understanding of the capabilities of this approach increases, the grammar will be scaled up to a much larger grammar of English. The limits of the generalization capability demonstrated here need to be further probed.

Still to be addressed are issues at the semantic and lexical levels. Our feature vectors purport to capture the patterns of activation that a lexical component would produce. Experiments are ongoing in this area.

Finally, garden path sentences need to be better understood. These should not be dismissed as simply different or anomalous. There is hope within our framework for an attack on these defiant sentence forms.

ACKNOWLEDGEMENTS

The authors express gratitude to William Ball, Steve Cousins, Georg Dorffner, Rose Fulcomer, David Harker, Dan Kimura, Ron Loui, John Merrill, Robert Port, and Guillermo Simari for thoughtful discussions and comments concerning this work. We, of course, are responsible for errors.

REFERENCES

Allen, J. (1987). *Natural Language Understanding*. Menlo Park: Benjamin/Cummings.

Charniak, E. (1983). A Parser with Something for Everyone. In King (Ed.), *Parsing Natural Language*, New York: Academic Press.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. Cambridge: MIT Press.

Cottrell, G.W. (1985). Connectionist Parsing. *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, Irvine, CA, 201-211.

Fanty, M. (1985). Context-Free Parsing in Connectionist Networks. (Technical Report 174), Computer Science Department, University of Rochester, Rochester, NY.

Kwasny, S.C. (1988a). A PDP Approach to Deterministic Natural Language Parsing. *Neural Networks 1*, Supplement 1, 305.

Kwasny, S.C. (1988b). A Parallel Distributed Approach to Parsing Natural Language Deterministically. (Technical Report WUCS-88-21), Department of Computer Science, Washington University, St. Louis, MO.

Kwasny, S.C., & Sondheimer, N.K. (1981). Relaxation Techniques for Parsing Ill-Formed Input. *American Journal of Computational Linguistics 7*, 99-108.

Marcus, M. P. (1980). *A Theory of Syntactic Recognition for Natural Language*. Cambridge: MIT Press.

McClelland, J.L., & Elman, J.L. (1986). The TRACE Model of Speech Perception. *Cognitive Psychology 18*, 1-86.

McClelland, J. L., & Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, Cambridge: MIT Press.

Milne, R. (1986). Resolving Lexical Ambiguity in a Deterministic Parser. *Computational Linguistics 12*, 1-12.

Pollack, J. (1988). Recursive Auto-Associative Memory: Devising Compositional Distributed Representations. (Report MCCS-88-124), New Mexico State University Las Cruces, NM.

Rumelhart, D. E., Hinton, G., & Williams, R.J. (1986). Learning Internal Representations by Error Propagation. In Rumelhart & McClelland *Parallel Distributed Processing*. Cambridge: MIT Press.

Selman, B. & Hirst, G. (1985). A Rule-Based Connectionist Parsing System. *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, Irvine, CA, 212-221.

Servan-Schreiber, D., Cleeremans, A., & McClelland, J.L. (1988). Encoding Sequential Structure in Simple Recurrent Networks. (Report CMU-CS-88-183), Carnegie Mellon University, Pittsburgh, PA.

Waltz, D.L. & Pollack, J.B. (1985). Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation. *Cognitive Science 9*, 51-74.

Weischedel, R.M. & Sondheimer, N.K. (1983). Meta-Rules as a Basis for Processing Ill-Formed Input. *American Journal of Computational Linguistics 9*, 161-177.

Weischedel, R.M. & Ramshaw, L. (1987). Reflections on the Knowledge Needed to Process Ill-Formed Language. In S. Nirenburg (Ed.), *Machine Translation: Theoretical and Methodological Issues*. Cambridge: Cambridge University Press.

Williams, R. J., & Zipser, D. (1988). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. (ICS Report 8805), University of California, San Diego, CA.