

# AN ADAPTIVE NEURAL NETWORK PARSER

**SAHNNY JOHNSON**

*Dept. of Computer Science, Knox College, Galesburg, IL*

**STAN C. KWASNY AND BARRY L. KALMAN**

*Dept. of Computer Science, Washington University, St. Louis, MO*

## **ABSTRACT:**

We investigate the applicability of an adaptive neural network to problems with time-dependent input by demonstrating that a deterministic parser for natural language inputs of significant syntactic complexity can be developed using recurrent connectionist architectures. The traditional stacking mechanism, known to be necessary for proper treatment of context-free languages in symbolic systems, is absent from the design, having been subsumed by recurrency in the network.

## **INTRODUCTION**

Applications of adaptive neural networks to problems with time-dependent input have yielded results manifesting considerable flexibility. One such problem area, natural language processing, is seen by some as pivotal to the success or failure of connectionism. Pinker and Prince (1988, p.78) stated that “*Connectionism, as a radical restructuring of cognitive theory, will stand or fall depending on its ability to account for human language.*” Whether or not one accepts the full force of this contention, it issues a clear challenge that is to a large extent the focus of our work.

Starting from the basic design of a deterministic natural language parser, we are introducing and evaluating connectionist techniques in a series of evolutionary steps toward a fully connectionist language understanding system. Deterministic or wait-and-see parsing (Marcus, 1980) is an appropriate model for the design of a neural net based parser in two important ways. First, it concentrates on syntax, which has the advantage of being relatively well understood. Second, the fixed-length input buffer of a deterministic parser matches the input strategy of recurrent neural networks, such as those due to Elman (1990), which iteratively process a fixed piece of a larger input. This makes for a natural fit between a deterministic approach to natural language processing and the processing strategy of a recurrent net.

In his determinism hypothesis, Marcus (1980, p.204) conjectured that “*there is enough information in the structure of natural language ... to allow left-to-right deterministic parsing of those sentences which a native speaker can analyze without conscious effort.*” A clear implication of this hypothesis is that natural language processing need not depend in any fundamental way on classical backtracking. Furthermore, no partial structures are required during parsing which will not later become part of the final structure — that is, there need be no false starts. The

---

This material is based upon work supported by the National Science Foundation under Grant No. IRI-9201987.

hypothesis does not include any details about the architecture or the rules. It does limit sentences to those that a native speaker can analyze without conscious effort, which eliminates garden-path sentences such as "The horse raced past the barn fell."

Marcus constructed PARSIFAL, a system that purported to examine this hypothesis but which is tied to a specific architecture involving a stack, a set of often complicated rules, and its own limitations. We are testing the hypothesis directly by developing a corpus of sentences which a native speaker can analyze without conscious effort, and constructing a neural net architecture that accepts a buffer of sentence elements fed in sequentially as input and produces structure building and manipulating (primitive) actions as output. Unlike recurrent parsing networks described elsewhere (see, for example, Das et al., 1992), our system contains no explicit internal or external stack. The stack's role is assumed to a sufficient degree by the recurrence in the network. In addition, we do not dwell on artificially contrived languages (like  $a^n b^n$ ), but adopt the artificial intelligence tradition of demonstrating performance over natural language examples.

The test of whether there is enough information in the structure of natural language to allow deterministic parsing lies in the test of whether network training succeeds. Training data for the network is developed by tracing the symbolic processing of a corpus of (non-garden-path) sentences; testing is performed on a set of similar but not identical sentences. Generalization is shown to occur.

## **DETERMINISTIC PARSING**

Our model of deterministic parsing is based loosely on PARSIFAL. In that system, parsing is performed deterministically by permitting the parser to look ahead at the next three constituents of the sentence, held in a three-place buffer. A stack is required in PARSIFAL to allow the recursive processing of embedded structures and to facilitate processing generally. Primitive actions which build structure and move constituents can be performed on the stack and in the buffer positions. Rules are usually associated with the current (top-level) node of the structure being built, which is held on the top of the stack. A processing step consists of selecting an applicable rule and firing the rule by performing its action, forcing changes to the stack and/or buffer. After a series of steps, a termination rule fires which ends the processing and the final structure is left on top of the stack.

In earlier work, we replaced the rules used by PARSIFAL with a feedforward neural network trained from either rules or sentences. Our goals in that work were to unify PARSIFAL and a variety of proposed extensions and amendments within one model and to test the ability of the parser to address issues of ambiguity and ill-formedness. The network learned a mapping from the patterns used to encode the sentence elements in the buffer and the structure on top of the stack to the appropriate rule and its action. The three-place buffer, stack, and actions were all handled symbolically. Figure 1 (a) illustrates the network design used in our earlier work (Kwasny & Faisal, 1990; Kwasny & Kalman, 1991; Kwasny & Faisal, 1992). Casting the rules as a neural network yielded significant advantages in robustness.

In our current work, an adaptive (or recurrent) network is utilized which permits some degree of memory of past decisions. Although stack-like processing is necessary to address embeddings of the type present in all context-free languages, the

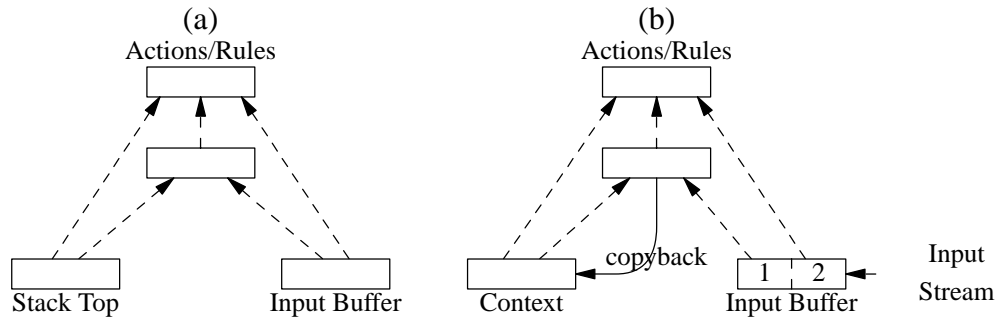


FIGURE 1. Network architectures. Dotted arrows represent full connectivity between layers.

recurrent design, shown in Figure 1 (b), obviates the need to present the top-of-stack contents to the network as in our previous work because sufficient information is represented in the recurrent connections. Because the choice of appropriate rule in a symbolic deterministic grammar is largely dependent on information stored on the stack, heavy responsibility in our adaptive network falls on the recurrent connections. The network is coerced, through training, to produce the proper push (create) and pop (drop) actions that traditionally would manipulate the stack without explicitly representing the stack. The output is a list of actions to be performed on the symbolic structures that evolve during processing.

During processing, the activation pattern of the hidden layer in the recurrent network is copied back to the input layer on each step. These activation patterns encode information about relevant past events so that current and future decisions can be influenced, much as the stack would influence symbolic processing. Since there is evidence that isolating the linearly separable relationship from the non-linear part makes for more effective training (Lee & Holt, 1992), and since parsing seems to require approximately linear processing for most of the steps, we adapted the recurrent network designed by Elman to include direct connections from input to output units.

With this design, we are moving closer to a confrontation with the determinism hypothesis: Can a parser extract the structure of English sentences left-to-right deterministically as the hypothesis claims? The real evidence is in the complexity of sentences we have processed, in the extent of coverage of the structures of English, and in the degree of success achieved.

## TRAINING A RECURRENT NETWORK FROM SENTENCE TRACES

In training a recurrent network, as in training a feedforward network, patterns must be obtained that illustrate, by example, the mapping to be learned. The primary difference comes in sequencing. Patterns must be arranged in the sequence in which they occur during processing. This is necessary to permit proper context patterns to be computed and derived as a by-product of training.

We begin by selecting a group of sentences to be used for training. Each sentence is processed symbolically from left to right using a lexicon to determine the appropriate pattern for each word. For the particular set of rules we are using, only two buffer positions are required. The buffer is encoded using 26 units for the first position and 12 units for the second. These units represent features such as part of

speech (noun, verb, auxiliary, etc.), verb form (infinitive, gerund, past participle, etc.), noun phrase, verb phrase, final punctuation, and others. If the first two words of the sentence were 'the cat', for example, two units in the representation of the first buffer position would be turned on to indicate that the first word is a determiner and that it is a symbol that can begin a noun phrase. The second buffer pattern would indicate that the second word is a singular noun. These patterns are provided by a small lexicon. The pattern representing the top of the stack would be set to empty. The parser selects the rule which is applicable when there is a determiner in the first buffer position, a singular noun in the second position, and the stack is (in this example) empty. The action selected will be to create (push) a new sentence node on the stack. The patterns from the buffer (but not the stack) and the pattern representing the selected output are collected at each step. In this way, we end up with a sentence trace in the form of a sequence of coded condition-action patterns suitable for training. The buffer patterns provide the input stream and the actions chosen at each step are the desired output.

While the patterns could be derived by hand, this would not be expeditious. In practice we use a set of grammar rules designed for this purpose. They are applied automatically to assure that consistent rule application and coding practices are followed. The grammar is based on one used in PARSIFAL and consists of 76 single-action rules, each of which selects one of 39 possible actions. The actions include: attach as an adjective, attach as an auxiliary, attach as a main verb, create a noun phrase node, create a verb phrase node, create a secondary sentence node, drop (pop the stack, shifting the top of the stack into the first buffer position), and stop. For the examples presented in this paper, 30 hidden units are required, which are copied back at each recurrent step.

Training is time consuming, but possible. Use of singular value decomposition (SVD) to transform the input patterns into a space in which the input units are aligned orthogonally is a primary reason for success in training and seems to be particularly important for sizable networks with large numbers of training patterns (see Kalman et al., 1993, for a complete discussion of this technique and results obtained). This method leads to quicker and more effective training, and permits the resulting weights to be back-transformed so that they perform identically on the original training patterns. SVD allows easy identification of those inputs that contribute weakly or not at all to the problem. For the parser, the number of input units was reduced from 38 to 34, effectively yielding an architecture of 64-30-39 for training and 68-30-39 for testing. Training itself is performed using a modified conjugate-gradient method (Kalman, 1990).

## RESULTS

For training purposes, we constructed a set of 109 sentences that collectively exercised 32 of the 39 possible actions and covered a variety of syntactic forms, concentrating on basic sentence structure and relative clauses. A representative sample is shown in the first column of Figure 2. The sentences were presented to the network in four groups. The first contained 67 simple sentences, such as sentences A1-A4 in the figure. Groups two, three, and four contained sentences with embedded relative

A1 The girl likes Mary.	B1 John likes her.
A2 Have they disappeared?	B2 Does Mary like the cat?
A3 The angry old man would like the little kitten for the barn.	B3 The tall fat man scheduled a meeting for Mary.
A4 Schedule the meeting!	B4 Punish him!
A5 They like the teacher who fell.	B5 The little girl likes the cat which she bought.
A6 She should schedule the meeting which the teacher wants.	B6 Should the girl have bought the red horse which the man wanted?
A7 The books which the teacher bought were read by the boys.	B7 Were the books which he had bought read by the tall teacher?
A8 Has the girl who was punished by the teacher disappeared?	B8 Has the little girl who was punished read the book?
A9 Mary has been punished by the teacher who scheduled the exam for Friday.	B9 The cat has been taken by the tall man who likes Mary.
A10 Should she have been taken to the meeting which the teacher scheduled?	B10 Should the girl have been taken to the house which fell?
	B11 The man who met John bought the cat which the girl liked for the teacher who disappeared.
	B12 John scheduled a meeting for the boy who would take the exam which the teacher scheduled.

FIGURE 2. Sample training (A) and testing (B) sentences.

clauses in object noun phrases (sentences A5-A6), subject noun phrases (sentences A7-A8), and prepositional phrase noun phrases (sentences A9-A10).

The recurrent network learned these sentence-processing patterns perfectly despite the fact that the average number of processing steps is 30 with the most complicated sentences requiring over 50 steps. This means that the recurrent network learned the proper dynamic trajectories to capture long-range dependencies — for example, that every “create” action requires a matching “drop” action.

As discussed above, our earlier work demonstrated how a parser built from a similar, but non-recurrent network could generalize well enough to overcome problems of lexical ambiguity and ill-formedness. In this work, we have concentrated on examining a form of generalization in the sequencing of rule application. That is, the primary difference between the sentences in the training set and those in the testing set is the sequence with which rules are applied. To demonstrate generalization in the recurrent network, new sequences must be allowed that are similar to those learned but not precisely the same. We observed this type of generalization in our testing.

The network was tested with 109 sentences which use the same subset of rules as the training sentences, but in different sequence. Examples of these are shown in the second column of Figure 2. Of the test sentences, 82 were quite similar to those trained, but with elements exchanged, expanded, added, or eliminated (sentences B1-B10). Testing with this group resulted in 1636 recognizable states that had not occurred in the training set. Of these new patterns, only 42 (2.6%) were predicted incorrectly. The remaining 27 test sentences required the trained network to make more of a leap. One group (see sentence B11) contained structures that were present in the training set, but each of these testing sentences had two or more embeddings within it, whereas all of the training sentences had one or none. The last group contained embedded sentences within embedded sentences (Figure 2, B12), which the network had not seen. Some of the sentences in these two groups require

80 or more processing steps, fully 30 more than any of the training sentences. Testing with the full set of 109 sentences resulted in 2834 unique states that had not occurred in the training set, with 92 patterns (3.2%) predicted incorrectly.

## **SUMMARY, CONCLUSIONS, AND FUTURE WORK**

We have demonstrated that extracting syntactic structure from relatively complex English sentences can be performed using recurrent networks. This evidence, coupled with our previous work, is leading us toward a complete and unbiased evaluation of the determinism hypothesis.

Our next step is to examine our grammar rules and to determine if a similar parser can be constructed which processes strictly left-to-right. The primary stack-popping mechanism in a deterministic grammar, drop, uses the buffer to hold a piece of the structure for further processing. This structure almost always gets attached at the next step to the constituent promoted to the top of the stack by the drop. However, more complicated sequences are also possible, such as drop/drop/switch/attach or drop/create/attach. In all these cases, processing moves back a step rather than progressing relentlessly forward. Mechanisms for bundling primitive actions in order to eliminate backward steps are under investigation.

## **REFERENCES**

- Das, S., Giles, C.L., & Sun, G.Z. (1992). Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. *Proc 14th Ann Conf Cog Sci Soc*. Bloomington: Indiana University.
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14, 179-212.
- Kalman, B. L. (1990). Super linear learning in back propagation neural nets. Tech Rep WUCS-90-21, Dept of Comp Sci, Washington University, St. Louis, MO.
- Kalman, B. L., Kwasny, S. C., & Abella, A. (1993). Decomposing input patterns to facilitate training. *Proc World Congress on Neural Networks*, V.III, 503-506.
- Kwasny, S. C., & Faisal, K. A. (1990). Connectionism and determinism in a syntactic parser. *Connection Science*, 2, 63-82.
- Kwasny, S. C., & Faisal, K. A. (1992). Symbolic parsing via sub-symbolic rules. In J. Dinsmore (Ed.), *Closing the gap: Symbolism vs. connectionism*. Hillsdale, NJ: LEA.
- Kwasny, S. C., & Kalman, B. L. (1991). The case of the unknown word: Imposing syntactic constraints on words missing from the lexicon. *Proc 3rd Midwest Art Intel & Cog Sci Soc Conf*.
- Lee, S. E., & Holt, B. R. (1992). Regression analysis of spectroscopic process data using a combined architecture of linear and nonlinear neural networks. *Proc Intl Joint Conf on Neural Networks V.IV*, Piscataway, NJ: IEEE.
- Marcus, M. P. (1980). *A theory of syntactic recognition for natural language*. Cambridge, MA: MIT Press.
- Pinker, S., & Prince, A. (1988). On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. In S. Pinker & J. Mehler (Eds), *Connections and symbols*. Cambridge, MA: MIT Press.