LETTER Communicated by Risto Miikkulainen

Analyzing Holistic Parsers: Implications for Robust Parsing and Systematicity

Edward Kei Shiu Ho
Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

Lai Wan Chan
Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, New Territories, Hong Kong

Holistic parsers offer a viable alternative to traditional algorithmic parsers. They have good generalization performance and are robust inherently. In a holistic parser, parsing is achieved by mapping the connectionist representation of the input sentence to the connectionist representation of the target parse tree directly. Little prior knowledge of the underlying parsing mechanism thus needs to be assumed. However, it also makes holistic parsing difficult to understand. In this article, an analysis is presented for studying the operations of the confluent preorder parser (CPP). In the analysis, the CPP is viewed as a dynamical system, and holistic parsing is perceived as a sequence of state transitions through its state-space. The seemingly one-shot parsing mechanism can thus be elucidated as a step-by-step inference process, with the intermediate parsing decisions being reflected by the states visited during parsing.

The study serves two purposes. First, it improves our understanding of how grammatical errors are corrected by the CPP. The occurrence of an error in a sentence will cause the CPP to deviate from the normal track that is followed when the original sentence is parsed. But as the remaining terminals are read, the two trajectories will gradually converge until finally the correct parse tree is produced. Second, it reveals that having systematic parse tree representations alone cannot guarantee good generalization performance in holistic parsing. More important, they need to be distributed in certain useful locations of the representational space. Sentences with similar trailing terminals should have their corresponding parse tree representations mapped to nearby locations in the representational space. The study provides concrete evidence that encoding the linearized parse trees as obtained via preorder traversal can satisfy such a requirement.

Neural Computation **13, 1137–1170** (2001)

1 Introduction

Traditionally, symbolic approaches have dominated the study of language parsing. Various models have been devised, such as the chart parser and the augmented transition network (Allen, 1995; Gazdar & Mellish, 1989). Generally these models were characterized by their algorithmic and rule-based nature. The parsing process was governed by a well-defined algorithm in which the input sentence was analyzed step by step, with the target parse tree being built incrementally at the same time. Besides, symbolic representations were adopted for sentences and parse trees. Also, the construction of the parser required a detailed knowledge of the underlying grammar. Although remarkable success has been achieved in computer language processing, their practicality in natural language processing has been relatively limited. The major reason is that they have poor error recovery capability when parsing ungrammatical sentences. Their rule-based nature is too rigid to provide enough robustness as demanded by natural languages (Kwasny & Faisal, 1992). Besides, the extreme flexibility of natural languages effectively prohibits the complete enumeration of the grammar rules, which are essential for constructing an algorithmic parser.

In view of this, researchers have turned their attention to more data-oriented paradigms. Generally a model would be employed that acquired the target language by exposure to samples generated by it (commonly called positive data).[1] Among various paradigms, the statistical approach (Charniak, 1993) has already aroused the interest of many researchers. On the other hand, as inspired by the study of cognitive science, neural networks have been applied in language parsing also. One of the earliest models was due to Fanty (1985), who proposed a procedure that could construct a connectionist parser from any context-free grammar given (so no learning was involved). The parser employed localist representation techniques and was purely syntactic in nature. A similar "nonlearning" model has also been devised by Selman (1985), who made use of a Boltzmann machine to implement the parser.

Later, Hanson and Kegl (1987) put forward a connectionist model called the PARSNIP which was actually an autoassociative feedforward network (Rumelhart, Hinton, & Williams, 1986). To process a sentence, the syntactic tags corresponding to the words in the sentence were presented to the input layer in parallel. In contrast to Fanty's and Selman's models, distributed representations rather than localist ones were adopted. Processing succeeded if the network could "reproduce" at its output layer the syntactic tag corresponding to each position of the input sentence (so it was more a language recognizer than a parser).

[1] In general, negative data (counterexamples not generated by the language) would also be needed in order to avoid overgeneralization, although Angluin (1980) has proved that, theoretically, certain classes of languages were learnable by using positive data alone.

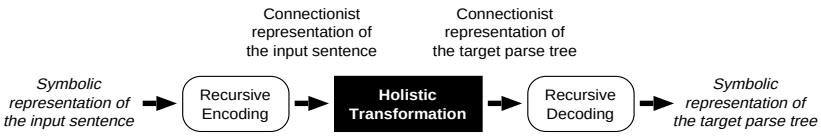

Figure 1: Holistic parsing paradigm (adapted from Ho & Chan, 1999).

A common drawback of these three models was that they all imposed a limit on the sentence's length. To circumvent this, Santos (1989) proposed the parsing and learning system (PALS), in which a matrix of units was used to represent the structure of the target parse tree. In each step, only a partial parse tree, called a snapshot, was stored in the matrix, which was combined with the next word of the sentence to grow into a larger snapshot. The complete parse tree was thus built up incrementally. This allowed the parser to handle sentences of unbounded length. The construction of the parse tree was guided by a set of language rules generated from the grammar rules. The applicability of each rule was determined by a rule weight that was adapted through learning.

Besides these early approaches, effort has been paid to integrate connectionist techniques into the symbolic parsing framework, leading to various hybrid parsers. For example, the connectionist deterministic parser (Kwasny & Faisal, 1992) consisted of a symbolic component, which is a deterministic parser, and a subsymbolic component, which is a feedforward network. During parsing, based on the contents of the parser's symbolic data structures, the feedforward network output the action to perform, and the parser manipulated its data structures accordingly. Other examples include the sentence gestalt model (St. John & McClelland, 1990), the PARSEC model (Jain, 1991), the neural network push-down automaton (NNPDA) (Sun, Giles, Chen, & Lee, 1993), and the subsymbolic parser for embedded clauses (SPEC) (Miikkulainen, 1996). However, these attempts still required a detailed specification of the parsing algorithm and the grammar. They were still algorithmic and rule based, although the rules might have more flexible decision boundaries. The inductive learning power of neural networks had not been fully utilized, and error recovery performance was unsatisfactory.

2 Holistic Parsing

Recently an alternative approach, holistic parsing, has been studied (see Figure 1). Several models have been proposed, including Reilly's parser (Reilly, 1992), the XERIC parser (Berg, 1992), the modular connectionist parser (Sharkey & Sharkey, 1992), the backpropagation parsing network

(BPN) (Ho & Chan, 1994), and the confluent preorder parser (CPP) (Ho & Chan, 1997).

In a holistic parser, connectionist representations are developed for sentences and parse trees. In contrast to the algorithmic approaches, parsing is achieved by mapping holistically the representation of the input sentence to the representation of the target parse tree. Ho and Chan (1999) proposed a general framework for holistic parser design. Several design dimensions have been identified:

> *Encoding sentences.* Conventionally, sentences are encoded as sequences by training a recurrent network. Two common choices are the simple recurrent network (SRN) (Elman, 1990) and the sequential recursive autoassociative memory (SRAAM) (Pollack, 1990).
>
> *Encoding parse trees.* Traditionally, parse trees have been represented as hierarchical data structures, and the recursive autoassociative memory (RAAM) (Pollack, 1990) has commonly been applied for this task. Recently, Kwasny and Kalman (1995) proposed linearizing parse trees by preorder traversal. The sequences obtained can then be encoded by a recurrent network (such as the SRN or the SRAAM).
>
> *Implementing the holistic transformation.* A feedforward network can be trained to effect an explicit transformation from the representation of the input sentence to the representation of the target parse tree. Alternatively, one may choose to encode the sentences or the parse trees first. The representations obtained can then be used as the training targets for encoding the others. Taking one step further, the representation of a sentence and that of its parse tree can co-evolve simultaneously instead of being developed independently. This technique is known as confluent inference (Chrisman, 1991).
>
> *Learning to parse phrases.* A holistic parser can be trained to parse phrases in addition to complete sentences. For example, parsing the verb phrase ⟨ V D N ⟩ gives the subtree (V (D N)).

As depicted in Figure 1, a holistic parser effectively encapsulates its underlying parsing mechanism in a "black box." Instead of defining how to achieve, we simply specify what to achieve. It has the advantage that the necessary grammatical knowledge can be inferred by inductive learning. But it is also difficult to understand how parsing is actually accomplished.

In this article, a model is presented for analyzing holistic parsers. The CPP is used as an example to illustrate the method.[2] In the model, the CPP is viewed as a dynamical system, whose state-space is defined as the set of all distributed representations that can possibly be developed over

[2] Experimental results (Ho & Chan, 1999) reveal that among all the holistic parsers proposed, the CPP has the best generalization performance and is also the most robust one.

the hidden layer of the network. Through training, a number of states are defined in the state-space where each state corresponds to a set of hidden-layer representations that produce the same result upon decoding. Some states are designated as final states, which, when being decoded, give a syntactically well-formed parse tree.[3] Effectively, a finite-state automaton is extracted. When parsing a sentence, the CPP moves from one state to another as each terminal of the sentence is being read. A trajectory is thus formed in the state-space, and parsing succeeds if the state that the trajectory terminates at is a final state.

The motivation of this study is threefold. First, the analysis allows the seemingly one-shot parsing mechanism of holistic parsers to be analyzed as a step-by-step inference process, with the intermediate parsing decisions being revealed by the states visited during parsing. This can enhance our understanding of how parsing is actually accomplished by a holistic parser.

Second, we find that in parsing an erroneous sentence, the trajectory of the CPP in the state-space will deviate from the normal track that is followed when the original intact sentence is parsed. However, each state of the CPP actually behaves like an attractor and encloses a set of points in the state-space that give the same result upon decoding. These points constitute the basin of attraction of the state. So if the parser's trajectory is deflected only slightly by the error in the sentence, it may still reach the basin of attraction of the target final state. In that case, the erroneous sentence can be recovered. This characteristic equips the CPP with a certain tolerance to noise.

Finally, the analysis reveals that the specific encoding method adopted by a holistic parser for representing parse trees has a profound effect on the distribution of the states in the state-space, which is a determining factor of the generalization capability of the parser. With respect to this issue, the simulation result provides concrete evidence that linearizing parse trees (as in the CPP) can improve the generalization performance of a holistic parser.

3 Confluent Preorder Parser (CPP)

Because we will use the CPP to illustrate the method for analyzing holistic parsers, this section first gives a brief review of it.

3.1 Training Methodology. The CPP combines two SRAAMs to form a dual-ported SRAAM (Chrisman, 1991), with one single hidden layer shared between them (see Figure 2). Two techniques are applied. First, each parse tree is linearized by preorder traversal to give a sequence (see also Kwasny & Kalman, 1995). For example, linearizing the parse tree in Figure 3 leads to the preorder traversal sequence ⟨ s np D N vp V np np D N pp P np D N ⟩.

[3] A parse tree or subtree is syntactically well formed if it can be derived step by step from a nonterminal by using the production rules of the context-free grammar given.

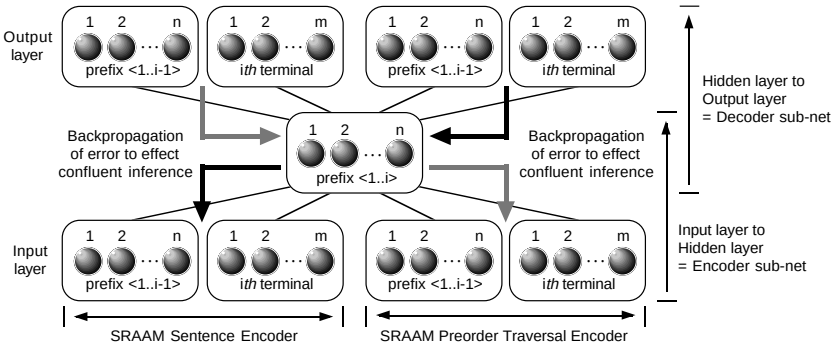

Figure 2: Confluent preorder parser (CPP).

The training sentences and their corresponding preorder traversals are then encoded by the sentence encoder and the preorder traversal encoder, respectively.

To encode a sentence, we input its terminals to the sentence encoder one at a time as the ith terminal field. In each time step i, the hidden-layer activation at time $i-1$ will serve as an additional input of the network as the prefix $\langle 1 \ldots i-1 \rangle$ field. The SRAAM then learns to reproduce the inputs at its output layer, and the resulting hidden-layer activation will be used as one of the inputs in the next time step. This process is repeated recursively until the whole sentence is read, and the final hidden-layer activation is used as the sentence coding. Preorder traversals are encoded by the preorder traversal encoder similarly.

Instead of training the two SRAAMs independently, confluent inference is applied. For each sentence, two extra training patterns are introduced, which involve adapting the weights of both SRAAMs. For example, given the sentence $\langle$ D N V D N $\rangle$, which corresponds to the preorder traversal

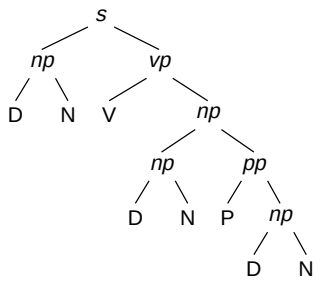

Figure 3: Parse tree of the sentence $\langle$ D N V D N P D N $\rangle$ (adapted from Ho & Chan, 1999).

Table 1: Training Patterns for Achieving Confluent Inference When Teaching the CPP to Parse ⟨ D N V D N ⟩.

Input Layer		Hidden Layer	Output Layer (target)	
Sentence Encoder	Preorder Traversal Encoder		Sentence Encoder	Preorder Traversal Encoder
⟨ D N V D ⟩ + N	Don't care	⟨ D N V D N ⟩	Don't care	⟨ s np D N vp V np D ⟩ + N
Don't care	⟨ s np D N vp V np D ⟩ + N	⟨ s np D N vp V np D N ⟩	⟨ D N V D ⟩ + N	Don't care

Note: Fields marked "don't care" signify that the weights connecting the units in these fields are fixed when learning the pattern concerned.

⟨ s np D N vp V np D N ⟩, the two patterns in Table 1 have to be included. In the first pattern, error is backpropagated from the output layer of the preorder traversal encoder to the input layer of the sentence encoder; in the second one, error is backpropagated from the output layer of the sentence encoder to the input layer of the preorder traversal encoder. These patterns place extra constraints on the training of the two SRAAMs such that when training succeeds, the sentence and its corresponding preorder traversal are represented by the same coding.

Besides complete sentences, the CPP learns to parse phrases to produce the corresponding subtrees. Experimental results (Ho & Chan, 1999) suggest that this can lead to better performance for a holistic parser in general.

3.2 Experiments. We use the context-free grammar in Table 2 (adopted from Pollack, 1990) to evaluate the performance of the CPP. In all, 112 sentences and their corresponding parse trees are generated. Among them, 80 sentences are randomly selected for training, and the remaining 32 sentences are reserved for testing. Three runs are performed (each uses different initial weights), and the average performance is reported.

Generalization performance is measured by the percentage of testing sentences that can be correctly parsed by the trained network. In addition to generalization performance, we evaluate its robustness. Four types of errors are considered: erroneous sentences with one terminal substituted by a wrong terminal (SUB), with an extra terminal inserted (INS), with one terminal omitted (OMI), and with two neighboring terminals exchanged (EX). Multiple erroneous sentences are generated systematically by injecting error into every possible position of each training sentence. There are 853 SUB sentences, 853 OMI sentences, 933 INS sentences, and 773 EX sentences. Robustness is defined as the percentage of erroneous sentences of

Table 2: Context-Free Grammar Used in the Experiment (Pollack, 1990).

Sentence	*Noun Phrase*	*Verb Phrase*	*Prepositional Phrase*	*Adjectival Phrase*
s → np vp	np → D ap	vp → V np	pp → P np	ap → A ap
s → np V	np → D N	vp → V pp		ap → A N
	np → np pp			

Table 3: Results of the Experiment.

Generalization	Error Recovery				
	SUB	OMI	INS	EX	Average
91.67%	94.72%	70.93%	50.80%	66.88%	70.46%

each type that can be successfully recovered. The results are summarized in Table 3.

4 Analyzing Holistic Parsers

In a holistic parser, sentences are encoded as sequences by using a recurrent network. During parsing, the representation of the input sentence is mapped to another piece of representation directly, which, upon decoding, gives the target parse tree. Apparently no intermediate decisions are involved.

However, during the recursive encoding of the input sentence, the hidden-layer activation of the recurrent network is evolving as each terminal of the sentence is being read. In some sense, it manifests the parser's temporary interpretation of the partial input sentence that it has seen so far. In each time step, the previous hidden-layer activation (much like a preliminary decision) is fed back to the input layer. This feedback is then composed with the current input terminal of the sentence (which provides some new information) to form the next hidden-layer activation (which represents an updated interpretation of the input sentence). Originally, only after the whole sentence is read will the final hidden-layer activation be decoded. But it is also worth examining the intermediate activation patterns so as to investigate the rational inference or knowledge resolution mechanism underlying the holistic parsing process.

4.1 A Dynamical Systems Model of Holistic Parsing. It is well known that neural networks with feedback links can be perceived as dynamical systems (Kolen, 1994; Rodriguez, Wiles, & Elman, 1999). From the network dynamics' point of view, the CPP is no different from an ordinary recurrent network. Hence, it can also be characterized as a dynamical system, and we define its state-space as the set of all distributed representations that can possibly be developed over the hidden layer of the network. From this, we apply the trained CPP to parse the 80 training sentences again. Besides the representations of the complete sentences, we collect the intermediate hidden-layer activation of the CPP in response to each terminal read. Each possible hidden-layer activation pattern corresponds to a point in the state-space of the dynamical system. Intuitively, it represents the parser's configuration during the parsing process. In parsing a sentence, the hidden-layer activation is changing as each terminal of the sentence is being read.

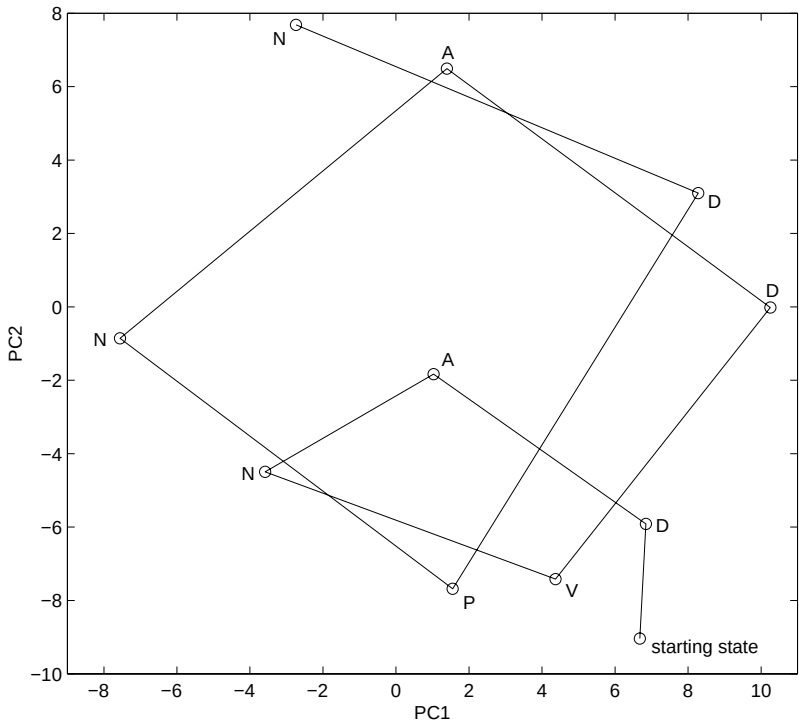

Figure 4: Trajectory of the CPP in the PC1-PC2 subspace when it parses the sentence ⟨ D A N V D A N P D N ⟩.

Correspondingly, the CPP can be thought of as moving from one point to another in the state-space. A trajectory is thus formed that traces the parsing process (see also Elman, 1990).

Unfortunately, the high dimension of the hidden layer makes it impossible to visualize the state-space directly. Principal component analysis is thus applied, and the state-space is projected onto the two-dimensional subspace spanned by the first two principal components: PC1 and PC2. For example, Figure 4 shows the trajectory of the CPP in the PC1-PC2 subspace when it parses the training sentence ⟨ D A N V D A N P D N ⟩.

4.2 Extracting a Finite-State Automaton from the CPP. The trajectories can in fact provide useful hints about the underlying parsing mechanism of the CPP. To reveal this information, we decode the points that the CPP visits when it parses the 80 sentences (these points correspond to hidden-layer activations). For each point, a sequence of terminals results that may represent the preorder traversal of a syntactically well-formed parse tree

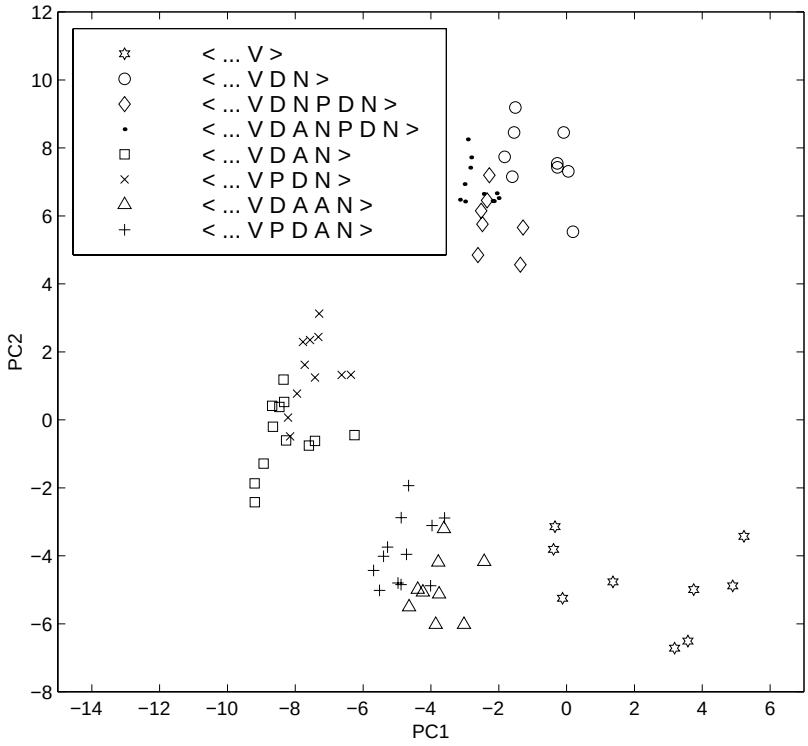

Figure 5: Distribution of the 80 final states identified for the CPP (that correspond to the 80 training sentences) in the PC1-PC2 subspace.

or subtree. In some sense, it is an interpretation of the configuration of the parser when it is situated at that point. For every unique sequence found, a new state is given rise to, whose identity is defined by the sequence. If the identity of a state is a legitimate preorder traversal, the state is said to be a final state. In all, 186 distinct states are identified: 102 final states, 83 intermediate states, and 1 starting state. Between them, there are 234 possible transitions.[4] Figure 5 shows the distribution of the 80 final states that correspond to the 80 training sentences in the PC1-PC2 subspace (note that the states are not identified by clustering the PCA or the representational space). Together, the states and the transitions define a finite-state automaton (Hopcroft & Ullman, 1979), which encodes the grammatical knowledge for the parsing function.

[4] The total number of states identified does not vary a lot in different runs of the experiments (within a range of ± 10). The same applies to the number of final states also.

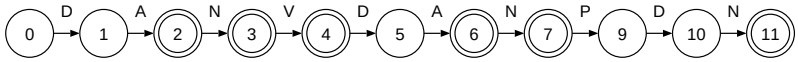

Figure 6: State transition sequence resulted when the CPP parses the sentence ⟨ D A N V D A N P D N ⟩ (see Figure 4 and Table 4).

Holistic parsing can thus be redefined. Before parsing begins, the CPP is in the starting state. It then changes from one state to another as each terminal of the input sentence is being read (we call this a state transition). If the last state that the CPP resides in is a final state, the sentence is successfully parsed, and the identity of the final state gives the parse tree required. Intuitively, holistic parsing of a sentence can also be analyzed as a step-by-step rational inference procedure, with the intermediate parsing decisions reflected by the identities of the states visited by the parser during the processing of the sentence.

For example, Figure 6 depicts the state transition sequence involved when the CPP parses the sentence ⟨ D A N V D A N P D N ⟩. Here, nodes with a double circle denote final states. For convenience, each state is labeled by a unique number. The identities of the states concerned are listed in Table 4.

Two observations merit further discussion. First, some of the state transition sequences resulted in parsing similar sentences (that share subsequences) have overlapped to a certain extent. Thus, some states are shared or reused. For example, when ⟨ D A N V D A N ⟩ and ⟨ D A N V D N P D N ⟩ are parsed, their respective prefixes ⟨ D A N V D A ⟩ and ⟨ D A N V D N ⟩ result in the same state transition sequence $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$. As a result of sharing states, the number of states of the finite-state automaton is effectively reduced. Recall that when the 80 training sentences are parsed, 186 states have been identified. But if they were represented as indepen-

Table 4: Identities of the States Appearing in Figure 6.

State	Identity
0	Starting state
1	···
2	(D N)
3	(D (A N))
4	((D (A N)) V)
5	···
6	((D (A N)) (V (D N)))
7	((D (A N)) (V (D (A N))))
9	((D (A N)) (V ···))
10	((D (A N)) (V ((D ···) ···)))
11	((D (A N)) (V ((D (A N)) (P (D N)))))

Note: ··· represents a sequence that cannot be interpreted to give a well-formed tree or subtree.

dent sequences, the number of states should have been 235.[5] Hence, it is a sign that the CPP is not simply memorizing the training examples. Intuitively, some kind of knowledge abstraction has occurred, in which the CPP spots out the similarities between the sentences and forms shared states accordingly to capture the commonalities.[6] For example, state 6 abstracts the common characteristics between the two similar prefixes ⟨ D A N V D A ⟩ and ⟨ D A N V D N ⟩.

In parsing the 80 training sentences, 42 shared states are found, and the maximum number of sequences encoded by a shared state is three (for example, state 49 represents the three subsequences ⟨ D N V D A ⟩, ⟨ D N V D N ⟩, and ⟨ D N V P D ⟩). Note that in addition to the similarities, the differences between the sentences are captured by the shared states as well. Consider ⟨ D A N V D A N ⟩ again. Upon decoding, state 6 gives the parse tree ((D (A N)) (V (D N))). After reading the last terminal, state 7 is reached whose identity is the parse tree ((D (A N)) (V (D (A N)))). Thus, the fact that ⟨ D A N V D A ⟩ has just been read is "carried" implicitly by state 6, although it is not manifested explicitly by its identity.

Note that the sharing of states may facilitate generalization also. When an unseen sentence similar to certain training sentences is parsed, some of the states that are discovered while parsing the training sentences would probably be revisited. Intuitively, these states identify the familiar fragments in the unseen sentence, based on which the CPP can work out the total parse tree. For example, state 50 originally represents the subsequence ⟨ D N V D A N ⟩ only. But after reading the prefix ⟨ D N V D A A ⟩ of the testing sentence ⟨ D N V D A A N ⟩, state 50 is visited again. In other words, state 50 becomes a shared state and encodes ⟨ D N V D A A ⟩ as well. When the last terminal N of the testing sentence is read, the CPP travels from state 50 to state 189 (a new state), which outputs the target parse tree. However, if the sentences were merely "hard-coded" as independent sequences, generalization would need to be done on the fly.

The second observation is that when we interpret the identities of the intermediate states, we find that some of them reveal traces of partial subtrees (such as state 9 and state 10 in Table 4). To a certain extent, they help to explain the underlying parsing mechanism of the CPP by illustrating how the target parse tree evolves during the parsing process. Besides, the observation serves as an evidence that the CPP has induced the internal structure of the sentences. Thus, grammatical knowledge has been acquired.

[5] In parsing the 80 training sentences, the average number of times each state is visited is 4.6 (excluding the starting state).

[6] In general, sentences sharing subsequences give rise to similar representations when being encoded by the CPP. If the overlaps between two sentences are substantial, their representations will be sufficiently close to each other in the representational space. Consequently, they are decoded to give the same state.

4.3 Explaining Generalization. Besides illustrating the holistic parsing mechanism, the dynamical systems model can be used to explain the generalization capability of the CPP. We claim that in addition to the final states corresponding to the training sentences, the training process will organize the state-space in such a way that extra final states are formed. Each extra final state, when being decoded, gives the parse tree of an unseen sentence. This enables the CPP to generalize to novel sentences.

Recall that in our experiment, 80 training sentences have been used. Each corresponds to a unique final state, giving a total of 80 final states. But it turns out that 22 extra final states have been identified. Among them, 9 correspond to some of the constituent phrases that have been included in training (e.g., the noun phrase ⟨ D N ⟩),[7] whereas the remaining 13 extra final states represent complete sentences that are not among the training sentences. In other words, they are novel sentences generalizable by the CPP (see Table 5).

Each of these 13 sentences is similar to the training sentences in two ways. First, it is either an exact prefix of a training sentence (e.g. state 28) or is sufficiently similar to the prefix of some training sentence (e.g., state 95). Second, for each of these generalizable sentences, there exists some sentence in the training set that has the same ending as it (e.g., the sentence ⟨ D N P D N V D N ⟩ as encoded by state 127 shares the same trailing terminals as the training sentence ⟨ D N P D A N V D N ⟩). Intuitively, each involves a novel composition of familiar constituents. That means that the constituent phrases used to construct them have also appeared somewhere in the training set (or, as for state 95, phrases that are sufficiently similar have been used before). But they have never been composed together in the same way as they are in the novel sentences.

We believe that both types of similarities facilitate the generalization of novel sentences. To illustrate this, consider state 99 in Table 5. Its corresponding sentence $S' = \langle$ D A A N P D N V $\rangle$ is a prefix of the training sentence ⟨ D A A N P D N V D A N ⟩. Hence, the preorder traversals of their respective parse trees—⟨ s np np D ap A ap A N pp P np D N V ⟩ and ⟨ s np np D ap A ap A N pp P np D N vp V np D ap A N ⟩—share the same prefix. However, their trailing parts are still quite different. This suggests that learning to parse a prefix of a complete sentence S (where the prefix itself is a sentence), given only the mapping between S and its corresponding parse tree, is by no means trivial.

This inadequacy is supplemented by the existence of certain training sentences whose endings are the same as or sufficiently similar to that of S' (one example is $S'' = \langle$ D N P D N V $\rangle$). The preorder traversals of these training sentences will have the same (or similar) trailing terminals as that

[7] These constituent phrases have been used to compose the training sentences (note that the CPP has learned to parse both complete sentences and phrases).

Table 5: The 13 Extra Final States Identified When Parsing the 80 Sentences in the Training Set.

State	Identity
4	((D (A N)) V) (a prefix of the training sentence ⟨ D A N V D A N ⟩)
99	(((D (A (A N))) (P (D N))) V) (a prefix of the training sentence ⟨ D A A N P D N V D A N ⟩)
110	(((D N) (P (D (A N)))) V) (a prefix of the training sentence ⟨ D N P D A N V D A A N ⟩)
28	((D (A (A N))) (V (D N))) (a prefix of the training sentence ⟨ D A A N V D N P D N ⟩)
72	(((D (A N)) (P (D N))) (V (D N))) (a prefix of the training sentence ⟨ D A N P D N V D N P D N ⟩)
89	(((D (A (A N))) (P (D (A N)))) (V (D N))) (a prefix of the training sentence ⟨ D A A N P D A N V D N P D N ⟩)
113	(((D N) (P (D (A N)))) (V (D (A N)))) (a prefix of the training sentence ⟨ D N P D A N V D A N P D N ⟩)
142	((((D (A N)) (P (D N))) (P (D (A N)))) (V (D (A N)))) (a prefix of the training sentence ⟨ D A N P D N P D A N V D A N P D N ⟩)
148	((((D (A N)) (P (D N))) (P (D N))) (V (D N))) (a prefix of the training sentence ⟨ D A N P D N P D N V D N P D N ⟩)
55	((D N) (V (P (D N)))) (similar to a prefix of ⟨ D N V P D A N ⟩)
95	(((D (A (A N))) (P (D (A (A N))))) V) (similar to a prefix of ⟨ D A A N P D A N V P D A N ⟩)
122	(((D N) (P (D (A N)))) (V (P (D N)))) (similar to a prefix of ⟨ D N P D A N V P D A N ⟩)
127	(((D N) (P (D N))) (V (D N))) (similar to a prefix of ⟨ D N P D N V D A N ⟩)

Notes: For state 4, the corresponding sentence ⟨ D A N V ⟩ is a prefix of several training sentences besides the one shown (a similar observation can be made for state 99 and state 110 also), whereas for each of the states 28, 72, 89, 113, 142, and 148, the respective sentence is a prefix of one training sentence only. For the remaining states, the sentences encoded are not the exact prefix of any training sentence

of S′ (e.g., the preorder traversal of S″ is ⟨ s np np D N pp P np D N V ⟩). Intuitively, they provide hints on how to parse a novel sentence that ends with similar terminals (such as S′).

More final states have been formed in fact. To discover them, we parse the 32 testing sentences again. Forty new states—17 final states and 23 intermediate states—are identified as a result. These 17 new final states correspond to 17 of the 29 testing sentences that can be successfully generalized by the CPP. The 12 final states that correspond to the remaining 12 testing sentences

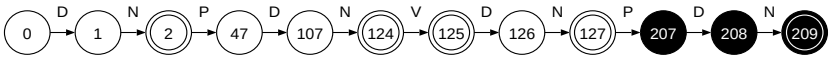

Figure 7: State transition sequence involved in parsing the testing sentence ⟨ D N P D N V D N P D N ⟩.

are among the 22 extra final states that have already been identified when parsing the 80 training sentences. For example, Figure 7 shows the state transition sequence in parsing the testing sentence ⟨ D N P D N V D N P D N ⟩, where darkened nodes represent states that are newly discovered. State 209 is one of those 17 new final states identified while parsing the 29 testing sentences. Its identity is the parse tree (((D N) (P (D N))) (V ((D N) (P (D N))))).

In Figure 8, the final states corresponding to the 29 testing sentences that can be successfully generalized by the CPP are mapped onto the PC1-PC2 subspace (the red symbols). Also shown are the final states that correspond to the 80 training sentences (the blue symbols). As depicted, the final states corresponding to the testing sentences are clustered around the final states corresponding to the training sentences.

As revealed by the discussion, the generalization performance of the CPP is intimately related to the number of final states that can be realized by training. Since the hidden units are real valued, an infinite number of states can be defined in principle. However, due to discretization in the encoding and decoding processes, only some of them can become final states in practice.

In general, the number of final states that can be realized is proportional to the number of training examples used. However, a recursive context-free grammar can potentially generate an infinite number of sentences. In that case, the CPP will have an infinite number of final states, with each representing a unique parse tree. Obviously it is impossible to achieve practically given only a limited number of training examples. Apparently, this places a theoretical limit on the CPP's computation power. What training should strive for is to organize the state-space in such a way that the maximum number of extra final states can be realized in addition to those defined by the training sentences. Doubtlessly, this at least requires careful selection of a training set that is representative of the possible types of structures and the possible modes of syntactic composition.

5 Explaining Error Recovery

The CPP is capable of parsing mildly ungrammatical sentences. In this section, we apply the dynamical systems model to investigate how the CPP corrects errors. Depending on the outcome of the recovery, there are three possible cases to consider.

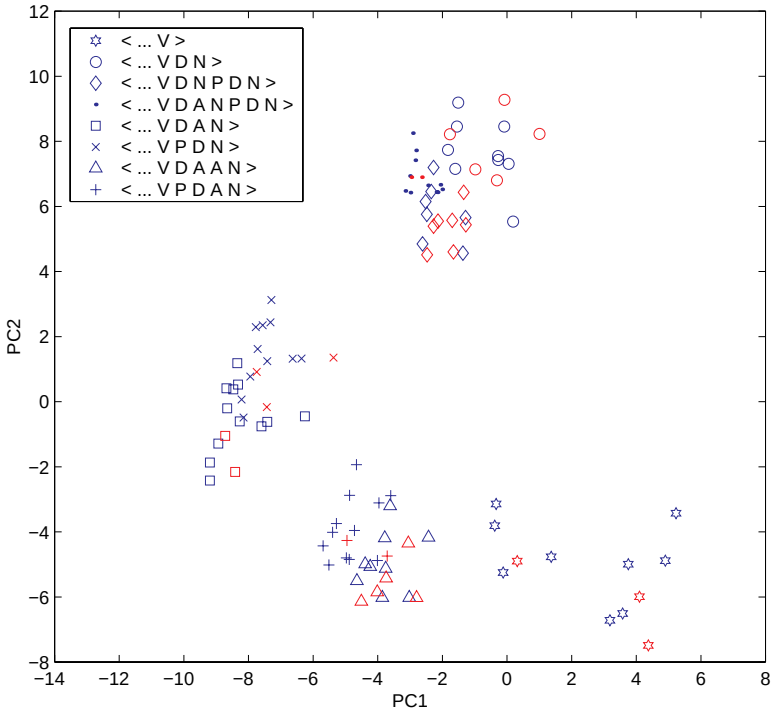

Figure 8: Distribution of the final states of the CPP in the PC1-PC2 subspace. The red symbols denote the 29 testing sentences that can be successfully generalized, and the blue symbols correspond to the 80 training sentences.

5.1 The Original Sentence is Recovered. Consider the erroneous sentence $E_1 = \langle$ D A N V P A N P D N $\rangle$, which is derived from the training sentence $S_1 = \langle$ D A N V D A N P D N $\rangle$ by substituting its fifth terminal D by a P. By examining the trajectory of the CPP in the PC1-PC2 subspace (see Figure 9), we see that when E_1 is parsed, the parser initially follows the same path as when S_1 is parsed. But as soon as the wrong terminal P is read (when the CPP is at point *t*), it deviates from the normal track and travels to *u* instead of *v*. Yet the discrepancy between the two paths decreases gradually thereafter as the remaining terminals in E_1 are processed. Finally, the CPP resides at *x*, whereas when S_1 is parsed, the trajectory terminates at *y*. But when *x* is decoded, the parse tree $T_1 =$ ((D (A N)) (V ((D (A N)) (P (D N))))) is obtained, which is exactly the parse tree of the original sentence S_1. E_1 is thus successfully recovered.

On the other hand, when two other erroneous sentences $E_2 = \langle$ D A N V D D N P D N $\rangle$ and $E_3 = \langle$ D A N V D A N P A N $\rangle$ are parsed (which are also

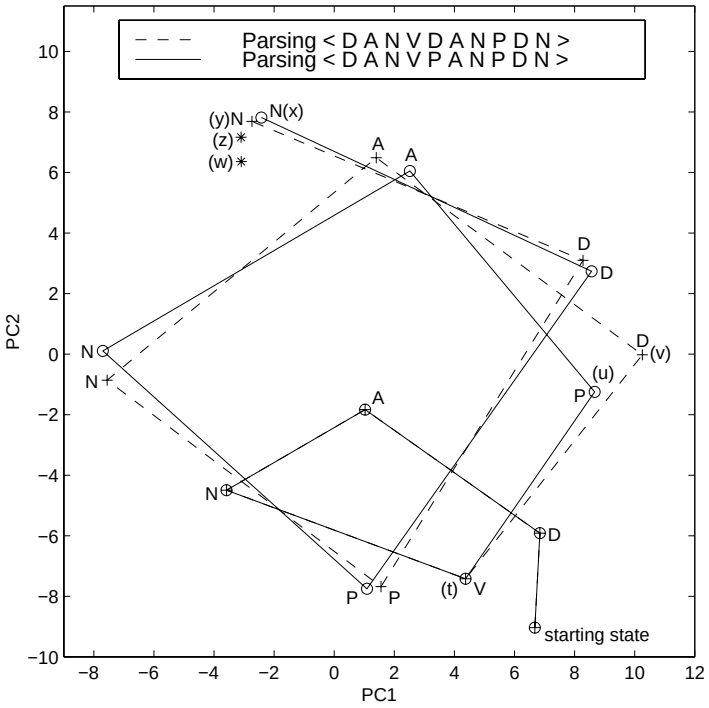

Figure 9: Trajectory of the CPP during the recovery of the erroneous sentence ⟨ D A N V P A N P D N ⟩ (solid line). The dashed line traces its trajectory in parsing the original correct sentence ⟨ D A N V D A N P D N ⟩. Points *w* and *z* mark, respectively, the final position of the parser when the erroneous sentences ⟨ D A N V D D N P D N ⟩ and ⟨ D A N V D A N P A N ⟩ are parsed. Upon decoding, all of *w*, *x*, *y*, and *z* give the same parse tree ((D (A N)) (V ((D (A N)) (P (D N))))).

derived from S_1 by substitution), the CPP terminates at *w* and *z* respectively (see Figure 9). Upon decoding, they give T_1 also.

These results suggest that the final state corresponding to T_1 is not a single point in the state-space. Instead, it occupies a region that at least encloses the points *w*, *x*, *y*, and *z*. Intuitively, the states are not discrete. Each of them behaves like an "attractor" and encloses a set of points in the state-space. The SRAAM representations (or hidden-layer activations) corresponding to the points give the same parse tree upon decoding. In some sense, these points constitute the basin of attraction of the state (note that the definitions of attractors and basins of attraction as adopted here are different from the conventional ones assumed in dynamical systems study). As *w*, *x*, *y*, and *z* are all lying within the basin of attraction of the same final

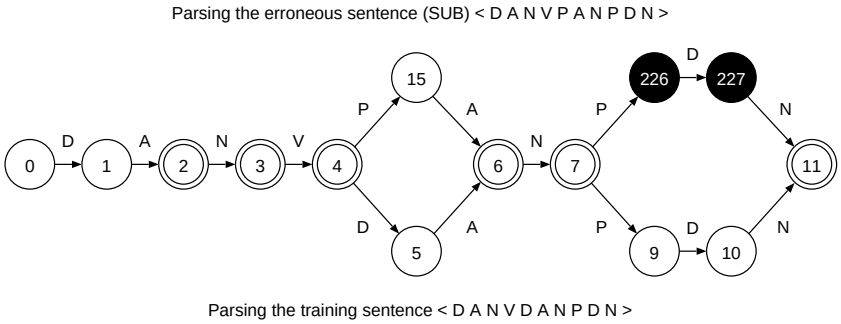

Figure 10: State transition sequence involved in parsing the erroneous sentence ⟨ D A N V P A N P D N ⟩. The identity of state 11 is the parse tree ((D (A N)) (V ((D (A N)) (P (D N))))).

state (state 11 in Figure 10), they are thus decoded to give the same parse tree T_1. This helps to explain how E_1, E_2, and E_3 are recovered. It is this attractor-like characteristic of states that equips the CPP with a certain tolerance to noise.

Taking one step further, we think that the basin of attraction of a final state, due to training, is larger than that of a nonfinal one. This can be revealed by the state transition sequence for parsing E_1 (see Figure 10). Upon reading the wrong terminal P (when the CPP is in state 4), the state transition sequence starts to deviate from its normal track (which is followed when S_1 is parsed) and the CPP moves to state 15. Surprisingly, the two sequences then converge again and overlap through the next two terminals. After that, they diverge for the second time. Finally, they terminate at the same state (state 11).

An explanation can be offered here. As shown in Table 6, the two states where the two sequences overlap (states 6 and 7) are both final states (they correspond to the training sentences ⟨ D A N V D N ⟩ and ⟨ D A N V D A N ⟩, respectively). It is thus reasonable to believe that their basins of attraction are comparatively larger, such that although the state transition sequence deviates from the original track, it can still pass through the basins of attraction of states 6 and 7, whereas states 9 and 10, being nonfinal, have smaller basins of attraction and the CPP just overshoots them, although it is in the vicinity of them (see Figure 9). Intuitively, states 6 and 7 are exerting attractive force on the CPP that is strong enough to pull it toward them. To a certain extent, this also helps to drift the trajectory of the CPP toward the basin of attraction of the final state 11 (thus, it has contributed to the recovery of E_1).

Another interesting observation is, as revealed by Figure 10, that the finite-state automaton extracted is nondeterministic. For example, when

Table 6: Identities of the States Appearing in Figure 10.

State	Identity	Final State
0	Starting state	
1	⟨ s ⟩	
2	⟨ np D N ⟩	√
3	⟨ np D ap A N ⟩	√
4	⟨ s np D ap A N V ⟩	√
5	⟨ P np D N vp ap A D N ⟩	
15	⟨ vp np D N vp ap A D N ⟩	
6	⟨ s np D ap A N vp V np D N ⟩	√
7	⟨ s np D ap A N vp V np D ap A N ⟩	√
9	⟨ s np D ap A N vp V np D D ap A N V ⟩	
226	⟨ s np D ap A N vp V pp P D ap A N V ⟩	
10	⟨ s np D ap A N vp V np np D ap A V np D N ⟩	
227	⟨ s np D ap A N vp V np np D ap A ap A D N ⟩	
11	⟨ s np D ap A N vp V np np D ap A N pp P np D N ⟩	√

the parser is in state 7, there are two different transitions for the same input symbol P. This reflects that the finite-state automaton is only a nonperfect approximation of the CPP, as the operation of the latter is deterministic.[8] But at the same time, it further supports our claim that the states are not discrete, and the parser is capable of making "context-sensitive" decisions during parsing: the exact edge taken during a state transition will depend on the preceding input symbols read. With reference to Figure 10, if ⟨ D A N V D A N ⟩ has just been read, the parser travels to state 9, whereas if ⟨ D A N V P A N ⟩ is read, it moves to state 226 instead. The parser thus behaves like a graded-state machine (Servan-Schreiber, Cleeremans, & McClelland, 1991), and apparently its states have some type of "memory."

5.2 A Grammatical Sentence Other Than the Original One is Recovered. Consider the erroneous sentence E_4 = ⟨ A A N P D N V ⟩, which is obtained by deleting the first terminal from the training sentence S_2 = ⟨ D A A N P D N V ⟩. As shown in Figure 11, the parser resides at *x* after reading the last terminal of E_4, whereas when S_2 is parsed, the trajectory terminates at *y*. Upon decoding, *x* gives the parse tree T_2 = (((D (A N)) (P (D N))) V), while *y* gives the parse tree T_3 = (((D (A (A N))) (P (D N))) V), which corresponds to the original sentence S_2. In Figure 11, another point *z* is also shown, which is the final position of the parser when the sentence S_3 = ⟨ D A N P D N V ⟩ is parsed. When being decoded, *z* gives the same parse tree T_2. By definition, *x* and *z* are in the same final state.

[8] Yet a nondeterministic finite-state automaton is equivalent to a deterministic one as far as expressive power is concerned. In fact, the transformation of a nondeterministic finite-state automaton to a deterministic one is well defined (Hopcroft & Ullman, 1979).

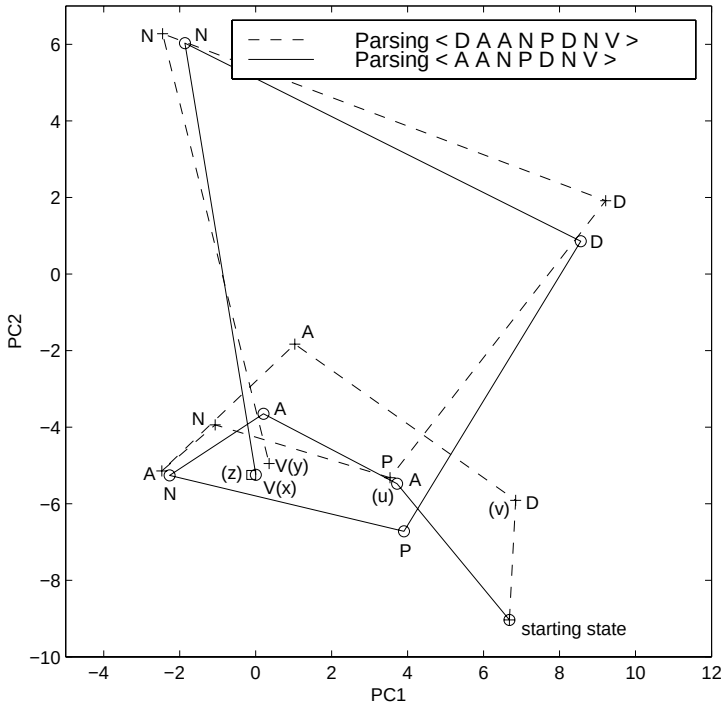

Figure 11: Trajectory of the CPP when parsing the erroneous sentence ⟨ A A N P D N V ⟩ (the solid line). The dashed line traces the trajectory when the original correct sentence ⟨ D A A N P D N V ⟩ is parsed. Point *z* is the final position of the parser when the sentence ⟨ D A N P D N V ⟩ is parsed. Upon decoding, both *x* and *z* give the parse tree (((D (A N)) (P (D N))) V), whereas *y* gives the parse tree (((D (A (A N))) (P (D N))) V).

Intuitively, the error in E_4 has deflected the trajectory of the CPP to such an extent that the point *x* where the trajectory terminates at is deviated sufficiently far from *y*. Effectively, *x* has "escaped" from the basin of attraction of the final state enclosing *y* (i.e., state 99 in Figure 12). Instead, it is "attracted" by another nearby final state enclosing *z* (state 70 in Figure 12). This explains why T_2 is produced instead of T_3.

5.3 A New Final State is Discovered. Consider the erroneous sentence E_5 = ⟨ D A A A N V D P A A N ⟩, which is produced from the training sentence S_4 = ⟨ D A A A N V D A A N ⟩ by inserting a superfluous terminal P after its seventh terminal. When E_5 is parsed, the trajectory terminates at *x*, whereas for S_4, it terminates at *y* (see Figure 13). As shown, *x* is sufficiently

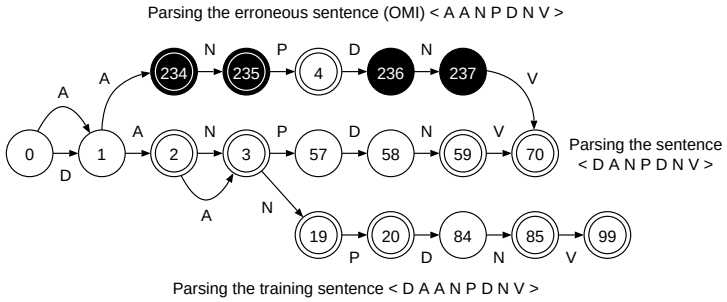

Figure 12: State transition sequence involved in parsing the erroneous sentence ⟨ A A N P D N V ⟩. The identities of states 70 and 99 are (((D (A N)) (P (D N))) V) and (((D (A (A N))) (P (D N))) V), respectively.

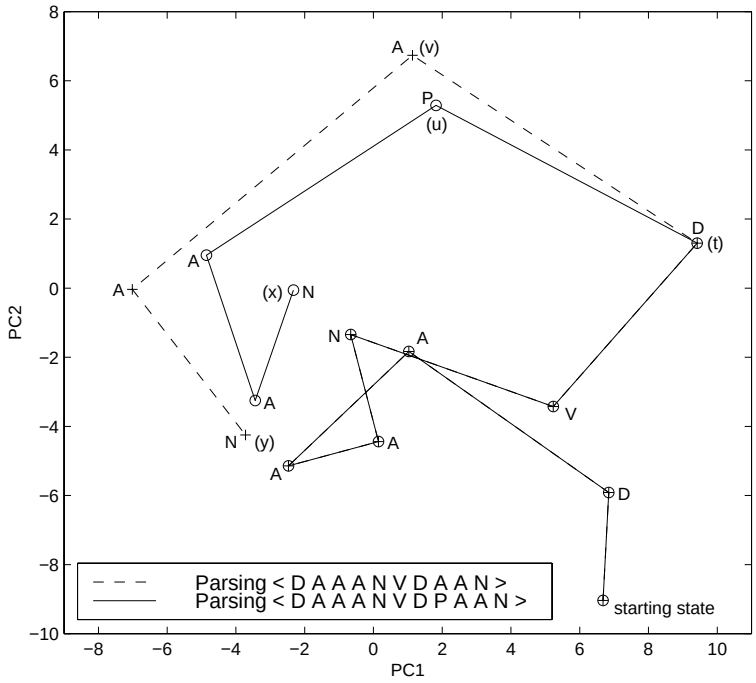

Figure 13: Trajectory of the CPP when parsing the erroneous sentence ⟨ D A A A N V D P A A N ⟩ (solid line). The dashed line shows the trajectory when the original correct sentence ⟨ D A A A N V D A A N ⟩ is parsed. At the beginning, the two trajectories overlap. But from point *t*, they diverge. Point *x* is situated in a new final state that corresponds to the parse tree ((D (A (A (A N)))) (V (D (A (A (A N)))))).

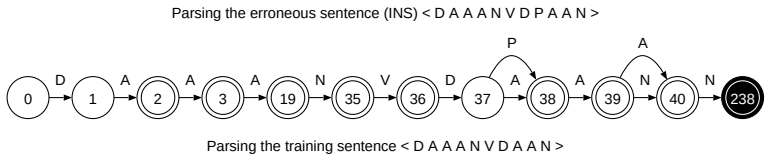

Figure 14: State transition sequence involved in parsing the erroneous sentence ⟨ D A A A N V D P A A N ⟩.

far away from the final state enclosing *y*. Upon decoding, *x* produces the parse tree $T_4 = ((D\,(A\,(A\,(A\,N))))\,(V\,(D\,(A\,(A\,(A\,N))))))$, which is not among the parse trees used for training (and it is not one of the parse trees used for testing, either). Thus, *x* has entered the basin of attraction of a new final state: state 238 in Figure 14 (whose identity is T_4). Moreover, this final state is "reachable" in the sense that when the unseen grammatical sentence ⟨ D A A A N V D A A A N ⟩ is parsed, T_4 is produced.

6 Implications for Systematicity in Representations

The performance of a holistic parser is affected by the number of final states that can be realized by training. Yet whether these final states can actually be reached is an equally important issue. Recall that the CPP will reside at a point in the state-space after reading the last terminal of the input sentence. If this point is lying within the final state corresponding to the target parse tree, parsing succeeds. Otherwise it fails, even if the final state does exist somewhere in the state-space. Hence, the performance of a holistic parser is also dependent on the distribution of its final states in the state-space. Since the final states are actually sentence or parse tree representations, the parser's performance is dependent on the distribution of these representations in the representational space (i.e., the state-space).

Regarding this issue, one would expect that sentences or parse trees having a similar structure should be represented by similar connectionist coding. Consequently, they should lie close together in the representational space. Fodor and Pylyshyn (1988) referred to this characteristic as the systematicity in representations and claimed that it is the key to good performance in connectionist artificial intelligence systems. However, we will demonstrate that having systematic representations alone is not a sufficient condition for good generalization performance in holistic parsing. Instead, the exact way that the parse tree representations are organized in the representational space is a determining factor of its generalization capability, which we will show is dependent on the encoding method used.

6.1 Systematicity Revisited. Fodor and Pylyshyn (1988) have posed a difficult challenge to connectionists. The systematicity property, they have

claimed, is a major element underlying human cognitive capacities. In their work, they have criticized that connectionist representations, lacking a constituent syntactic structure, cannot explain or even exhibit systematicity without creating connectionist implementations of classical symbolic systems. Their work has stimulated much discussion as well as controversy, and a number of approaches and systems have been proposed that claim to be able to meet the systematicity requirement (e.g., Chalmers, 1992; Niklasson & Van Gelder, 1994; Smolensky, 1990). To a certain extent, they have successfully demonstrated that connectionist systems are capable of performing structure-sensitive operations using distributed representations.

Along another line of research, much work has also been done on formalizing the systematicity concept. Among them, Hadley (1994) has defined four degrees of systematicity—weak, quasi-, strong, and semantic systematicity—for characterizing the generalization performance of a language learning system. Each degree reflected how novel the testing items were relative to the training data used. His model has been elaborated by Christiansen and Chater (1994), who proposed that the capability of a connectionist system to process novel inputs could be classified into three levels: weak, quasi-, and strong generalization. The authors have also provided some hints of how strong generalization could possibly be achieved by a connectionist system. On the other hand, a more comprehensive framework has been put forward by Niklasson and Van Gelder (1994), which classified different degrees of systematicity into six levels.

Strictly speaking, the CPP does not fit in any of these classification models, since lexical categories are used instead of words. Besides, each sentence contains a single clause only, and there are no embedded sentences. But the testing sentences used in our experiments are novel in the sense that each sentence involves a novel syntactical composition of familiar constituents. And regarding the systematicity requirement, the representations developed by the CPP do exhibit certain syntactical structure. With reference to Figure 5, training sentences that end with similar terminals have their corresponding representations lying close to one another in the PC1-PC2 subspace (this is because sentences are encoded from left to right by using an SRAAM, which emphasizes the most recent inputs to the network, that is, the trailing terminals). Roughly eight clusters can be identified, each consisting of sentences having the same verb phrase. Moreover, as depicted in Figure 8, each testing sentence is mapped to the cluster of sentences that have the same trailing terminals. These observations illustrate that the representational space of the CPP is syntactically organized, and systematicity in representations is thus exhibited (equivalently, we also have systematicity in final states).

6.2 The P_{RAAM} Parser. Having systematic representations alone is not a sufficient condition for good generalization performance of a holistic parser. To support our argument, we construct another holistic parser, the

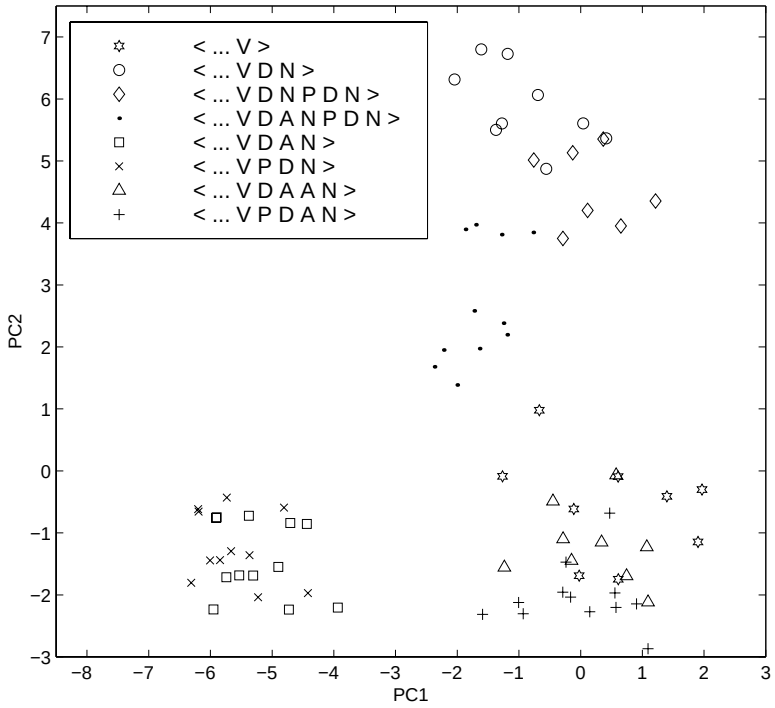

Figure 15: Distribution of the 80 final states (corresponding to the 80 training sentences) as identified by the P_{RAAM}.

P_{RAAM}, which is a derivative of the CPP (the symbol "P" in P_{RAAM} stands for "parser"). In the P_{RAAM}, parse trees are encoded as hierarchical data structures by training a RAAM. This is in contrast to the CPP, where parse trees are linearized into sequences, which are then encoded by using an SRAAM. But in other respects, they are the same (sentences are encoded by an SRAAM, and confluent inference is applied). The same sets of 80 training cases and 32 testing cases as adopted by the CPP are used to train and evaluate the P_{RAAM}, respectively. The generalization performance is found to be 53.13%, which is considerably worse than that of the CPP. Principal component analysis is then carried out on the representations produced. If we label each representation by the trailing part of the sentence encoded (the verb phrase), the representations do not form very clear-cut clusters in the PC1-PC2 subspace (see Figure 15). Apparently the representations produced by the P_{RAAM} are not systematic.

But actually this is not the case. Recall that in the P_{RAAM}, parse trees are encoded as hierarchical data structures by training a RAAM. As a result,

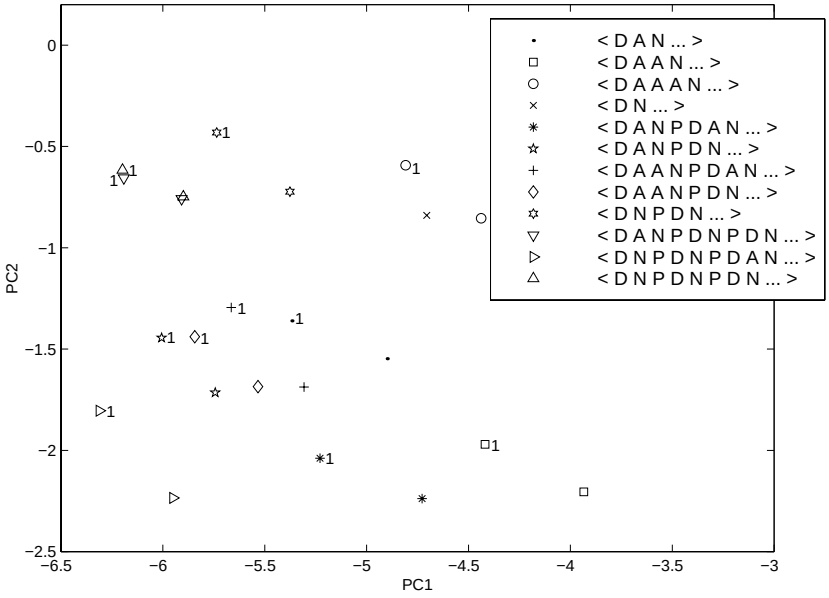

Figure 16: Distribution of the representations produced by the P_{RAAM} for sentences that end with the verb phrases ⟨ V D A N ⟩ or ⟨ V P D N ⟩ (symbols labeled by 1 correspond to sentences that end with ⟨ V P D N ⟩).

a sentence representation, being identical to the representation of its corresponding parse tree due to confluent inference, will be much affected by the order and manner in which the subtrees are composed together to form the total parse tree. Effectively, each representation takes the entire parse tree structure into account, with more weight on the higher levels, rather than being dictated solely by the last few terminals of the sentence encoded (as when parse trees are linearized). This explains why clear-cut clusters are not observed in Figure 15.

As an illustration, let us examine the representations of the sentences that end with the verb phrases ⟨ V D A N ⟩ or ⟨ V P D N ⟩. As shown in Figure 15, the two clusters of representations are tangled up with each other in the PC1-PC2 subspace. But if we label each representation by the subject noun phrase of the sentence instead, more clear-cut clusters can be observed. As depicted in Figure 16, sentences with the same subject noun phrase have their coding mapped to nearby locations in the PC1-PC2 subspace. Moreover, a certain degree of neighboring relationship is observed among the clusters. Precisely, two clusters that correspond to sentences having similar subject noun phrases are located close to each other in the representational

space (e.g., the cluster ⟨ D A A N P D N . . . ⟩ is situated in between the two clusters ⟨ D A N P D N . . . ⟩ and ⟨ D A A N P D A N . . . ⟩). The reason is that each parse tree representation is produced by composing the coding of the subject noun phrase and the verb phrase. Hence, both will affect the distribution of the representations.

To summarize, both parsers produce systematic representations, but their representations are organized along different dimensions of syntactical characteristics (the CPP emphasizes the trailing terminals of the sentence, while the P_{RAAM} takes the entire parse tree into account), which is caused by the use of different encoding methods for parse trees (parse trees are linearized in the CPP but not in the P_{RAAM}). This results in the discrepancy in their performances.

6.3 Linearizing Parse Trees Improves Generalization Performance. Consider a novel sentence S′. To parse S′, we first encode it by an SRAAM (recall that in both the CPP and the P_{RAAM}, input sentences are encoded by an SRAAM). Based on our prior discussion, the coding produced will be similar to the representations $R_{S_1}, R_{S_2}, \ldots, R_{S_n}$ of those training sentences $S_1, S_2, \ldots, S_n$ that end with the same terminals as S′. Consequently, the coding and, correspondingly, their final states will lie close together in the representational space.[9] Note that due to confluent inference, $R_{S_1}, R_{S_2}, \ldots, R_{S_n}$ are equal to the corresponding parse tree representations $R_{T_1}, R_{T_2}, \ldots, R_{T_n}$, respectively, where T_i is the parse tree of S_i ($i = 1, \ldots, n$).

$R_{S'}$ is then decoded to give a parse tree T. If T is to be equal to the correct parse tree T′ of S′, the representation R_T should be sufficiently similar to the correct parse tree representation $R_{T'}$. In other words, R_T ($= R_{S'}$) has to be situated near to $R_{T'}$ in the representational space; that is, they are in the same final state. As R_T lies close to R_{T_1}, R_{T_2}, ..., R_{T_n}, so does $R_{T'}$. This implies that the parse tree representations should be organized according to the trailing terminals of the corresponding sentences encoded. By encoding the linearized forms of parse trees as in the CPP, the representations obtained will be organized according to the trailing terminals of the corresponding sentences. But a similar effect is not achievable if parse trees are encoded hierarchically as in the P_{RAAM}. This explains why the generalization capability of the CPP is stronger than that of the P_{RAAM}. It is thus evident that linearizing parse trees can improve the generalization performance of a holistic parser.

As an illustration, let us examine the subset of testing sentences that end with the verb phrase ⟨ V D N ⟩ (see Figure 17). There are five such cases. Only one can be parsed by the P_{RAAM} (the failed cases are labeled 1, 2, 3,

[9] As shown in Figures 5 and 8, in the CPP, the training sentences form different clusters in the PC1-PC2 subspace according to their trailing terminals, and each testing sentence is mapped to the cluster of sentences that end with the same terminals.

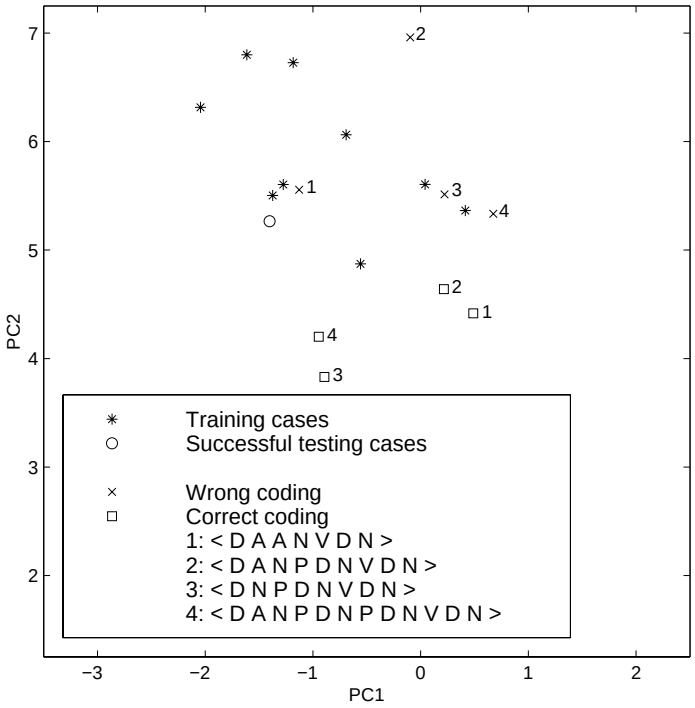

Figure 17: Distribution of the final states for those sentences that end with the verb phrase ⟨ V D N ⟩. There are five testing cases. The ○ symbol corresponds to the one that can be successfully parsed by the P_{RAAM}; the labels 1, 2, 3, and 4 denote the four failed cases. In each failed case, the sentence representation produced by the SRAAM is marked by ×, and the corresponding correct parse tree coding as produced by the RAAM is marked by □.

and 4, respectively). However, the parse trees in all five cases can actually be represented by the RAAM of the P_{RAAM} (in fact, the RAAM has perfect generalization with respect to all the 32 testing cases). So the problem is just that the SRAAM representation produced by encoding the input sentence is seriously deviated from the range of correct parse tree representations (i.e., the desired final state) that can be decoded by the RAAM to give the target parse tree. As shown, the sentence representations produced by the SRAAM of the P_{RAAM} in the failed cases (denoted by ×) are located near the cluster of representations of training sentences that end with the verb phrase ⟨ V D N ⟩. However, the corresponding parse tree coding produced by the RAAM of the P_{RAAM} (denoted by □) is remote from that cluster. Parsing thus fails.

As a final remark, although encoding parse trees hierarchically will lead to poor generalization in holistic parsing, the coding formed, being systematic, may still be useful for subsequent structure-sensitive operations. In fact, Chalmers (1992) has achieved a structure transformation task by using a feedforward network to map an active sentence to its passivized counterpart, where both sentences were encoded by a RAAM. What we want to reiterate is that having representations systematically placed in the representational space does not guarantee good generalization. Whether they are organized in a way that facilitates the task at hand is even more important. To conclude, there exists no single scheme that works for all tasks. The best choice is simply problem dependent.

6.4 Effect of Long-Term Dependencies on Generalization Performance. Readers may be concerned that as a result of linearizing parse trees, two different sentences that share the same ending will have similar representations. Consequently, their difference can easily be lost in a linearized structure (as in the CPP). This issue is commonly known as the long-term dependency problem (see, e.g., Lin, Horne, Tino, & Giles, 1996). In general, the problem is unavoidable for the CPP because it is a recurrent network operationally. But empirically, our experimental results show that its effect on the performance of the CPP is not too apparent. Consider the following testing sentences:

1. $\text{Test}_1 = \langle$ D N P D N P D N V D N P D N $\rangle$.
2. $\text{Test}_2 = \langle$ D N P D N P D N V D A N P D N $\rangle$.
3. $\text{Test}_3 = \langle$ D A N P D N P D A N V D N P D N $\rangle$.
4. $\text{Test}_4 = \langle$ D A N P D N P D A N V D A N $\rangle$.
5. $\text{Test}_5 = \langle$ D A N P D N P D A N V P D A N $\rangle$.

All of these long sentences can be successfully generalized by the CPP. If we look at the training set, each has a similar counterpart (where Test_i corresponds to Train_i, i being from 1 to 5):

1. $\text{Train}_1 = \langle$ D A N P D N P D N V D N P D N $\rangle$.
2. $\text{Train}_2 = \langle$ D A N P D N P D N V D A N P D N $\rangle$.
3. $\text{Train}_3 = \langle$ D N P D N P D A N V D N P D N $\rangle$.
4. $\text{Train}_4 = \langle$ D N P D N P D A N V D A N $\rangle$.
5. $\text{Train}_5 = \langle$ D N P D N P D A N V P D A N $\rangle$.

Each Test_i differs from the respective Train_i by one terminal only, and they share the same ending. In the experiments, the CPP is capable of distinguishing the two sentences in each pair and produces the correct parse tree for each (in fact, in none of the testing cases has the CPP misinterpreted a testing sentence as another training or testing sentence). That means that

the distinguishing feature in each case, albeit small and appearing early in the sentence, can still be preserved as the remaining terminals are read. This reflects that given the maximum length of the sentences used in the experiments (which is 17), the influence of long-term dependencies on the CPP's performance is not noticeably significant. Its effect may become more prominent when longer sentences are involved.

Representing parse trees hierarchically will suffer a similar problem as the long-term dependency effect. If the parse trees of two sentences differ only slightly at the leaf level, the difference may be lost during the recursive bottom-up encoding process, so the parser may have difficulty distinguishing them, and the same parse tree is produced. For example, when the testing sentence ⟨ D A N P D N P D A N V D A N ⟩ is parsed, the P_{RAAM} gives ((((D N) (P (D N))) (P (D (A N)))) (V (D (A N)))), which actually corresponds to the training sentence ⟨ D N P D N P D A N V D A N ⟩. Such errors have in fact occurred several times for the P_{RAAM} and have caused its poor generalization performance. This suggests that the long-term dependency effect is more apparent when parse trees are encoded hierarchically than when they are linearized.

7 Discussion and Conclusion

We have proposed a method for analyzing holistic parsers. Based on the formalism, holistic parsing is achieved as if a rational inference process is involved during which the parse tree is built in a step-by-step manner. This illustrates that abstraction of grammatical knowledge has actually occurred during training. The analysis also provides insight into the issue of how the performance of a holistic parser is affected by the encoding methods for sentences and parse trees. Although we have demonstrated the approach by using the CPP only, we think that it is equally applicable to other holistic parsers, provided that a recurrent network of some type is employed for encoding sentences.

To a certain extent, our approach can also be used as a general method for inducing finite-state approximation of context-free grammars. Although finite-state automata are less powerful than their context-free counterparts, they are more efficient language models computationally. In fact, Roche (1997) has shown that finite-state formalisms are capable of modeling a wide range of syntactic structures of the English language efficiently. But conventional approaches (e.g., Pereira & Wright, 1997) usually require the full specification of the grammar in order to build the finite-state automaton for parsing, which may not be available in practice (especially when natural languages are concerned).

Other researchers have also attempted to extract a finite-state automaton from a recurrent network. Our model differs from theirs in two major aspects. First, these approaches dealt with regular languages primarily, and the network was trained on either a sequence prediction task (e.g., Servan-

Schreiber et al., 1991) or a sequence classification task (e.g., Giles et al., 1992; Lawrence, Giles, & Fong, 1998; Watrous & Kuhn, 1992). But for holistic parsers, we aim at parsing context-free languages exclusively. Thus, we can at best obtain a finite-state approximation of the target context-free grammar only.

Second, states are identified differently. In previous approaches, clustering or quantization techniques were often applied to partition the hidden-layer activation space of the network into states. For example, in Giles et al. (1992), each hidden neuron's range was divided into q partitions of equal width. Thus for N hidden neurons, there are q^N possible states. Unavoidably, certain parameters will be involved in these algorithms (such as q) that may affect the number of states and the finite-state automaton extracted. More important, the performance of the finite-state automaton may also be influenced. But often their optimal values can only be determined empirically (Giles et al., 1992).

As different from other approaches, our extraction method is data oriented and assumes no prior knowledge of the grammar. The states are defined deterministically by the symbolic decoding's results, so no parameters are involved, and they are directly interpretable. Moreover, final states have larger basins of attraction than nonfinal states. This implies that it is not always possible to tell whether two points in the state-space belong to the same state by measuring the distance between them. In some sense, this rules out methods where states are defined by clustering the distributed representations using some kind of numerical metric (such as the Euclidean metric).

Despite its advantages, there is a concern that the number of states as identified by our model may be unbounded. New states may potentially be created when novel or ungrammatical sentences are parsed (although states are reused sometimes). On the one hand, this result is natural; finite-state automata can recognize only regular grammars, which are less powerful language models than context-free grammars. So to parse more sentences, more states are needed. Paradoxically, as a context-free grammar can generate an infinite number of sentences in general, the finite-state automaton extracted will need an infinite number of states in the extreme case.

On the other hand, the number of states can actually be reduced. We may allow two states whose identities are only slightly different to be treated as the same state. Consider Figure 10 again. When ⟨ D A N V P A N P D N ⟩ is parsed, two new states are given rise: states 226 and 227. Upon decoding, state 226 gives ⟨ s np D ap A N vp V pp P D ap A N V ⟩, which differs from the identity of state 9 (⟨ s np D ap A N vp V np D D ap A N V ⟩) by two terminals only. Hence, state 226 can be treated as the same state as state 9. Similarly, states 227 and 10 can be merged. Note that a parameter has to be defined here, which is the maximum number of mismatched terminals allowed. It serves as a similarity measure that decides when two states can be merged.

Alternatively, we may restrict the maximum length of the states' identities. Recall that when a distributed representation is decoded, a sequence is produced from right to left recursively, which may possibly represent the preorder traversal of a syntactically well-formed parse tree. Suppose the maximum length is set to n. Then all but the right-most n symbols of the sequence are discarded. Intuitively, only the right-most n symbols are considered significant. The remaining subsequence is then used as the identity of the state. As the number of symbols is finite, the number of states should also be bounded.[10]

Besides being an explanatory device, our model helps to identify those factors that are pertinent to the performance of a holistic parser. The generalization capability is related to the number of final states that can be realized through training as well as their distribution in the state-space, and the sizes of the basins of attraction of the final states will affect the robustness. These characteristics are mainly dependent on the training network's process. It is thus useful to explore their relationship with factors such as the number of hidden units (which determines the maximum representational capacity of the network), the training environment (e.g., the learning rate), the coding for the terminals, and the training patterns used (their variety and combination).

In fact, we believe that there exists a trade-off between the number of final states formed and the average size of their basins of attraction. Although increasing the number of final states (say, by using more training examples) can improve generalization capability, it may also decrease the sizes of their basins of attraction and increase the "density" of the representational space. Robustness is thus degraded. So, it is useful to have a formulation of this trade-off in terms of the type of performance expected (e.g., whether generalization or robust parsing is of more concern, and the level of robustness desired). More important, we hope that through regulating the training process, we can exercise control over the attractors and states formed (their number, their distribution in the state-space, and their sizes) (see e.g., Tsung & Cottrell, 1993). In this way, we can position the holistic parser at a "suitable point" in the trade-off where the specific performance demand is met.[11] This may shed light on the further improvement of the performance of a holistic parser.

[10] These two techniques can be applied only to nonfinal states, as each final state corresponds to a unique parse tree. If their number is reduced, the number of sentences that can be parsed will decrease also. So in the extreme case, the number of final states is still unlimited.

[11] For example, by choosing the initial weights of the network carefully, we might be able to affect the representations formed, which will influence the distribution of the states.

Acknowledgments

The work described in this article was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (PolyU 4133/97E).

References

Allen, J. (1995). *Natural language understanding*. Redwood City, CA: Benjamin/Cummings.

Angluin, D. (1980). Inductive inference of formal languages from positive data. *Information and Control*, *45*, 117–135.

Berg, G. (1992). A connectionist parser with recursive sentence structure and lexical disambiguation. In *Proceedings of the Tenth Conference on Artificial Intelligence (AAAI-92), San Jose* (pp. 32–37).

Chalmers, D. J. (1992). Syntactic transformation of distributed representations. In N. Sharkey (Ed.), *Connectionist natural language processing* (pp. 46–55). Norwell, MA: Kluwer.

Charniak, E. (1993). *Statistical language learning*. Cambridge, MA: MIT Press.

Chrisman, L. (1991). Learning recursive distributed representations for holistic computation. *Connection Science*, *3*, 345–366.

Christiansen, M. H., & Chater, N. (1994). Generalization and connectionist language learning. *Mind and Language*, *9*, 273–287.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Fanty, M. (1985). *Context-free parsing in connectionist networks* (Tech. Rep. No. TR-174). Rochester, NY: University of Rochester.

Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, *28*, 3–71.

Gazdar, G., & Mellish, C. (1989). *Natural language processing in Prolog: An introduction to computational linguistics*. Reading, MA: Addison-Wesley.

Giles, C., Miller, C., Chen, D., Chen, H., Sun, G., & Lee, Y. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, *4*, 393–405.

Hadley, R. F. (1994). Systematicity in connectionist language learning. *Mind and Language*, *9*, 247–272.

Hanson, S. J., & Kegl, J. (1987). PARSNIP: A connectionist network that learns natural language grammar from exposure to natural language sentences. In *Proceedings of the Ninth Annual Cognitive Science Society Conference* (pp. 106–119).

Ho, K. S. E., & Chan, L. W. (1994). Representing sentence structures in neural networks. In *Proceedings of the International Conference in Neural Information Processing Systems (ICONIP'94), Seoul* (pp. 1462–1467).

Ho, K. S. E., & Chan, L. W. (1997). Confluent preorder parsing of deterministic grammars. *Connection Science*, *9*, 269–293.

Ho, K. S. E., & Chan, L. W. (1999). How to design a connectionist holistic parser. *Neural Computation*, *11*, 2085–2106.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages, and computation*. Reading, MA: Addison-Wesley.

Jain, A. N. (1991). Parsing complex sentences with structured connectionist networks. *Neural Computation*, *3*, 110–120.

Kolen, J. F. (1994). *Exploring the computational capabilities of recurrent neural Networks.* Unpublished doctoral dissertation, Ohio State University.

Kwasny, S. C., & Faisal, K. A. (1992). Symbolic parsing via subsymbolic rules. In J. Dinsmore (Ed.), *The symbolic and connectionist paradigm: Closing the gap* (pp. 209–236). Hillsdale, NJ: Erlbaum.

Kwasny, S. C., & Kalman, B. L. (1995). Tail-recursive distributed representations and simple recurrent networks. *Connection Science*, *7*, 61–80.

Lawrence, S., Giles, C., & Fong, S. (1998). Natural language grammatical inference with recurrent neural networks. *IEEE Transactions in Knowledge and Data Engineering*, *12*, 126–140.

Lin, T., Horne, B. G., Tino, P., & Giles, C. L. (1996). Learning long-term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, *7*, 1329–1338.

Miikkulainen, R. (1996). Subsymbolic case-role analysis of sentences with embedded clauses. *Cognitive Science*, *20*, 47–73.

Niklasson, L. F., & Van Gelder, T. (1994). On being systematically connectionist. *Mind and Language*, *9*, 288–302.

Pereira, F. C. N., & Wright, R. N. (1997). Finite-state approximation of phrase-structure grammars. In E. Roche & Y. Schabes (Eds.), *Finite-state language processing* (pp. 149–173). Cambridge, MA: MIT Press.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, *46*, 77–105.

Reilly, R. (1992). Connectionist techniques for on-line parsing. *Network*, *3*, 37–45.

Roche, E. (1997). Parsing with finite-state transducers. In E. Roche & Y. Schabes (Eds.), *Finite-state language processing* (pp. 241–281). Cambridge, MA: MIT Press.

Rodriguez, P., Wiles, J., & Elman, J. L. (1999). A recurrent neural network that learns to count. *Connection Science*, *11*, 5–40.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations through error propagation. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing: Experiments in the microstructure of cognition* (pp. 318–362). Cambridge, MA: MIT Press.

Santos, E. (1989). A massively parallel self-tuning context free parser. In D. S. Touretzky (Ed.), *Advances in neural information processing systems*, *1* (pp. 537–544). San Mateo, CA: Morgan Kaufmann.

Selman, B. A. (1985). *Rule-based processing in connectionist system for natural language understanding* (Tech. Rep. No. CSRT-168). Toronto: Computer Science Research Institute, University of Toronto.

Servan-Schreiber, D., Cleeremans, A., & McClelland, J. L. (1991). Graded state machine: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, *7*, 161–193.

Sharkey, N. E., & Sharkey, A. J. C. (1992). A modular design for connectionist parsing. In A. N. Marc & F. J. Drossaers (Eds.), *Proceedings of the Twente Work-*

shop on Language Technology 3: Connectionism and Natural Language Processing (pp. 87–96).

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence, 46*, 159–216.

St. John, M. F., & McClelland, J. L. (1990). Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence, 46*, 217–257.

Sun, G. Z., Giles, C. L., Chen, H. H., & Lee, Y. C. (1993). *The neural network pushdown automata: Model, stack and learning simulations* (Tech. Rep. Nos. UMIACS-TR-93-77). College Park: University of Maryland.

Tsung, F.-S., & Cottrell, G. W. (1993). *Phase-space learning for recurrent networks* (Tech. Rep. No. CS93-285). San Diego: Department of Computer Science and Engineering, University of California, San Diego.

Watrous, R., & Kuhn, G. (1992). Induction of finite state languages using second-order recurrent networks. In J. Moody, S. Hanson, & R. Lippman (Eds.), *Advances in neural information processing systems, 4* (pp. 309–316). San Mateo, CA: Morgan Kaufmann.

Received March 25, 1999; accepted June 15, 2000.