Deductive and Inductive Learning in a Connectionist Deterministic Parser

Kanaan A. Faisal & Stan C. Kwasny

Center for Intelligent Computer Systems[1]
Department of Computer Science
Washington University, St. Louis, MO 63130

INTRODUCTION

Deterministic Parsing of Natural Language, as performed by PARSIFAL (Marcus, 1980), has shown that such a task can be conducted under the somewhat severe restriction of the determinism hypothesis using a rule-based approach. Others have independently extended PARSIFAL to the task of parsing ungrammatical sentences in PARAGRAM (Charniak, 1983), resolving lexical ambiguities in ROBIE (Milne, 1986), and acquiring syntax in LPARSIFAL (Berwick, 1985).

Why have these researchers chosen to focus on extensions with these rather narrow goals? The answer, in part, lies in the general difficulty of the task and the limitations of conventional, symbolic means. We have found it beneficial to combine these tasks into one implementation which is partly symbolic and partly sub-symbolic. The results of our experiments hold implications for rule-based expert systems.

A Connectionist Deterministic Parser (CDP) is under development (Kwasny & Faisal, 1989). CDP combines the concepts and ideas from deterministic parsing together with the generalization and robustness of connectionist, adaptive (neural) networks. A backpropagation neural network simulator, which features a logistic function that computes values in the range of -1 to $+1$, is being used in this work. The ultimate goal is to produce a parser that has some reasonable facility with language and does not fail on inputs that are only slightly different from the inputs it is designed to process.

There are important advantages to constructing rule-based systems using neural networks (Gallant, 1988). Our focus is on building a connectionist parser, but with more general issues in mind. How successfully can a connectionist parser be constructed and what are the advantages? Success clearly hinges on the careful selection of training sequences. Our experiments have examined two different approaches and compared them.

The "deductive" strategy uses rule "templates" derived from the rules of a deterministic grammar. It is deductive in the sense that it is based on general knowledge in the form of rules although the resultant network is required to process specific sentences. The "inductive" strategy derives its training sequence from coded examples of sentence processing. It is inductive in the sense that it is based on traces of the processing of specific sentences but is required to generalize to a wider range of examples. The goal of deductive learning initially is to produce a network which is capable of mimicking the rules on which training is based. The goal of inductive learning initially is to produce a network which is capable of processing the sentences on which its training is based. Once that learning has been accomplished, simulation experiments are done to examine certain generalization capabilities of the resulting networks.

Deductive training generally performs well on all generalization tasks and outperforms inductive training by scoring generally higher on all experiments. Reasons for this include the specificity of the inductive training data as well as the lack of a large amount of training data in the inductive case required to provide sufficient variety.

LEARNING A RULE-BASED GRAMMAR

A deterministic parser applies rules to a stack and buffer of constituents to generate and perform actions on those structures. One of its primary features is that it does not backtrack, but proceeds forward in its processing never building structures which are later discarded.

Training of CDP proceeds by presenting patterns to the network and teaching it to respond with an appropriate action using backpropagation (Rumelhart, et al, 1986). The input patterns represent encodings of the buffer positions and the top of the stack from the deterministic parser. The output of the network contains a series of units representing actions to be performed during processing and judged in a winner-take-all fashion. Network convergence is observed once the network can achieve a perfect score on the training patterns themselves and the error measure has decreased to an acceptable level (set as a parameter).

[1] The sponsors of the Center are McDonnell Douglas Corporation and Southwestern Bell Telephone Company.

Once the network is trained, the weights are stored in a file so that various experiments can be performed. Each sentence receives a score representing the overall average strength of responses during processing. The score for each processing step is computed as the reciprocal of the error for that step. The error is computed as the Euclidean distance between the actual output and an idealized output consisting of a -1 value for every output unit except the winning unit which has a +1 value. The errors for each step are summed and averaged over the number of steps. The average strength is the reciprocal of the average error per step.

Deductive Learning. Each grammar rule is coded as a training template which is a list of feature values, but templates are not grouped into rule packets. In general, each constituent is represented by an ordered feature vector in which one or more values is ON(+1) for features of the form and all other values are either OFF(-1) or DO NOT CARE (?). A rule template is instantiated by randomly changing ? to +1 or -1. Thus, each template represents many training patterns and each training epoch is slightly different. During training, the network learns the inputs upon which it can rely in constructing its answers and which inputs it can ignore.

The probability of a ? becoming a +1 or -1 is equal and set at 0.5. Each rule template containing n ?'s can generate up to 2^n training cases. Some rule templates have over 30 ?'s which means they represent approximately 10^9 unique training cases. It is obviously impossible to test the performance of all these cases, so a zero is substituted for each ? to provide testing patterns. Zero is used since it represents the mean of the range of values seen during training.

A slightly modified version of the grammar from appendix C of Marcus (1980) was used as a basis for all deductive training experiments in this paper. This appendix includes the rules specifically discussed by Marcus in building his case for deterministic parsing and can be taken as representative of the mechanisms involved. To assure good performance by the network, training has ranged from 50,000 to 200,000 presentations cycling through training cases generated from the rule templates.

Inductive Learning. Inductive learning begins with training data derived as "sentence traces" of deterministic parsing steps. Training proceeds by presenting patterns of these steps to the network and teaching it to respond with an appropriate action. A small set of positive sentence examples were traced which resulted in 64 unique training patterns. These were used for all inductive experiments in this paper.

This approach can be compared with LPARSIFAL which attempts to learn PARSIFAL rules from examples of *positive evidence* (i.e., grammatical sentences). LPARSIFAL starts with a small set of rules and gradually builds up new rules. In effect, the system is inductively learning the grammar rules from sentence examples. The target for learning in LPARSIFAL is the PARSIFAL grammar. LPARSIFAL requires several hundred sentences to acquire approximately 70% of the parsing rules originally hand-written for the Marcus parser. In our experiments, the network exhibited better than 70% coverage of our rule-based grammar after training on a small number of traces.

NETWORK ARCHITECTURE

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

In the set of experiments described here, the network has a three-layer architecture with 35 input units, 20 hidden units, and 20 output units. Each input pattern consists of three feature vectors from the buffer items and one stack vector. Each vector activates 14 input units in a pattern vector representing a word or constituent of the sentence. The stack vector activates seven units representing the current node on the stack. In our simplified version of the grammar, only two items are coded from the buffer and thus 35 input units are sufficient. One hidden layer has proven sufficient in all of our experiments. The output layer represents the 20 possible actions that can be performed on each iteration of processing. All weights in the network are initialized to random values between –0.3 and +0.3.

During sentence processing, the network is presented with encodings of the buffer and the top of the stack. What the model actually sees as input is not the raw sentence but a canonical representation of each word in the sentence in a form that could be produced by a simple lexicon, although such a lexicon is not part of the model in its present form. The network produces the action to be taken which is then performed. If the action creates a vacancy in the buffer and if more of the sentence is left to be processed then the next sentence component is moved into the buffer. The process then repeats until a stop action is performed, usually when the buffer becomes empty. Iteration over the input stream is achieved in this fashion.

PERFORMANCE COMPARISON

CDP is capable of processing a variety of simple sentence forms such as simple declarative, simple passive, and imperative sentences as well as yes-no questions. For test and comparison purposes, several sentences were coded that would parse correctly by the rules of the deterministic parser. Also, several mildly ungrammatical and lexically ambiguous sentences were coded to determine if the network would generalize in any useful way. The objective was to test if the syntactic context could aid in resolving such problems.

TABLE 1: Grammatical Sentences Used in Testing

	Sentence Form	Deductive Average Strength	Inductive Average Strength
(1)	John should have scheduled the meeting.	283.3	84.7
(2)	John has scheduled the meeting for Monday.	179.3	84.2
(3)	Has John scheduled the meeting?	132.2	64.4
(4)	John is scheduling the meeting.	294.4	83.5
(5)	The boy did hit Jack.	298.2	76.2
(6)	Schedule the meeting.	236.2	67.8
(7)	Mary is kissed.	276.1	84.9
(8)	Tom hit(v) Mary.	485.0	80.3
(9)	Tom will(aux) hit(v) Mary.	547.5	78.7
(10)	They can(v) fish(np).	485.0	80.3
(11)	They can(aux) fish(v).	598.2	76.8

Parsing Grammatical Sentences. Grammatical sentences, by our definition, are those which parse correctly in the rule-based grammar from which we derived the training set. Table 1 shows several examples of grammatical sentences which are parsed successfully along with their response strengths in both deductive and inductive learning. Each example shows a relatively high average strength value, indicating that the training data has been learned. Also, the deductive average strength value is higher than the corresponding inductive average strength.

TABLE 2: Ungrammatical Sentences Used in Testing

	Sentence Form	Deductive Average Strength	Inductive Average Strength
(12)	*John have should scheduled the meeting.	25.1	6.6
(13)	*Has John schedule the meeting?	38.1	18.2
(14)	*John is schedule the meeting.	4.7	4.9†
(15)	*The boy did hitting Jack.	26.6	7.5‡

Parsing Ungrammatical Sentences. An important test of the generalization capabilities of CDP is its response to ungrammatical sentences. Such capabilities are strictly dependent upon the experiences of the network during training since in deductive training no relaxation rules were added to the original grammar to handle ungrammatical cases and in inductive training no ungrammatical sentences were used.

In this set of experiments a few ungrammatical sentences were tested that were similar to the training data and within the scope of our encoding. Table 2 contains examples that have produced reasonable structures when presented to our system along with their response strengths. Note that overall average strength is lower for ungrammatical sentences when compared to similar grammatical ones.

In sentence (12), the structure produced was identical to that produced while parsing sentence (1), but with lower strength in the inductive case. The only difference is that the two auxiliary verbs, *have* and *should*, were reversed. Sentence (13) contains a disagreement between the auxiliary *has* and the main verb *schedule* and yet the comparable grammatical sentence (3) parsed identically in both approaches, but with lower strength again in the inductive approach.

Sentence (14) can be compared with sentence (4). In the deductive case, a structure similar to that built for sentence (4) is indeed constructed. However, in the inductive case (marked with †), the network attempts to process 'is' as if it were indicating the passive tense. Although this is incorrect for this sentence, it is not an unreasonable choice. Sentence (15) can be compared with sentence (5), but there is not one clear choice in how the sentence should appear if grammatical. The deductive-trained network processes sentence (15) as sentence (5), while the inductive result (marked with ‡) shows the sentence processed as if it were progressive tense ('The boy is hitting Jack'). In PARAGRAM, a nonsensical parse structure is produced for sentence (15), as reported by Charniak (p. 137).

TABLE 3: Lexically Ambiguous Sentences Used in Testing

	Sentence Form (Words in <> are presented ambiguously)	Deductive Average Strength	Inductive Average Strength
(16)	<Will> he go?	83.6	14.3
(17)	Tom <will> hit Mary.	118.7	19.9
(18)	Tom <hit> Mary.	39.0	2.5
(19)	They <can> fish.	4.5	2.6
(20)	They can <fish>.	172.2	4.9

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

Sentence (17) contains the word *will* coded ambiguously as an NP and an auxiliary, modal verb. In the context of the sentence, it is clearly being used as a modal auxiliary and the parser treats it that way. A similar result was obtained for sentence (18). In sentence (18), *hit* is coded to be ambiguous between an NP (as in playing cards) and a verb. The network correctly identifies it as the main verb of the sentence. Sentence (19) presents *can* ambiguously as an auxiliary, modal, and main verb, while *fish* is presented uniquely as an NP. *Can* is processed as the main verb of the sentence. Compare this example with sentence (10) of Table 1. Here, each word is presented unambiguously with *can* coded as a verb and *fish* coded as an NP. The same structure results in each case, with the average strength level much higher in the unambiguous case. By coding *fish* ambiguously as a verb/NP and coding *can* uniquely as an auxiliary, the result obtained is as shown for sentence (20), which is comparable to sentence (11).

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

DISCUSSION

While deductive training exhibits better performance than inductive training for all tasks, there are tradeoffs in the two approaches. Deductive training requires rules as the basis for rule templates while inductive training requires a large amount of data to be successful. Fortunately there is a middle ground which allows mixtures of the two training strategies. Training can be performed using rule templates as well as patterns based on sentence traces. In a recent set of experiments in which the two types of training data were combined, the network was capable of generalizing in ways similar to deductive learning, but also showed particularly good performance on the specific cases reflected in the inductive data.

What does this mean for expert systems? Where knowledge naturally exists in rule form and such rules can be reliably stated, rule templates can be formed which generate appropriate training cases. However, where knowledge only exists in the form of anecdotal cases, it can be expressed in the form of inductive training patterns. As new cases are discovered for which the rules do not apply, inductive data can be easily constructed and the network retrained. Contrast this with the typical rule-based expert system in which each new rule may require re-thinking an entire set of existing rules.

Our work has shown the Trade-tradeoffs between deductive and inductive learning. Both have a place in the construction of a neural network designed to perform a complex rule-based task such as parsing.

REFERENCES

Berwick, Robert C. (1985). *The Acquisition of Syntactic Knowledge*, Cambridge: MIT Press.

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

Gallant, Stephen I. (1988). Connectionist Expert Systems. *Communications of the ACM 31*, 2, 152-169.

Kwasny, S.C., and Faisal, K.A. (1989). Competition and Learning in a Connectionist Deterministic Parser. *Proceedings of the 11th Annual Conference of Cognitive Science Society*, Ann Arbor, Michigan.

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

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

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

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

Deductive and Inductive Learning in a Connectionist Deterministic Parser

Kanaan A. Faisal & Stan C. Kwasny

Center for Intelligent Computer Systems[1]
Department of Computer Science
Washington University, St. Louis, MO 63130

INTRODUCTION

Deterministic Parsing of Natural Language, as performed by PARSIFAL (Marcus, 1980), has shown that such a task can be conducted under the somewhat severe restriction of the determinism hypothesis using a rule-based approach. Others have independently extended PARSIFAL to the task of parsing ungrammatical sentences in PARAGRAM (Charniak, 1983), resolving lexical ambiguities in ROBIE (Milne, 1986), and acquiring syntax in LPARSIFAL (Berwick, 1985).

Why have these researchers chosen to focus on extensions with these rather narrow goals? The answer, in part, lies in the general difficulty of the task and the limitations of conventional, symbolic means. We have found it beneficial to combine these tasks into one implementation which is partly symbolic and partly sub-symbolic. The results of our experiments hold implications for rule-based expert systems.

A Connectionist Deterministic Parser (CDP) is under development (Kwasny & Faisal, 1989). CDP combines the concepts and ideas from deterministic parsing together with the generalization and robustness of connectionist, adaptive (neural) networks. A backpropagation neural network simulator, which features a logistic function that computes values in the range of -1 to $+1$, is being used in this work. The ultimate goal is to produce a parser that has some reasonable facility with language and does not fail on inputs that are only slightly different from the inputs it is designed to process.

There are important advantages to constructing rule-based systems using neural networks (Gallant, 1988). Our focus is on building a connectionist parser, but with more general issues in mind. How successfully can a connectionist parser be constructed and what are the advantages? Success clearly hinges on the careful selection of training sequences. Our experiments have examined two different approaches and compared them.

The "deductive" strategy uses rule "templates" derived from the rules of a deterministic grammar. It is deductive in the sense that it is based on general knowledge in the form of rules although the resultant network is required to process specific sentences. The "inductive" strategy derives its training sequence from coded examples of sentence processing. It is inductive in the sense that it is based on traces of the processing of specific sentences but is required to generalize to a wider range of examples. The goal of deductive learning initially is to produce a network which is capable of mimicking the rules on which training is based. The goal of inductive learning initially is to produce a network which is capable of processing the sentences on which its training is based. Once that learning has been accomplished, simulation experiments are done to examine certain generalization capabilities of the resulting networks.

Deductive training generally performs well on all generalization tasks and outperforms inductive training by scoring generally higher on all experiments. Reasons for this include the specificity of the inductive training data as well as the lack of a large amount of training data in the inductive case required to provide sufficient variety.

LEARNING A RULE-BASED GRAMMAR

A deterministic parser applies rules to a stack and buffer of constituents to generate and perform actions on those structures. One of its primary features is that it does not backtrack, but proceeds forward in its processing never building structures which are later discarded.

Training of CDP proceeds by presenting patterns to the network and teaching it to respond with an appropriate action using backpropagation (Rumelhart, et al, 1986). The input patterns represent encodings of the buffer positions and the top of the stack from the deterministic parser. The output of the network contains a series of units representing actions to be performed during processing and judged in a winner-take-all fashion. Network convergence is observed once the network can achieve a perfect score on the training patterns themselves and the error measure has decreased to an acceptable level (set as a parameter).

[1] The sponsors of the Center are McDonnell Douglas Corporation and Southwestern Bell Telephone Company.

Once the network is trained, the weights are stored in a file so that various experiments can be performed. Each sentence receives a score representing the overall average strength of responses during processing. The score for each processing step is computed as the reciprocal of the error for that step. The error is computed as the Euclidean distance between the actual output and an idealized output consisting of a -1 value for every output unit except the winning unit which has a +1 value. The errors for each step are summed and averaged over the number of steps. The average strength is the reciprocal of the average error per step.

Deductive Learning. Each grammar rule is coded as a training template which is a list of feature values, but templates are not grouped into rule packets. In general, each constituent is represented by an ordered feature vector in which one or more values is ON(+1) for features of the form and all other values are either OFF(-1) or DO NOT CARE (?). A rule template is instantiated by randomly changing ? to +1 or -1. Thus, each template represents many training patterns and each training epoch is slightly different. During training, the network learns the inputs upon which it can rely in constructing its answers and which inputs it can ignore.

The probability of a ? becoming a +1 or -1 is equal and set at 0.5. Each rule template containing n ?'s can generate up to 2^n training cases. Some rule templates have over 30 ?'s which means they represent approximately 10^9 unique training cases. It is obviously impossible to test the performance of all these cases, so a zero is substituted for each ? to provide testing patterns. Zero is used since it represents the mean of the range of values seen during training.

A slightly modified version of the grammar from appendix C of Marcus (1980) was used as a basis for all deductive training experiments in this paper. This appendix includes the rules specifically discussed by Marcus in building his case for deterministic parsing and can be taken as representative of the mechanisms involved. To assure good performance by the network, training has ranged from 50,000 to 200,000 presentations cycling through training cases generated from the rule templates.

Inductive Learning. Inductive learning begins with training data derived as "sentence traces" of deterministic parsing steps. Training proceeds by presenting patterns of these steps to the network and teaching it to respond with an appropriate action. A small set of positive sentence examples were traced which resulted in 64 unique training patterns. These were used for all inductive experiments in this paper.

This approach can be compared with LPARSIFAL which attempts to learn PARSIFAL rules from examples of *positive evidence* (i.e., grammatical sentences). LPARSIFAL starts with a small set of rules and gradually builds up new rules. In effect, the system is inductively learning the grammar rules from sentence examples. The target for learning in LPARSIFAL is the PARSIFAL grammar. LPARSIFAL requires several hundred sentences to acquire approximately 70% of the parsing rules originally hand-written for the Marcus parser. In our experiments, the network exhibited better than 70% coverage of our rule-based grammar after training on a small number of traces.

NETWORK ARCHITECTURE

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

In the set of experiments described here, the network has a three-layer architecture with 35 input units, 20 hidden units, and 20 output units. Each input pattern consists of three feature vectors from the buffer items and one stack vector. Each vector activates 14 input units in a pattern vector representing a word or constituent of the sentence. The stack vector activates seven units representing the current node on the stack. In our simplified version of the grammar, only two items are coded from the buffer and thus 35 input units are sufficient. One hidden layer has proven sufficient in all of our experiments. The output layer represents the 20 possible actions that can be performed on each iteration of processing. All weights in the network are initialized to random values between –0.3 and +0.3.

During sentence processing, the network is presented with encodings of the buffer and the top of the stack. What the model actually sees as input is not the raw sentence but a canonical representation of each word in the sentence in a form that could be produced by a simple lexicon, although such a lexicon is not part of the model in its present form. The network produces the action to be taken which is then performed. If the action creates a vacancy in the buffer and if more of the sentence is left to be processed then the next sentence component is moved into the buffer. The process then repeats until a stop action is performed, usually when the buffer becomes empty. Iteration over the input stream is achieved in this fashion.

PERFORMANCE COMPARISON

CDP is capable of processing a variety of simple sentence forms such as simple declarative, simple passive, and imperative sentences as well as yes-no questions. For test and comparison purposes, several sentences were coded that would parse correctly by the rules of the deterministic parser. Also, several mildly ungrammatical and lexically ambiguous sentences were coded to determine if the network would generalize in any useful way. The objective was to test if the syntactic context could aid in resolving such problems.

TABLE 1: Grammatical Sentences Used in Testing

	Sentence Form	Deductive Average Strength	Inductive Average Strength
(1)	John should have scheduled the meeting.	283.3	84.7
(2)	John has scheduled the meeting for Monday.	179.3	84.2
(3)	Has John scheduled the meeting?	132.2	64.4
(4)	John is scheduling the meeting.	294.4	83.5
(5)	The boy did hit Jack.	298.2	76.2
(6)	Schedule the meeting.	236.2	67.8
(7)	Mary is kissed.	276.1	84.9
(8)	Tom hit(v) Mary.	485.0	80.3
(9)	Tom will(aux) hit(v) Mary.	547.5	78.7
(10)	They can(v) fish(np).	485.0	80.3
(11)	They can(aux) fish(v).	598.2	76.8

Parsing Grammatical Sentences. Grammatical sentences, by our definition, are those which parse correctly in the rule-based grammar from which we derived the training set. Table 1 shows several examples of grammatical sentences which are parsed successfully along with their response strengths in both deductive and inductive learning. Each example shows a relatively high average strength value, indicating that the training data has been learned. Also, the deductive average strength value is higher than the corresponding inductive average strength.

TABLE 2: Ungrammatical Sentences Used in Testing

	Sentence Form	Deductive Average Strength	Inductive Average Strength
(12)	*John have should scheduled the meeting.	25.1	6.6
(13)	*Has John schedule the meeting?	38.1	18.2
(14)	*John is schedule the meeting.	4.7	4.9†
(15)	*The boy did hitting Jack.	26.6	7.5‡

Parsing Ungrammatical Sentences. An important test of the generalization capabilities of CDP is its response to ungrammatical sentences. Such capabilities are strictly dependent upon the experiences of the network during training since in deductive training no relaxation rules were added to the original grammar to handle ungrammatical cases and in inductive training no ungrammatical sentences were used.

In this set of experiments a few ungrammatical sentences were tested that were similar to the training data and within the scope of our encoding. Table 2 contains examples that have produced reasonable structures when presented to our system along with their response strengths. Note that overall average strength is lower for ungrammatical sentences when compared to similar grammatical ones.

In sentence (12), the structure produced was identical to that produced while parsing sentence (1), but with lower strength in the inductive case. The only difference is that the two auxiliary verbs, *have* and *should*, were reversed. Sentence (13) contains a disagreement between the auxiliary *has* and the main verb *schedule* and yet the comparable grammatical sentence (3) parsed identically in both approaches, but with lower strength again in the inductive approach.

Sentence (14) can be compared with sentence (4). In the deductive case, a structure similar to that built for sentence (4) is indeed constructed. However, in the inductive case (marked with †), the network attempts to process 'is' as if it were indicating the passive tense. Although this is incorrect for this sentence, it is not an unreasonable choice. Sentence (15) can be compared with sentence (5), but there is not one clear choice in how the sentence should appear if grammatical. The deductive-trained network processes sentence (15) as sentence (5), while the inductive result (marked with ‡) shows the sentence processed as if it were progressive tense ('The boy is hitting Jack'). In PARAGRAM, a nonsensical parse structure is produced for sentence (15), as reported by Charniak (p. 137).

TABLE 3: Lexically Ambiguous Sentences Used in Testing

	Sentence Form (Words in <> are presented ambiguously)	Deductive Average Strength	Inductive Average Strength
(16)	<Will> he go?	83.6	14.3
(17)	Tom <will> hit Mary.	118.7	19.9
(18)	Tom <hit> Mary.	39.0	2.5
(19)	They <can> fish.	4.5	2.6
(20)	They can <fish>.	172.2	4.9

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

Sentence (17) contains the word *will* coded ambiguously as an NP and an auxiliary, modal verb. In the context of the sentence, it is clearly being used as a modal auxiliary and the parser treats it that way. A similar result was obtained for sentence (18). In sentence (18), *hit* is coded to be ambiguous between an NP (as in playing cards) and a verb. The network correctly identifies it as the main verb of the sentence. Sentence (19) presents *can* ambiguously as an auxiliary, modal, and main verb, while *fish* is presented uniquely as an NP. *Can* is processed as the main verb of the sentence. Compare this example with sentence (10) of Table 1. Here, each word is presented unambiguously with *can* coded as a verb and *fish* coded as an NP. The same structure results in each case, with the average strength level much higher in the unambiguous case. By coding *fish* ambiguously as a verb/NP and coding *can* uniquely as an auxiliary, the result obtained is as shown for sentence (20), which is comparable to sentence (11).

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

DISCUSSION

While deductive training exhibits better performance than inductive training for all tasks, there are tradeoffs in the two approaches. Deductive training requires rules as the basis for rule templates while inductive training requires a large amount of data to be successful. Fortunately there is a middle ground which allows mixtures of the two training strategies. Training can be performed using rule templates as well as patterns based on sentence traces. In a recent set of experiments in which the two types of training data were combined, the network was capable of generalizing in ways similar to deductive learning, but also showed particularly good performance on the specific cases reflected in the inductive data.

What does this mean for expert systems? Where knowledge naturally exists in rule form and such rules can be reliably stated, rule templates can be formed which generate appropriate training cases. However, where knowledge only exists in the form of anecdotal cases, it can be expressed in the form of inductive training patterns. As new cases are discovered for which the rules do not apply, inductive data can be easily constructed and the network retrained. Contrast this with the typical rule-based expert system in which each new rule may require re-thinking an entire set of existing rules.

Our work has shown the Trade-tradeoffs between deductive and inductive learning. Both have a place in the construction of a neural network designed to perform a complex rule-based task such as parsing.

REFERENCES

Berwick, Robert C. (1985). *The Acquisition of Syntactic Knowledge*, Cambridge: MIT Press.

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

Gallant, Stephen I. (1988). Connectionist Expert Systems. *Communications of the ACM 31*, 2, 152-169.

Kwasny, S.C., and Faisal, K.A. (1989). Competition and Learning in a Connectionist Deterministic Parser. *Proceedings of the 11th Annual Conference of Cognitive Science Society*, Ann Arbor, Michigan.

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

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

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

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