Creating intelligen

BY DANA NIGRO

FOR COMMON-SENSE APPLICATIONS, CHIPS MAY NOT BE AS "SMART" AS NEURONS YET —BUT THEY'RE GAINING ON US.

Back in the 1950s and '60s, researchers in the infant field of artificial intelligence first proclaimed that computers would soon be able to think, learn, and create as well as or better than humans—a concept frighteningly embodied by HAL in Arthur Clarke's *2001: A Space Odyssey*.

As the millennium approaches, we have created computers that can recognize handwriting, mimic speech, identify people (to some extent) by their voices and faces, even occasionally beat a chess grandmaster or find a "creative" proof to a long-unsolved mathematical theorem. But we are far from having a computer that appears as intelligent, or as "human," as those in sci-fi movies.

Despite the early optimism, creating artificial intelligence has turned out be a profoundly challenging problem. But a core group of professors and researchers in Cornell's Department of Computer Science is building on what we have learned so far and applying AI to matters far more practical than beating Gary Kasparov at chess—such as planning military missions, making wireless phone networks run more smoothly, and creating intelligent "data-mining" agents that retrieve exactly the information you want.

ce

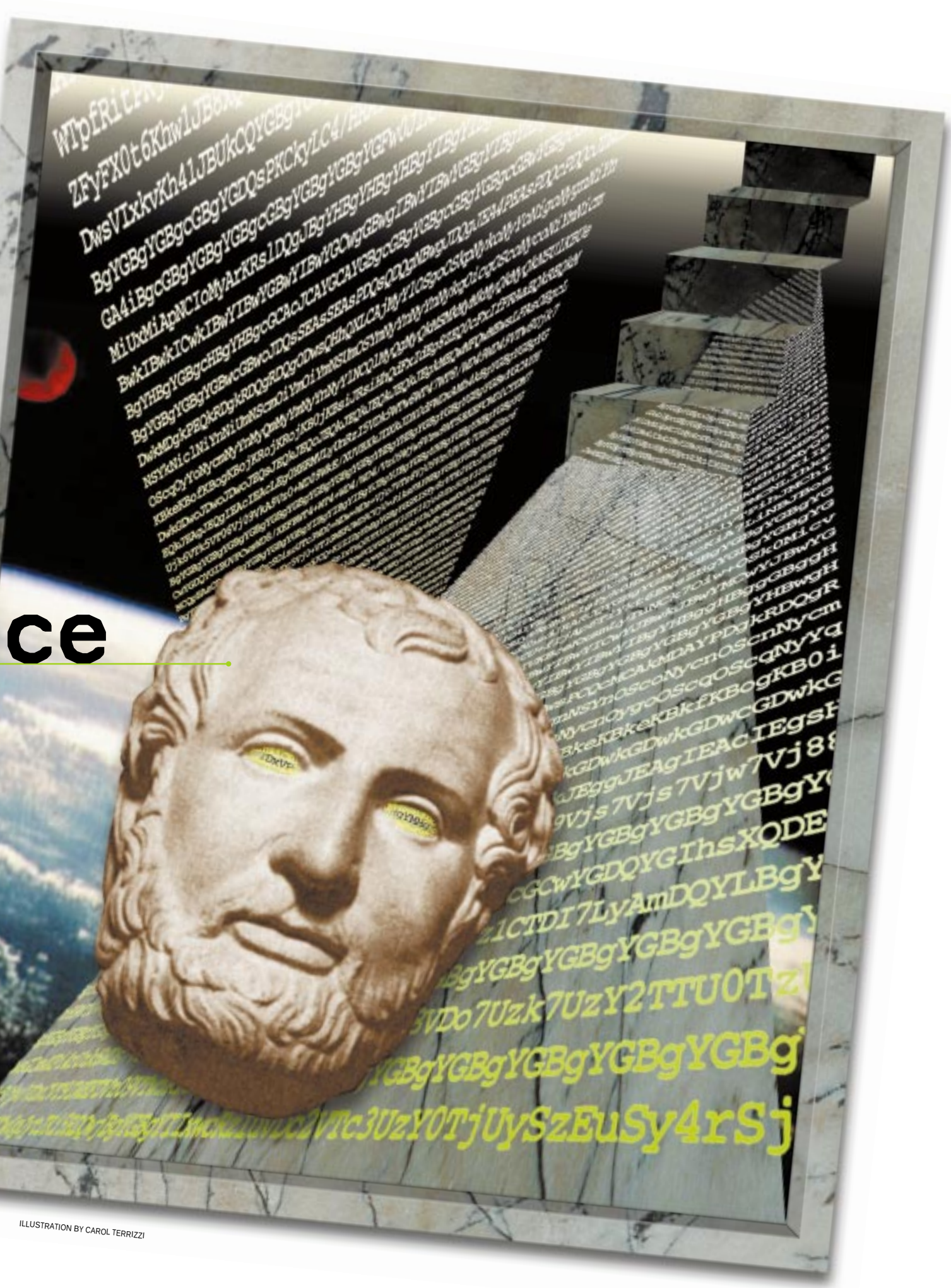


ILLUSTRATION BY CAROL TERRIZZI

These scientists are finding ways for computers to "understand" typewritten text, "learn" from examples instead of being told explicitly how to do every task, "reason" about uncertain situations, and then use their knowledge to make sensible decisions on complicated real-world problems. In addition, they are figuring out why some types of problems are so hard for computers to solve.

What does it mean to be smart? One way to approach AI is to build models of human thinking by, for example, interviewing chess experts on how they play chess. But the computer science department concentrates on developing entirely new ways to exceed human performance on certain tasks, believing machines will never operate in quite the same way as the brain. "Fundamentally different 'hardware' may require fundamentally different algorithms," explains Associate Professor Bart Selman, a key member of the College of Engineering's burgeoning AI program.

What exactly is artificial intelligence? Researchers in the field can debate the nuances of this for hours, but basically it means designing machines that can perform tasks that people do particularly well, including problem-solving, decision-making, and learning. "You're trying to bottle the brain in a computer," quips Selman, who came to Cornell from AT&T Bell Laboratories' AI research department.

One common definition of AI, known as the Turing test, gives a person 30 minutes to type into a computer terminal, ask all kinds of questions, and receive answers. At the end of that time, if the person can't determine whether the respondent at the other end of the line is a computer or a human, then that machine has demonstrated intelligence.

To pass the Turing test, a computer would need at least the following four things, which make up the major fields of study in AI: communication skills (natural language processing), ways of storing information in a format that the machine can use (knowledge representation), the ability to use that information to answer questions and draw new conclusions (automated reasoning), and the ability to adapt to new circumstances and detect patterns (machine learning).

No computer has passed the Turing test yet. So far, programs can only function in specific domains and under the conditions for which they are written. A backgammon-playing computer may be able to repeatedly beat world-class players, but it can't steer a spacecraft or tell you what to adjust in your office software to get tomorrow's presentation to look the way your boss wants.

Where do today's computers stand? If you compare speed and capacity, says Selman, a 1999 home computer has the brainpower of a spider. IBM's Deep Blue is roughly the equivalent of a mouse (if a rodent could play chess and do nothing else). And a modern super-computer still rates slightly below a monkey. While computers are faster than human brains, they still have fewer transistors than the brain has neurons and are structured quite differently.

It's too much! The problem with problems Among the challenges AI researchers still face is how to give a machine an understanding of subject matter and content. "It still needs knowledge of the real world," says Selman. "How do you get common sense into a computer?"

In the real world, many problems have multiple choices that lead to more and more choices. These "combinatorial" problems can often be represented to the computer in Boolean logic, in which true or false variables are linked by ANDs, ORs, and NOTs.

Combinatorial problems are hard to solve by trial-and-error examination of all the possibilities, because the search base grows exponentially with the size of the problem. For example, in a simple 4X4 crossword puzzle, there are 26^{16} letters to consider (26 possible letters for each of 16 squares). That's more than 43 trillion billion possibilities! And *The New York Times* puzzle might require something like 26^{196} possible letter combinations.

Despite their ability to handle giant numbers, computers still get stuck on certain problems, and Selman has taken on the task of figuring out why. "I study what makes a problem hard and what makes it easy," he says.

On a very basic level, Selman has figured out that certain types of Boolean problems cross a size threshold at which they suddenly go from very easy to impossibly hard. Problems in which the ratio of constraints to variables is small can be solved quickly, and problems with large clause-to-variable ratios can't be solved, but you learn that quickly. "In the middle, it's difficult," says Selman. It's at the transition point from easy to hard that the computer racks up long "run times" trying to solve the problems.

To speed up the task, the computer needs ways of narrowing down the possibilities. So Selman develops algorithms that can handle thousands of variables and, with his colleagues, studies which run fastest.

War Games . . . and other planning problems Upstairs from Selman in Upson Hall, research associate Carla Gomes pulls up what looks like a game—a big square filled with smaller squares, some of which have been assigned colors—on her computer screen. The goal is to fill all the rest of the little squares with a set of colors so that there is only one of each color in each row and column. Gomes spends a lot of time with these puzzles, only she's not playing them the way computer Solitaire addicts secretly play when they're supposed to be working. In fact, the U.S. Air Force has given her funding to do just this sort of thing.

Gomes studies how fast the computer can solve the puzzles using different algorithms, if it can solve them

at all. "Mathematically there are so many possibilities, and it's hard for the computer to distinguish the bad from the good options," Gomes explains. "Humans can automatically discard tons of possibilities."

While it's not of much use to anyone if a computer can fill a square with different colors, these problems mimic the complexity in real-world combinatorial problems. The successful algorithms can move from square patterns to complicated, large-scale scheduling and planning decisions—say, figuring out the most efficient movement of military troops and cargo for a mission. Gomes's work extends beyond the military; it could just as easily be used for designing cellular phone networks or setting up machines in factories to meet a company's production requirements.

PHOTOGRAPH BY CHARLES HARRINGTON / UP

AI team members from left to right: Assistant Professor Lillian Lee, Associate Professor Bart Selman, Assistant Professor Claire Cardie, Research Associate Carla Gomes, and Professor Joseph Halpern.

To speed up work on these planning problems, Gomes could narrow down the possibilities by telling the computer to solve the squares with the fewest options left. But interestingly, for many types of problems, adding randomness to the algorithm actually helps provide better, quicker solutions.

"When we add randomness, the computer can solve it in a short time or it can get lost," says Gomes. "It's like a person in a garden maze. You can get lucky and get out in only a few steps or you can make lots of wrong turns."

She runs another puzzle to illustrate randomized backtracking—in which the computer picks a random starting point, assigns a value to that variable, and follows the possible paths from there. If it reaches a point where none of the possible values for a certain variable work, it must backtrack to the previous variable and give it a new value, repeating this process until it reaches an answer. In this case, the computer solves the pattern in two backtracks, a matter of seconds. Gomes runs it again and this time the computer takes a minute, but does not find an answer before it exceeds the maximum number of backtracks that she has set to minimize time delays.

Gomes and Selman closely study these types of problems, known as "heavy-tails" because if you graph the possible combinations versus the time they take to compute, they don't resemble a typical bell curve with a high middle and long thin tails. Instead, the extremes are more common (thus the tails on the graph are fatter)—there are a lot of starting points that lead nowhere and there are a lot of cases where, if certain values are set, they determine all the other variables and the problem can be solved quickly.

"If the search is taking a long time, you can just abort and do it again and hopefully it will find a solution in a shorter run the next time," says Gomes. "The computer could get stuck for days otherwise."

By setting up automatic restarting after a certain number of backtracks, the total time to solve problems ultimately ends up being less than for other algorithms. The trick, which Gomes and Selman continue to work on, is to find out in advance which problems will have "heavy tails" and also to fine-tune the best number of backtracks to use as a cutoff.

"The benefit to introducing randomness to a program, which is counterintuitive, is that it makes the program more robust," concludes Gomes. "If we don't use randomness, we run the risk of the result always being the same and the decision being one of the bad ones."

She demonstrates a World War II problem. The computer must assign planes from three bases to patrol the ocean in squadrons of varying sizes. After analyzing the cost (in this case, the distance the planes must fly), it turns out the computer's first solution is not a good one: Planes from one base are serving a far-away area that is much closer to another base. Running the program again provides a much better solution in which all the sectors are efficiently covered by squadrons relatively close to their own bases. All this was done in a matter of minutes, not the hours it would have taken a human to figure it out. Pretty "smart."

The "What if?" worries Dealing with problems that have a clearly defined set of variables and a concrete mathematical solution is hard enough. But what happens when you bring all the unpredictability of real life into a situation?

"If you knew the way the world was going to be, you could make decisions easily," says Professor Joseph Halpern. "If you knew the winner in a horse race, you wouldn't have a problem. But probability is involved."

Among the things Halpern works on is how to model uncertainty in AI problems, or in other words, how to represent what the computer does and does not know at any given time and the likelihood of something unknown being or becoming true. Using different models, Halpern then looks at how to get a system to best reason about that uncertainty in making a decision.

Uncertainty can be represented quantitatively using probability theory, but that's only part of the equation. Most real-life problems don't fit neatly into the mathematical approaches that easily solve puzzles involving coin tosses and card games.

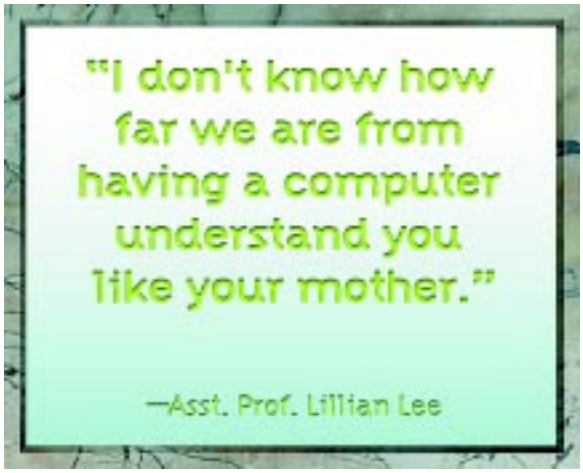
"Let's say you're supposed to meet someone at 5:00, and they haven't shown up at 5:15," Halpern postulates. "Do you stick around or leave? It depends. How much did you want to meet that person in first place? That's the utility part—how much do you care? If it's someone you've been getting together with pretty often, how likely is it that they are going to get there soon?" He adds, "These things arise all the time in systems applications."

Some approaches to uncertainty and decision theory look at things—the likelihood of something happening, the utility, and the cost of waiting versus quitting and restarting the transaction—in a more qualitative manner.

"We don't have nice numbers in practice," says Halpern. In the case above, determining probability is not as simple as figuring out the likelihood of drawing an ace out of deck if you're only dealt six cards. You're making guesses based on the person's past behavior, where they're coming from, and how important it is that they be there on time. And you're determining how much it matters by whether you're meeting a blind date versus an old friend or a business colleague.

"If you don't know the probabilities and people aren't prepared to say exactly what the utility is, how do you do qualitative decision theory efficiently and effectively?" asks Halpern.

This is among the questions that he spends much of his time trying to answer, but he isn't entirely wrapped up in theory. Using his models, Halpern works with other members of the engineering school to build more effective systems, including a project on wireless telephone networks.



If you have used a cellular phone in your car, you have probably had your call dropped as you drove along or haven't able to place a call from a certain area. The problem is that your phone company has a limited amount of bandwidth and can only accept a set number of calls at a time.

"If you are a cell phone company, you have to decide whether to accept a call or block it," explains Halpern. "Even if you have the lines to accept a call, it's not obvious that you should. If there's only one line left, should you accept? The world is divided up into different cells that handle a call as someone is driving along. If I use up all the lines in a cell, and someone already on a call drives over into that cell, I have to drop the call."

Before allocating the phone lines, a system has to consider how likely it is that another call will come over and what the cost is of dropping a call versus blocking it. "People get madder if you drop their call than if you block it," comments Halpern. "Plus, not all calls are equal. If your call is an FTP file transfer, you don't mind; the phone will redial and pick up where it left off. If you are having a serious conversation, you mind a lot."

Halpern has applied his decision theory work to fields as diverse as air traffic control and database query planning. "People have always thought intuitively about it, but it pays to formalize it more carefully," Halpern concludes. "By doing that, you get systems that adjust themselves automatically and are more user-friendly."

Please don't let me be misunderstood More efficient and user-friendly information-finding systems are also the ultimate goal of Assistant Professor Claire Cardie's research. As anyone who has ever looked for information on the Internet knows, using a search engine to scan hundreds of thousands of documents and web sites for key words can be incredibly frustrating. Frequently, you wonder why the search engine retrieved most of the items since half the information isn't even remotely relevant. Wouldn't it be nice if the computer could actually understand what it is searching and provide only relevant details neatly summarized for us?

Cardie, whose work combines natural language processing and machine learning, is trying to develop a system that understands typewritten text. "What 'understand' means depends on your goals," she comments.

In her case, Cardie focuses on information extraction. A natural language processing system is given large amounts of text—such as news articles from a wire service—and searches through it to find relevant information on a specific subject, like natural disasters. The system then creates simple summaries of the information, listing user-defined

criteria such as date, time, location, damage, injuries, and total cost. The summaries could be automatically incorporated into databases and used to answer questions from scientists, journalists, and other researchers.

Among the tasks the computer must first accomplish are assigning "part of speech tags" to the words in the text and analyzing the sentence structure. Then the system must identify relevant words and their relations to each other ("hurricane" and "hit Florida" and "damaged 14 homes"). While the basic ability to diagram a sentence can be programmed into a computer, the machine doesn't understand the meaning of the words. This is where the work of Cardie's colleague, Assistant Professor Lillian Lee, ties in.

Lee points out that there's lots of ambiguity in what seem like simple sentences, giving the example: "I saw her duck with the telescope." Is "duck" a noun or a verb? Who has the telescope—the speaker, the woman, or the duck? And to whom does "her" refer? These sorts of questions can easily trip up a computer.

By taking statistical information about language usage from large amounts of text, Lee hopes to be able to automatically find patterns in the data. The information-extraction programs could then rely on these patterns to solve ambiguities or correct errors in the text (for example, recognizing a typo by checking which is more likely to occur: "eat a peach" or "eat a beach"). But there's a problem—it's hard to get reliable estimates of the probability of certain events or strings of words occurring. "In daily life, you are saying new things every day and new events occur all the time," she says. "It's called the sparse data problem—there's a lot of data, but it doesn't tell you about new events.

"How do people deal with new events? By analogy," Lee continues. "We use similar events to predict the behavior of novel events." So she is evaluating various ways of determining the similarities of words by grouping them into classes. This helps address another challenge information-extraction programs must face: figuring out whether a word is a new entity or a synonym for something it has already identified in the text ("tornado" and "twister" or "Ford" and "Big Three auto manufacturer").

Natural-language processing systems that can read and summarize text already exist in very specific fields and can do tasks such as helping to analyze life insurance applications, summarizing medical patient records, and classifying legal documents. These systems rely heavily on large amounts of domain-specific knowledge that must be hand-coded and requires the expertise of specialists in the field and computational linguists. "It can take a year to design one for a new domain," says Cardie.

She wants to build these systems automatically by using machine learning techniques. Instead of writing a program that tells the system exactly how to do each task, you provide examples of how to do that task. "If you give the computer the diagrams for 1,000 sentences, then when you give it a new sentence it has never seen, the machine can diagram the sentence automatically based on the correlations it's seen in the 1,000 'training' sentences," explains Cardie.

Among their benefits, learned approaches perform as well or better on certain tasks than the time-intensive hand-coded approaches, which aren't flexible enough to handle the unique language and range of events that occur in real-world text. By learning from training examples that cover the specific area of knowledge for which it is used, the system increases its accuracy. Yet, since the general language-understanding skills learned are automatic, it would be relatively easy to switch the skill from one domain of knowledge to another with some retraining. Ultimately, these learned-approach systems could be trainable entirely by their end users, without the help of experts.

"We don't know how to do all this yet," acknowledges Cardie, pointing out that researchers are still working to improve the amount of relevant information recalled and the accuracy of the systems. There is still much to be done in figuring out which learning algorithms are best for each language task and which sentences make the best training examples.

"Right now we can get 60 to 65 percent accuracy for information extraction systems that are written by hand," she says. "Automatic learning is under 60 percent, which is still usable." (To put this into context, if two people summarize the same block of text, they agree only 80 percent of the time.) But certain areas of language understanding are highly effective, she adds; part-of-speech tags are 95 percent accurate and sentence parsing is about 85 percent. "But when you put it all together, the errors multiply," Cardie explains. "Sentences are complicated. In news articles, there are lots of foreign names and complicated syntax. It's hardest with real text like that."

Despite these difficulties, the successes make Lee enthusiastic about the future of natural language processing and AI. "Many people have said that solving natural language could solve AI," she points out. "Any problem you want to attack is there in language." It's such an important area, she says, that at one point in 1997, Microsoft chairman Bill Gates declared he was betting the company on natural language processing.

She points out that NLP programs can tackle automated translation and solve questions about authorship of historical documents; there are already products on the market that allow you to talk to your computer. But she says with a laugh, "I don't know how far we are from having a computer that understands you like your mother."

Maybe the computer won't understand you as well as your mom does, but with all the work the Cornell researchers have done in the areas of planning and reasoning, it may give better advice.