The Science of Language: COMPUTATIONAL

by Jason Eisner

If you love language, a career in computer science might not spring to mind as the best way to pursue that interest. But, as this young professor discovered, teaching computers to learn language not only presents challenges you won't find in your English classroom, but also provides a whole new way to appreciate the complexity of language.

In sixth grade I felt that I was a pretty good programmer, having gotten our classroom TRS-80 to play a game of craps. It was a companionable machine, with a comfortable 4K of memory, and for my next project I decided to program it to converse with me. Twenty years later, I'm still working on it—having learned a lot more about computer science, human language, and mathematics along the way.

The Turing Test

Back in 1950, the mathematician Alan Turing, one of the great founders of computer science, famously proposed conversation as the test of artificial intelligence (AI). He argued that if a computer could successfully pass as a human for 30 seconds of typed conversation, we might as well call it intelligent. Like me, he underestimated the problem, predicting it would be solved by 2000. I'm secretly glad it turned out to be hard enough to give me a lifetime of interesting work!

Turing was picturing college-level conversations about Shakespeare and chess puzzles. He didn't realize that even simple language requires a daunting amount of knowledge and reasoning ability. How would a computer figure out that "they" refers to different groups of people in the following two sentences?

The city council denied the marchers a permit because they feared violence.

The city council denied the marchers a permit because they advocated violence.

Some computational linguists already try to build "deep" language systems that can do this. Such systems "understand" enough about a specialized topic, like city permits or travel planning, that they can not only figure out what you mean, but also draw appropriate conclusions that let them help you out.

What Computers Need to Know about Language

Many of us are more focused on understanding the special properties not of city permits, but of language itself. A major concern of AI is to get computers to **represent**, use, and acquire facts. So the big questions are: Exactly what facts do you subconsciously know about your language? How do you make use of them? And how did you figure them out as a toddler, just by listening to other people make sound vibrations in the air?

The language facts I'm talking about (you know millions) include where the verb goes in a sentence, which words in the dictionary are related, which sounds can get dropped in fast speech, what the present perfect tense ("have eaten") means logically, and when it is appropriate to use "herself" rather than "her." These are all very hard questions to answer fully (try it), and it's amazing that you tackled them at age four!

LINGUISTICS

So how should a computer represent, use, and acquire such facts? To **represent** the facts of a given language, we would prefer to use a single scheme that is flexible enough to deal with any human language—just as a baby's brain is flexible enough to learn English, Korean, Sanskrit, Warlpiri, or Zulu. To devise such schemes (an ongoing job!), we draw on the work of linguists who study all those languages. For our purposes the schemes have to be quite formal, and they often use some discrete mathematics, like graph theory and abstract algebra.

Using the facts of a language to solve a problem requires us to design algorithms. Different problems call for different algorithms. We work, for example, on algorithms to take spoken dictation, diagram sentences, translate from one language to another, summarize email, figure out whether you confused "their" with "they're" or "there," read aloud with natural intonation, and convert news articles into databases of facts.

Learning a language automatically is like doing automatic scientific discovery. We can give a powerful computer a lot of data—for example, two million sentences from the *Wall Street Journal*— and ask it to search for hypotheses that explain the data well but also conform well to our general scheme for describing human languages.

Learning is Ambiguous

To get a sense of the problems, suppose I tell you that "Time flies like an arrow," a sentence you hadn't heard before.

If you don't speak English, you're up a creek: so far as you know, I could have meant anything at all.

If you know how English does and doesn't permit ideas to be expressed, you will be able to narrow my meaning down, but not completely! Maybe I was making a metaphorical remark about time, or a literal one. Maybe I was instructing you as to how you should time the flies, or which flies to time. Maybe I was telling you about a fly species called "time flies" that are fond of a particular arrow, or arrows in general. (Groucho Marx quipped that time flies like an arrow, but fruit flies like a banana.) Or maybe what I actually said was "Tie 'em flies, like in our row." All of these interpretations are possible in principle, and an appropriately programmed computer can come up with them.

If you also know what happens frequently in English, you can rule out most of these interpretations as possible but unlikely. Most sentences do not start with verbs, "time" is usually an abstract noun, and "like"is usually a preposition. Furthermore, you've heard people say "The time sure flew past tonight," but never "Look at those time flies." All these little bits of evidence add up somehow to nudge you toward reading the sentence as a remark about time. Probability and statistics turn out to be extremely useful tools in getting computers to add up such evidence.

A wonderful introduction to such issues is Steven Pinker's *The Language Instinct* (1994). It focuses more on linguistics and psycholinguistics than on computational linguistics, but it should get your mind churning.

The Best of All Worlds

What I love most about my field is that it engages every part of my mind. For me, it was a way to get the benefits of majoring in writing, empirical science, math, computer science, and engineering all at once, without losing focus. As a researcher, I like to keep crossing that spectrum: noticing nuances of language, trying to fit them into patterns, turning those patterns into rigorous mathematics, developing algorithms to work with the mathematics, coding those algorithms as programs, and studying how well the programs do.

The social aspect of research is also fun for me—collaborating with colleagues, giving talks at conferences, and participating in my research community by reviewing other people's papers and organizing workshops. In addition, since I chose to work at a university rather than an industrial lab, I help train the next generation of computational linguists and also get to teach other topics in computer science. It's a good full life, and it sure beats a lifetime of playing craps with a 4K TRS-80.

*A new assistant professor, **Jason Eisner** took his first computer science class at CTY in 1982, the first year that class was offered. He is one of several faculty at the University of Rochester who study how humans process language and how to get computers to do the same. He's currently developing a statistical approach to learning grammatical transformations (the basis of Noam Chomsky's linguistics revolution in 1957). He's also been bringing computer science techniques to a current revolution in phonology (the study of how languages combine sounds).*