Jason M. Eisner

Research Summary (November 2012)

Work may be browsed by topic at http://cs.jhu.edu/~jason/papers.

1 Research Area

My research program aims to develop fundamental methods for natural language processing (NLP) and computational linguistics (CL). The goal is to recover the linguistic structure and meaning of words, sentences, documents, or whole languages. I regard these as problems of statistical inference.

Inference requires a model: I work to enable more sophisticated modeling. Why is this important? Because difficult NLP tasks will require increasingly nuanced attention to the linguistic phenomena. Ultimately we want broadly competent NLP/CL systems that can consider many aspects of language at once.

Beyond designing *specific* probabilistic models of linguistic phenomena, I often develop *general* modeling techniques—along with new inference and learning algorithms and software frameworks to make such models practical, since complex models present computational and implementational difficulties.

I have been lucky to have strong students as collaborators on many of my papers. Our work ranges across a wide variety of NLP applications, and considers a wide variety of learning settings. Typically these papers include significant empirical validation, which has often raised the state of the art.

The engineering motivation: Computers must learn to understand human language. A huge portion of human communication, thought, and culture now passes through computers. Ultimately, we want our devices to help us by understanding text and speech as a human would—both at the small scale of intelligent user interfaces and at the large scale of the entire multilingual Internet.

The scientific motivation: Human language is fascinatingly complex and ambiguous. Yet babies are born with the incredible ability to discover the structure of the language around them. Soon they are able to rapidly comprehend and produce that language and relate it to events and concepts in the world. Figuring out how is a grand challenge for both cognitive science and machine learning.

The computer science motivation: NLP problems are excellent examples of more general problems in machine learning, combinatorial optimization, declarative algorithms, and data-intensive computing. I have often taken a broader perspective and addressed the problems in this more general form.

2 General Methods

2.1 Structured Prediction Algorithms for Complex Models

"Structured prediction" is the problem of modeling unknown variables that are themselves complex structures. The machine learning community has focused on modeling vectors (via graphical models). But linguistics must predict unbounded structures such as *strings*, *trees*, and *knowledge bases*.

I introduced a number of key dynamic programming algorithms for parsing and machine translation (see section 3.1). The dynamic programming assumptions can break if we enrich our linguistic models—but rather than throw out these core algorithms and start over, I have shown in a series of papers how they can be *embedded as efficient subroutines* within other exact or approximate methods such as

- belief propagation
- dual decomposition
- row generation
- stochastic local search

Dynamic programming allows these methods to work with exponentially or infinitely large variable domains (e.g., *string-valued* random variables), high-degree factors and constraints that are made tractable by their special structure, and exponentially large local neighborhoods. This is a powerful and reusable idea that has inspired others. E.g., my 2008 paper "Dependency parsing by belief propagation" apparently triggered the recent flurry of work on dual decomposition in NLP (David Sontag, p.c.) at MIT, Columbia, and CMU, as well as other "structured" message-passing algorithms at places like Berkeley and UMass. I am actively working on further accuracy and efficiency improvements.

2.2 Training Better Structured Predictors

Since I think the way of the future is to build rich, faithful models, it is a serious problem that better modeling may lead to to worse predictions under approximate inference (or astronomically slow predictions under exact inference). In several recent papers, I have been working on improving the accuracy and speed of structured prediction, by training a runtime policy (sequential decision rule) to achieve good performance on a given or estimated data distribution. This is the "right" approach under Bayesian decision theory. But since the policy may take many steps to make a prediction and obtain a delayed reward, training becomes a type of reinforcement learning problem.

To design our family of runtime policies, we generally use the rich model and adapt some standard approximate inference algorithm (such as loopy belief propagation, best-first or pruned dynamic programming, agglomerative clustering, or a combinatorial algorithm). However, by explicitly searching not for MAP parameters, but for parameters and other policy choices that minimize loss $+ \lambda \cdot$ runtime, we find that we obtain good inference while compensating for the errors of model mismatch and the base approximate inference algorithm. This gives often striking accuracy gains. We have also shown some ability to speed up with little or no loss in accuracy.

Many important questions remain to be answered, including how to construct good policy families; how to improve policy search within the family; and how to estimate a policy's test-time risk using imputed data if the observed data are incomplete, sparse, or non-IID (e.g., relational learning may train on a single huge partially observed graph).

2.3 Algorithms for Finite-State Machines and Hypergraphs

Weighted finite-state machines are widely applicable in NLP and speech recognition, to represent certain probability distributions over strings or over pairs of (monotonically aligned) strings. Weighted hypergraphs can represent more general probability distributions over strings, trees, or pairs thereof, including parse forests and translation forests. My algorithmic contributions include the fundamental, widely used algorithms for the common problems of

- training the parameters of such distributions
- computing various first-order and higher-order gradients and expectations of such distributions

as well as more specialized methods for

- finding the most probable string from such a distribution[1] (our approximate "variational decoding" algorithm is used by MT researchers to choose the best translation from a hypergraph)
- minimizing a semiring-weighted DFA
- restricting a transducer so that each input maps only to its *best* output (in a certain sense)
- unsupervised learning when the training dataset is given as a finite-state distribution over strings rather than a list of distinct strings

I have also proposed models consisting of *many* finite-state machines, which serve as the weighted constraints in a constraint system (e.g., a Markov Random Field over string-valued variables). Such models are powerful but computationally challenging. My papers so far on such models cover

- formalization and undecidability properties
- approximate joint inference by belief propagation (see section 2.1)
- exact joint inference by dual decomposition (see section 2.1)
- applications to modeling morphological paradigms (see section 3.2) and to consensus decoding

2.4 The Dyna Language

As NLP models and inference procedures have gotten fancier, they have also gotten more difficult to implement. A decent system often requires an investment of tens of thousands of lines of code (or more), across hundreds of files. This is a serious obstacle to progress in our field. It imposes barriers to entry, barriers to experimentation, and barriers to education. Many good ideas are simply not tried or not combined—it is too hard to play around.

To remove this obstacle, I have been developing a declarative programming language, Dyna, as a unifying framework for concisely specifying data and algorithms. The storage and computation strategies are left up to the system.

An early version of Dyna (the 2005 compiler) was targeted only at semiring-weighted dynamic programming. This version was used at the time in 15+ NLP research papers as well as a class at JHU, and it is currently used for teaching "NLP Algorithms" in CMU's class of that name. But in the past few years, the language design has evolved to become substantially more powerful. We show its broad applicability through a series of examples—remarkably short and clean descriptions of various AI computations.

Dyna is a pure language without I/O. A Dyna program simply specifies a collection of named data items. One can externally query this "dynabase" to look up all items whose structured names match

[1]This would count as a fundamental algorithm for a common problem if it were exact, but the problem is is NP-hard.

a specified pattern. The twist is that some items in the dynabase are defined by rule from other items, perhaps in other dynabases, and perhaps recursively. Thus Dyna can specify both data and algorithms. Furthermore, the algorithms are dynamic (reactive), since externally updating the value of one item may affect the values of many derived items. The rules are easy to write because they exploit the structured item names (cf. terms in pure Prolog, which is a subset of Dyna).

It is the system's job to choose efficient data structures and algorithms to support the possible queries and updates. We are collaborating on flexible, adaptive runtime architectures with my colleague Prof. Yanif Ahmad, who is building a declarative infrastructure ("K3") for managing large-scale data and computations.

2.5 Parameter Search

Many machine learning methods call for minimizing some non-convex objective function, such as risk (see section 2.2). I have shown how to use algorithmic differentiation (back-propagation) to compute the gradients and Hessian-vector products within general families of inference procedures, and have demonstrated that on the NLP problems that I have tried, deterministic annealing can often mitigate the non-convexity. I have also proposed new approaches such as contrastive estimation (a faster and sometimes more appropriate objective function) and structural annealing. All of these techniques have become part of the repertoire of the NLP field.

One way to make learning easier is to get better annotation. I proposed using human annotators more effectively by having them highlight portions of the input ("rationales") that support their judgment. The learning algorithm is then enhanced to use this additional information. This method has been adopted by others in NLP and in computer vision. Unlike other recent approaches that try to get more value from annotators, it does not require annotators to understand the features used internally by the system.

At the other end of the spectrum, I have developed new techniques for semi-supervised learning, notably strapping ("bootstrapping without the boot").

3 Application-Specific Methods

I have worked on diverse NLP applications because they present interesting challenges and opportunities. With my students and collaborators, I have built systems for parsing, machine translation, alignment, grammar induction, part-of-speech tagging, morphology, phonology, name clustering, word sense disambiguation, selectional preference, information extraction, coreference resolution, sentiment analysis, text categorization, document clustering, topic modeling, text entry, text compression, and cryptanalysis. These papers are mainly statistical systems, and show empirical improvements against previous systems or baselines, sometimes setting a new high-water mark on their tasks.[2]

I have done the most sustained work on the problems discussed below. (I also have over a dozen papers on machine translation, but they do not form a single research program.)

[2]E.g., our recent papers (2012) on coreference resolution and on various kinds of CRFs are the current state of the art.

3.1 Parsing, Syntax, Synchronous Syntax, and Grammar Induction

Exact parsing algorithms that I developed for the following grammar formalisms are quite widely used in the field:

- projective dependency grammars
- combinatory categorial grammars
- lexicalized context-free grammars
- lexicalized tree-adjoining grammars

They are regarded as the fundamental exact algorithms for those formalisms. In addition, my linear-time partial parsing scheme ("vine parsing") is used within Google's parsing engines. I've also showed generalizable formal transformations that can automatically derive some parsing algorithms from one another.

A parser is only as accurate as its linguistic model. I have shown how to enrich syntactic models so that they consider lexicalization, dependency length, non-local statistical interactions ("non-local" = beyond what the grammar formalism provides), or syntactic transformations that generate new rules.

Finally, I have generalized synchronous models in various ways so that they no longer assume that a sentence and its translation have strictly isomorphic syntactic structure. These methods include synchronous tree-substitution grammars, quasi-synchronous models, and models for reordering input words prior to translation. These methods have had good uptake by the MT community. In addition, other researchers have applied quasi-synchronous models to non-MT problems such as modeling (question, answer) pairs, entailment, paraphrase, and text simplification.

I have long been interested in grammar induction for natural language, and have published several moderately successful attacks on the problem, but it remains essentially unsolved. I am currently trying some new attacks, using ideas from deep learning, spectral learning, and operations research.

3.2 Morphology

In a series of papers, I have been building up a comprehensive model of morphology. So far, we can learn how the words of a language are inflected, given a plain text corpus plus a small supervised set of known inflectional paradigms. The model is principled and generates each "level" of linguistic description in turn:

1. The **grammar** and subregularities of the language (via many appropriately structured finite-state transducers coordinated in a Markov Random Field—section 2.1).
2. The infinite inventory of **types** and their inflectional paradigms (via a Dirichlet process mixture model based on the above grammar).
3. The corpus of **tokens** (by sampling inflected words from the above inventory).

There are many potential extensions to the model and the inference algorithm. Our current inference algorithm cleanly integrates several techniques that handle the different levels of the model: classical dynamic programming operations on the finite-state transducers, loopy belief propagation in the Markov Random Field, and MCMC and MCEM for the non-parametric DP mixture model.

I have also begun working on hierarchical nonparametric Bayesian models of semantics and information extraction, with a level-by-level generative story that is in the same spirit as the above. A network of concept types and facts is generated along with linguistic means for expressing these facts. Inference must then recover all of this from observed linguistic data.

3.3 Linguistic Transformations

Some of my papers make use of "transformation processes," in which one linguistic form mutates into another. One application is forms that get visibly modified. Recently I have been working on variant forms of person names: `Hillary Rodham Clinton` may be referred to in many ways because of typographic errors, nicknaming, transliteration, initials, honorifics, etc. Moreover, a mutant form like "Mrs. Hilary R. Clintom" may mutate again. We find that at all levels of supervision, reconstructing a latent *phylogeny* of name mentions is better able to recover a mutation model, and to determine coreference, than a simpler model that relates each variant individually to the canonical form.

However, I originally proposed transformation processes simply to model correlations among the probabilities of "related" events. Words that are used as active verbs can probably be used passively as well (syntactic subcategorization). Words that take `beer` as a direct object can probably take `wine` as well (selectional preference).

As Chomsky did with transformational grammar, we can invoke transformations to explain why $p(x_i)$ and $p(y_i)$ (e.g., rates of active and passive uses of word i, or rates of `beer` and `wine` as its object) are correlated across values of i. One explanation is that some fraction of x_i tokens always transform into y_i tokens before they are observed.[3] This suggests that the same correlation should hold even for sparsely observed i, and that gives us a smoothing effect in the posterior. Learning the Markov chain of likely transformations among events is analogous to learning the kernel of a Gaussian process, because both recover a notion of which events are correlated. However, the details are different (e.g., transformations are asymmetric while kernels are symmetric).

3.4 Phonology

In earlier days, I did foundational work in computational Optimality Theory for phonology. This involved substantive engagement with the linguistics (e.g., studying hundreds of OT sub-grammars then being proposed by others). From this study I synthesized concrete formal proposals about Universal Grammar:

- primitive Optimality Theory—a substantive restriction on the form of constraints
- directional constraint evaluation—a new candidate evaluation mechanism with attractive computational properties.

Within this framework I was able to provide

- finite-state algorithms for generation and comprehension
- complexity-theoretic analyses of learning
- a linguistic reanalysis of the typology of metrical stress

[3] Or vice-versa, or some common ancestor transforms into both.