FACTORIE: Efficient Probabilistic Programming for Relational Factor Graphs via Imperative Declarations of Structure, Inference and Learning

Andrew McCallum, Khashayar Rohanemanesh, Michael Wick, Karl Schultz, Sameer Singh
Department of Computer Science, University of Massachusetts Amherst, Amherst, MA 01002
mccallum@cs.umass.edu

Discriminatively trained undirected graphical models, or *conditional random fields* [7], have garnered tremendous interest and empirical success in natural language processing, computer vision, bioinformatics and many other areas [18, 1, 13]. Some of these models use simple structure (*e.g.* linear-chains, grids, fully-connected affinity graphs), but there has been increasing interest in more complex relational structure—capturing more arbitrary dependencies among sets of variables, in repeated patterns. Reimplementing variant structures from scratch is difficult and error-prone, however, and thus there have been several efforts to provide a high-level language in which new undirected model structures can be specified. These include SQL [19], first-order logic [15], and others such as Csoft [20].

Regarding logic, for many years there has been considerable effort in integrating first-order logic and probability [12, 10, 16, 14, 15]. However, we contend that in this combination, the 'logic' aspect is mostly a red herring. The power of relational factor graphs is in their repeated relational structure and tied parameters. First-order logic is one way to specify this repeated structure although it is not necessarily the best; in fact, its focus on boolean variables, difficulty in representing if-then-else, inability to address graph problems such as reachability, and confusability to humans [3] makes it less than ideal. Logical inference is appealing but is replaced by probabilistic inference in these models, and is thus usually superfluous. In traditional *deterministic* programming Prolog has been largely dropped in favor of imperative programming since programmers realized that (a) they typically need to keep the imperative solver in mind afterall to make tractable solutions, and (b) much human knowledge is naturally expressed procedurally anyway.

We propose an approach to probabilistic programming that preserves the *declarative* statistical semantics of factor graphs, while at the same time leveraging *imperative* constructs to greatly aid both efficiency and natural intuition in specifying model structure, inference and learning. Rather than first-order logic, model authors have access to an object-oriented Turing-complete language when writing their model specification. The point, however, is not merely to be higher in the Chomsky hierarchy; it is ease-of-use and efficiency. (Implementations of logic primitives are also provided.)

Our design goals include: (1) representation of factor graphs, with their ability to express both undirected and generative directed graphical models, (although our emphasis is on undirected); (2) massive scalability, in terms of the observed input data set size, output data configuration space, and the complexity (tree-width) of the factors; scalability should not rely on parallelism, but should be able to take advantage of it when available; (3) efficient, built-in support for discriminative training—in particular incremental, online learning; (4) a natural, easy-to-use avenue for procedural knowledge to be expressed and integrated with declarative knowledge; our methods of injecting imperative/procedural knowledge are indicated below in italics.

We achieve these goals with a system we call FACTORIE (loosely named for "Factor graphs, Imperative, Extensible"), a software library for an object-oriented, strongly-typed, functional programming language called *Scala* [11]. (Our ideas could be implemented in any of a number of programming languages; Scala is an up-and-coming JVM language with many powerful features.) By providing a library and direct access to a full programming language (as opposed to our own, new "little language"), the model authors have familiar and extensive resources for implementing the procedural aspects of the design, as well as the ability to beneficially mix data pre-processing, evaluation, and other bookkeeping code in the same file as the probabilistic model specification.

For inference, we rely on MCMC to achieve efficiency with models that not only have large tree-width, but an exponentially-sized unrolled network, as is common with complex relational data [5, 15, 9]. The key is to avoid unrolling the network over multiple hypotheses, and to represent only one variable-value configuration at a time. As in BLOG [9], MCMC steps can adjust model structure as necessary, and with each step the library automatically builds a *DiffList*—a compact object containing the variables changed by the step, as well as undo and redo capabilities. Calculating the factor graph's 'score' for a step only requires DiffList variables, their factors, and neighboring variables. In fact, unlike BLOG and BLAISE [2], we build inference and learning entirely on DiffList scores and never need to score the entire model, (which enables efficient reasoning about observed data larger than memory, or models in which the number of factors is a high-degree polynomial of the number of variables).

Model Structure. Factors, their templates and variables (binary, categorical, ordinal, real, etc) are typed objects in our object-oriented system, enabling beneficial encapsulation, abstraction, inheritance and composition, and also permiting new user-defined variable types, such as set-valued variables. At the heart of model structure definition is the pattern of connectivity between variables and factors, and the DiffList should have extremely efficient access to this. Unlike BLOG, which uses a complex, highly-indexed data structure that must be updated during inference, we specify this connectivity imperatively: factor template objects have *methods* that find the factor's other variable arguments given a single variable from the DiffList. Our programmer's interface for this *imperative structure definition* is extremely compact. This approach also efficiently supports model structure that varies conditioned on input and output [8] variable values. Variables' value-assignment methods can be overriden to automatically change other variable values in coordination with the first assignment—an often-desirable encapsulation of domain knowledge we term *imperative variable coordination*. For example, in response to a named entity word label change, a coreference mention can have its string value automatically adjusted, rather than relying on MCMC inference to stumble upon this self-evident coordination. Additionally, in a somewhat unconventional use of functional mapping, we support separation between factor *neighbors* and *sufficient statistics*. Neighbors are variables touching the factor whose changes imply that the factor needs to be re-scored. Sufficient statistics are the minimal set of variable values that determine the score contribution of the factor. These are usually the same; however, by allowing a user-defined function to perform the mapping, we provide an extremely powerful yet simple way to allow model designers to represent their data in ways most natural to the program, and then seperately concern themselves with how to score them. For example, the two neighbors of a skip-edge factor [17] may each have cardinality equal to the number of named entities, N, but we may only care to have the skip-edge factor enforce whether or not they match, mapping N^2 down to 2. We term this *imperative variable-sufficient mapping*.

Inference. A key component of many MCMC inference procedures is the proposal distribution, or "jump function," which is *imperative* by nature. Of course this is also a natural place for injecting prior knowledge about the coordination of variable values and various structural changes. In fact, in some cases we can avoid expensive deterministic factors altogether with *property-preserving* jump functions [4]. For example, coreference transitivity can be efficiently enforced by proper initialization and a transitivity-preserving jump function; projectivity in dependency parsers can be enforced similarly. We term this *imperative constraint preservation*.

Learning. Obtaining the gradient for traditional maximum likelihood (ML) parameter estimation requires inference of factor marginals, which can be extremely expensive in complex models. Various online learning methods, such as perceptron, avoid the need for marginals, but still require full decoding, which can also be expensive inside the inner loop of learning. We avoid both of these expenses by *sample-rank* [4], which approximates the parameter gradient incrementally from individual MCMC steps: for each jump, we obtain the difference between the model scores of the original and proposed configurations (efficiently through the DiffList). We also obtain the difference in 'true scores'—a measure of which is closer to the gold-standard labels. If the model's relative ranking differs from the true relative ranking (or the score difference violates a margin), we perform a perceptron-style update using the sufficient statistics of the DiffList. We have proven convergence of sample-rank, mirroring the perceptron proof subject to marginal separability. Empirically the method is extremely successful; it provides the current state-of-the-art in DARPA ACE coreference, surpassing the previous results by 10% absolute [4]. Note that, unlike SEARN [6], which incrementally builds the structured output and scores incomplete configurations, our approach always scores (differences of) complete configurations—which is important for obtaining distributions over possible worlds, and for models with factors that would be missing arguments in partial configurations.

Acknowledgments

This work was supported in part by the Center for Intelligent Information Retrieval, in part by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under contract number NBCHD030010, and AFRL #FA8750-07-D-0185, in part by The Central Intelligence Agency, the National Security Agency and National Science Foundation under NSF grant #IIS-0326249, in part by Lockheed Martin through prime contract #FA8650-06-C-7605 from the Air Force Office of Scientific Research, and in part by NSF grant #IIS-0803847.

References

[1] A. Bernal, K. Crammer, A. Hatzigeorgiou, and F. Pereira. Global discriminative learning for higher-accuracy computation gene prediction. In *PloS Computational Biology*, 2007.

[2] K. A. Bonawitz. *Composable Probabilistic Inference with Blaise*. PhD thesis, Massachusetts Institute of Technology, 2008.

[3] P. Cheng and K. Holyoak. Pragmatic reasoning schemas. *Cognitive Psychology*, 17:391–416, 1985.

[4] A. Culotta. *Learning and inference in weighted logic with application to natural language processing*. PhD thesis, University of Massachusetts, May 2008.

[5] A. Culotta and A. McCallum. Tractable learning and inference with high-order representations. In *International Conference on Machine Learning Workshop on Open Problems in Statistical Relational Learning*, Pittsburgh, PA, 2006.

[6] H. Daumé III, J. Langford, and D. Marcu. Search-based structured prediction. Technical Note, 2006.

[7] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[8] D. McAllester, M. Collins, and F. Pereira. Case-factor diagrams for structured probabilistic modeling. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 382–391, Arlington, Virginia, United States, 2004. AUAI Press.

[9] B. Milch, B. Marthi, and S. Russell. *BLOG: Relational Modeling with Unknown Objects*. PhD thesis, University of California, Berkeley, 2006.

[10] S. Muggleton and L. DeRaedt. Inductive logic programming theory and methods. In *Journal of Logic Programming*, pages 629–679, 1994.

[11] M. Odersky and al. An overview of the scala programming language. Technical Report IC/2004/64, EPFL Lausanne, Switzerland, 2004.

[12] D. Poole. Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

[13] A. Quottoni, M. Collins, and T. Darrell. Conditional random fields for objet recognition. In *NIPS*, 2004.

[14] L. D. Raedt and K. Kersting. Probabilistic logic learning. *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining*, 5:2003, 2003.

[15] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[16] T. Sato and Y. Kameya. Prism: a language for symbolic-statistical modeling. In *In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97*, pages 1330–1335, 1997.

[17] C. Sutton and A. McCallum. Collective segmentation and labeling of distant entities in information extraction. Technical Report TR # 04-49, University of Massachusetts, July 2004.

[18] C. Sutton and A. McCallum. An introduction to conditional random fields for relational learning. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[19] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI02)*, 2002.

[20] J. Winn and T. Minka. Infer.net/csoft, 2008. http://research.microsoft.com/mlp/ml/Infer/Csoft.htm.