Efficient Probabilistic Programming Languages

Robert Zinkov
University of Southern California
zinkov@usc.edu

Abstract

In recent years, declarative programming languages specialized for probabilistic modeling has emerged as distinct class of languages. These languages are predominantly written by researchers in the machine learning field and concentrate on generalized MCMC inference algorithm. Unfortunately, all these languages are too slow for practical adoption. In my talk, I will outline several places where compiler optimizations could improve these languages and make them more usable in an industrial setting.

1. Introduction

Probabilistic programming languages are a domain-specific language for probabilistic modeling. They enable fitting models to data using a generative model. A generative model[5] is a probabilistic story for how the observed data was generated from unobserved variables. As an example, consider Latent Dirchilet Allocation[3]. In BUGS[7], this program is a dozen lines long. As a standalone implementation, LDA is hundreds of lines of code. Probabilistic programming languages are a powerful abstraction tool.

2. The Problem

As the size of datasets in machine learning grow larger[1], we have the desire to use more complex models to explain the subtleties of the data. These subtlties are hard to capture with simple linear models. These languages if they can be improved exist in the perfect space for doing this modeling and being very readable.

With the exception of HANSEI[6], most of these languages were not developed by programming languages researchers. They were created to aid in prototyping complex models[7][4].

The majority of the effort has been spent on optimizing the sampling procedure. This ignores that even using a sophisticated MCMC mechanism, these languages take longer to train on hand-written samplers using naive methods.

3. Relevance

This is important as languages significantly slower than hand-written models will not be used. As the models can't be tested with a dataset of realistic size, it isn't even possible to use them for prototyping. This has the nasty and invisible side-effect that artificially simple models are favored as they are the only ones that are testable.

4. Possible Solution

A large degree of the ineffiencies come from lack of compiler optimizations. Only recently was BLOG[2] to use data structures and inference directly from C. Church is written in an interpretted implementation of Scheme.

References

[1] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *CoRR*, abs/1110.4198, 2011.

[2] N. Arora, S. Russell, and E. Sudderth. Automatic inference in blog. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[3] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[4] N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J. Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence*, volume 22, page 23, 2008.

[5] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[6] O. Kiselyov and C. Shan. Embedded probabilistic programming. In *Domain-Specific Languages*, pages 360–384. Springer, 2009.

[7] A. Thomas, D. Spiegelhalter, and W. Gilks. Bugs: A program to perform bayesian inference using gibbs sampling. *Bayesian statistics*, 4:837–842, 1992.

[Copyright notice will appear here once 'preprint' option is removed.]