Building blocks for exact and approximate inference

Jacques Carette
McMaster University

Praveen Narayanan
Indiana University

Wren Romano
Indiana University

Chung-chieh Shan
Indiana University

Robert Zinkov
Indiana University

Abstract

A promising approach to implementing black-box inference is to search a space of inference strategies. However, a search space has not been defined to date that includes combinations of exact and approximate inference methods, such as Rao-Blackwellized Metropolis-Hastings and Gibbs sampling of continuous distributions. We present a handful of inference building blocks that not only constitute black-box inference methods themselves but also generate a search space of inference strategies that includes such combinations. The building blocks include simplifying and sampling from distributions, based in turn on calculating their expectation, disintegration, and density. We have implemented these building blocks as probabilistic-program transformations and specified them in terms of a distribution semantics.

1 Introduction

Ideally, black-box inference could be applied willy-nilly to a broad family of models, such as probabilistic programs. Inside the black box, however, a driver might seek a particular inference strategy suited to each given model, much as a database server might seek a particular execution plan suited to each given query. To carry out this approach to black-box inference, we must define a search space of strategies, like Metropolis-Hastings (MH) proposal distributions. Although approximate methods apply to more models, we want our search space to represent exact methods too when applicable, so that the probabilistic programmer need not manually rewrite their code, say to integrate out a variable.

In this work, we introduce inference building blocks that constitute such a search space—one that represents and composes approximate and exact methods. We implement each building block as a transformation that takes programs in a probabilistic language as input and (non-probabilistically) produces programs in the same probabilistic language. We specify each building block as a black box, in terms of distributions denoted by programs.

2 Specifying building blocks

Figure 1 shows our building blocks as a dependency graph. Each node represents a transformation on probabilistic programs. For example, the Expectation transformation takes as input two programs m and f, where m represents a distribution on some space A and f represents a function from A to $\mathbb{R}$. It produces as output a new program that represents the integral of f with respect to m. Because this transformation is symbolic and exact, m and f may well refer to variables whose values are unknown, which would then also parameterize the output.

We use the term "distribution" to mean a possibly unnormalized distribution, not necessarily a *probability* distribution (which means a distribution whose total is 1). When m happens to be a

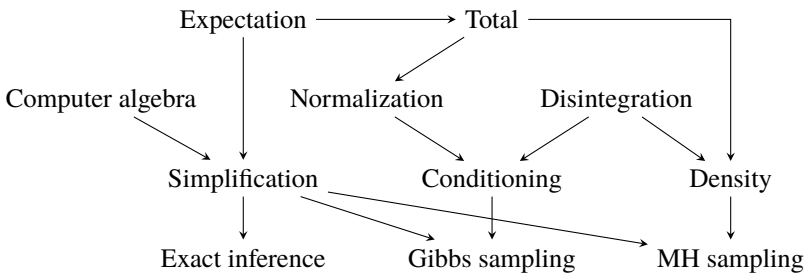

Figure 1: Dependencies among inference building blocks

probability distribution, the program produced represents the expectation of f with respect to m—hence we call this transformation Expectation. In particular, when f is the identity function, we obtain the expected value of m.

Each edge $S \rightarrow T$ in the graph indicates that we use one transformation S to implement another transformation T. For example, the Total transformation turns a program m representing a distribution into a program representing the total of m. Our implementation of the Total transformation simply invokes the Expectation transformation, setting f to the constant function returning 1.

As just demonstrated, we specify what each transformation does in terms of distributions and math. Thus a transformation can be applied to a probabilistic program without exposing their inner workings to each other. In particular, the central transformations in Figure 1 are specified as follows:

Simplification turns one program that represents a distribution into another program that represents the same distribution (and is easier to read and faster to run, we hope). Because we perform symbolic and exact simplification based on computer algebra, the input and output programs may well refer to variables whose values are unknown.

Normalization turns (a program representing) a distribution μ into (another program representing) the *probability* distribution that is a scalar multiple of μ.

Disintegration turns (a program representing) a joint distribution μ on some product space (A, B) into (a program representing) a distribution ν on A and (a program representing) a conditional distribution κ on B given A. The output marginal distribution ν and conditional distribution κ are guaranteed to together generate the input joint distribution μ (Chang and Pollard 1997).

Conditioning turns a joint distribution μ on some product space (A, B) into the conditional *probability* distribution on B given A.

Density turns a distribution μ on some space A into its density, which is a function from A to $\mathbb{R}$.

3 Implementing and using building blocks

We described above how to implement Total using Expectation. It is also easy to implement Normalization using Total, Conditioning using Normalization and Disintegration, and Density using Total and Disintegration. The transformations whose implementation is the most involved and innovative are Simplification (Carette and Shan 2015) and Disintegration (Shan and Ramsey 2015), which we detail elsewhere.

Perhaps a more exciting example is implementing MH sampling (Metropolis et al. 1953; Hastings 1970) as a program transformation. MH sampling is typically described as a randomized algorithm whose coding requires target and proposal densities. Our building blocks render such a textbook description executable: Our implementation of MH sampling uses the Density transformation to generate an MH sampler automatically from a proposal and a target distribution. Part of the generated MH sampler can then undergo Simplification, which would cancel out any repeated factors in the acceptance-ratio computation.

Similarly, Gibbs sampling (Geman and Geman 1984; Gelfand et al. 1992) is typically described as a randomized algorithm whose coding requires a sampler for a conditional probability distribution.

Again our building blocks render such a textbook description executable. In particular, because Simplification recognizes common continuous distributions and recovers their parameters exactly, our implementation of Gibbs sampling can update continuous as well as discrete variables.

Returning to our motivation in the introduction, suppose our model describes a joint distribution

$$p(\theta, x, y) = p(\theta) \cdot p(x \mid \theta) \cdot p(y \mid \theta, x),$$

where θ is some parameter we want to infer, x is latent, and y is observed. Starting with a program representing this distribution, we can obtain a Rao-Blackwellized (Casella and Robert 1996) MH sampler for $p(\theta \mid y)$ as follows.

1. Compose the program with the deterministic function $(\theta, x, y) \mapsto (y, \theta)$, to obtain a program that represents $p(y, \theta)$.
2. Apply Conditioning, to obtain a program that represents $p(\theta \mid y)$ but may sample from x internally.
3. Apply Simplification, to integrate out x with luck.
4. Apply MH sampling, to obtain an MH sampler, which is a transition probability distribution on θ, parameterized by y.
5. Apply Simplification, to cancel out repeated factors in the acceptance-ratio computation.

We emphasize that each of these steps is a non-probabilistic, exact, and symbolic transformation on probabilistic programs. Hence, the final result is the code for an MH sampler that takes a concrete observed value of y as input and runs in time comparable to that of a hand-written MH sampler.

Acknowledgments

This research was supported by DARPA grant FA8750-14-2-0007, NSF grant CNS-0723054, Lilly Endowment, Inc. (through its support for the Indiana University Pervasive Technology Institute), and the Indiana METACyt Initiative. The Indiana METACyt Initiative at IU is also supported in part by Lilly Endowment, Inc.

References

Jacques Carette and Chung-chieh Shan. Simplifying probabilistic programs using computer algebra, 2015. URL `http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR719`.

George Casella and Christian P. Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83 (1):81–94, 1996.

Joseph T. Chang and David Pollard. Conditioning as disintegration. *Statistica Neerlandica*, 51(3): 287–317, 1997.

Alan E. Gelfand, Adrian F. M. Smith, and Tai-Ming Lee. Bayesian analysis of constrained parameter and truncated data problems using Gibbs sampling. *Journal of the American Statistical Association*, 87(418):523–532, 1992.

Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6): 721–741, 1984.

W. Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

Chung-chieh Shan and Norman Ramsey. Symbolic Bayesian inference by lazy partial evaluation, 2015. URL `http://www.cs.tufts.edu/~nr/pubs/disintegrator-abstract.html`.