Composing Modeling and Inference Operations with Probabilistic Program Combinators

Eli Sennesh
Northeastern University
Boston, MA
esennesh@ccis.neu.edu

Adam Ścibior
University of Cambridge and MPI Tübingen
Cambridge, United Kingdom
ams240@cam.ac.uk

Hao Wu
Northeastern University
Boston, MA
haowu@ccis.neu.edu

Jan-Willem van de Meent
Northeastern University
Boston, MA
jwvdm@ccis.neu.edu

Abstract

Probabilistic programs with dynamic computation graphs can define measures over sample spaces with unbounded dimensionality, which constitute programmatic analogues to Bayesian nonparametrics. Owing to the generality of this model class, inference relies on "black-box" Monte Carlo methods that are often not able to take advantage of conditional independence and exchangeability, which have historically been the cornerstones of efficient inference. We here seek to develop a "middle ground" between probabilistic models with fully dynamic and fully static computation graphs. To this end, we introduce a *combinator* library for the Probabilistic Torch framework. Combinators are functions that accept models and return transformed models. We assume that models are dynamic, but that model composition is static, in the sense that combinator application takes place prior to evaluating the model on data. Combinators provide primitives for both model and inference composition. Model combinators take the form of classic functional programming constructs such as `map` and `reduce`. These constructs define a computation graph at a coarsened level of representation, in which nodes correspond to models, rather than individual variables. Inference combinators implement operations such as importance resampling and application of a transition kernel, which alter the evaluation strategy for a model whilst preserving *proper weighting*. Owing to this property, models defined using combinators can be trained using stochastic methods that optimize either variational or wake-sleep style objectives. As a validation of this principle, we use combinators to implement black box inference for hidden Markov models.

1 Introduction

Bayesian nonparametric models have traditionally leveraged exchangeability in order to define predictive distributions that marginalize over an unbounded number of degrees of freedom. In recent years, the field of probabilistic programming has explored a different (yet related) class of models. A probabilistic program can be thought of as a stochastic simulator that is conditioned on observed variables. When the probabilistic programming language supports recursion, probabilistic programs can define priors that sample from models with an unbounded number of random variables, providing a programmatic alternative to classic Bayesian nonparametric models.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

A probabilistic program must support two operations. First, it must be possible to generate samples by evaluating the program. In general, any (halting) evaluation instantiates some finite set of random variables, whose values are referred to as a *trace*. The second operation that must be implemented is the evaluation of the unnormalized density function of a program for any trace. These operations can be formalized in two equivalent forms of denotational semantics in which a program f with inputs y either evaluates to an unnormalized measure $\gamma_f(x \mid y)$, or a weighted sample $x, w \backsim f(y)$ [9, 8]. Inference seeks to characterize the target density $\pi_f(x \mid y) = \gamma_f(x \mid y)/Z_f(y)$. As in other inference problems, the integral $Z_f(y) = \int \gamma_f(x \mid y)\, dx$ is typically intractable, and is typically approximated using Monte Carlo methods.

In both nonparametric models and general probabilistic programs, we can significantly improve the performance of approximate Bayesian inference by imposing some *a priori* assumptions about the graphical form of the joint distribution to be conditioned on our observations. For example, in some models we can alternate between updates to local and global plates of variables. In a Hidden Markov Model (HMM), for instance, we can predict transition probabilities from state sequences, or vice versa. Research in probabilistic programming has traditionally emphasized the development of assumption-free inference methods. To address model-specific inference optimizations, we develop abstractions to modularly and compositionally specify models and inference strategies.

2 Model and Inference Composition

In recent years, there have been a number of efforts to develop specialized inference methods for probabilistic programming. The Venture [3] platform provides primitives for inference programming that can act on subsets of variables in an execution trace. There has also been work to formalize notions of valid inference composition. The Hakaru language [5] frames inference as program transformations, which can be composed so as to preserve a measure-theoretic denotation [11]. Work by Scibior et al. [9] defines measure-theoretic validity criteria for compositional inference transformations.

Models in Probabilistic Torch are written in Python and can make use of `if` expressions, loops, and other control flow constructs. Models can dynamically instantiate random variables in a data-dependent manner, although the computation graph becomes difficult to analyze statically[7]. To reason without static guarantees, we postulate these requirements for model and inference composition:

1. Composition is static, evaluation is dynamic. A model is statically composed from other models, while each evaluation based on data generates a unique trace. In our later HMM example, we can compose a model that samples global parameters with a model for a sequence of variably many states and observations. Evaluations which traverse the same control-flow path while sampling different random values yield traces we can use as samples from the same distribution.

2. Inference operations preserve proper weights. A program f defines a measure $\gamma_f(x \mid y)$, which may be unnormalized, conditioned on some set of inputs y. This measure can represent a prior distribution, or a distribution that is conditioned using observed variables or factors. We here assume that valid evaluation strategies for a program yield *properly weighted* [4] samples (X, W) such that, for all measurable functions h,

$$\mathbb{E}[h(X)W] = \int h(x)\, \gamma_f(x \mid y)\, dx. \tag{1}$$

This property implies that $\mathbb{E}[W] = Z(y)$, i.e. the weight W is an unbiased estimator of the normalizer. A default evaluation strategy that satisfies this assumption is likelihood weighting, in which samples are proposed from the program prior and conditioning operations define an importance weight.

We require that any inference combinator must preserve proper weighting. Operations that satisfy this requirement include importance sampling, importance resampling, Sequential Monte Carlo (SMC), and application of a transition kernel. It follows that any composition of these operations also preserves proper weighting, resulting in an inference strategy that is properly weighted by construction[9].

Model Combinators
$\dfrac{(x, w) \leftsquigarrow f(y) \qquad (x', w') \leftsquigarrow g(x)}{(x', w \cdot w') \leftsquigarrow \texttt{compose}(f, g)(y)}$
$\dfrac{(x, w) \leftsquigarrow f(y_1, y_2)}{(x, w) \leftsquigarrow \texttt{partial}(f, y_1)(y_2)}$
$\dfrac{(x_n, w_n) \leftsquigarrow f(y_n) \quad \text{for } n = 1, \ldots, N}{((x_1, \ldots, x_N), \prod_{n=1}^{N} w_n) \leftsquigarrow \texttt{map}(f, (y_1, \ldots, y_N))}$

Table 1: Big-step semantics for model combinators.

Inference Combinators
$\dfrac{(x, w) \leftsquigarrow g(y) \qquad w' = \gamma_f(x \mid y)/w}{(x, w') \leftsquigarrow \texttt{importance}(f, g)(y)}$
$\dfrac{(x^k, w^k) \leftsquigarrow f(y) \qquad \alpha^k \sim \text{Cat}\left(\frac{w^1}{\sum_{k=1}^K w^k}, ..., \frac{w^K}{\sum_{k=1}^K w^k}\right)}{(x^{\alpha^k}, \frac{1}{K}\sum_{k=1}^K w^k) \leftsquigarrow \texttt{resample}(f, K)(y)}$
$\dfrac{(x, w) \leftsquigarrow f(y) \qquad x' \sim q_g(x' \mid x) \qquad w' = \frac{\gamma_f(x'\mid y) q_g(x \mid x')}{\gamma_f(x \mid y) q_g(x' \mid x)} w}{(x', w') \leftsquigarrow \texttt{move}(f, g)(y)}$

Table 2: Big-step semantics for inference combinators.

3 Model Combinators

A model is a stochastic computation that returns a properly weighted sample. Model evaluation produces a trace, an object that holds values and densities for the set of random variables instantiated during a particular evaluation of the model. Traces can be conditioned on other traces to implement proposals. Combinators accept models as inputs and return a model.

Table 1 shows a number of combinators corresponding to functional programming constructs, with their semantics. In addition to those for which we give the semantics, we have also implemented `reduce` as a basic folding combinator, and such common model families as `ssm` (state-space model), `mixture`, and `hmm` (a hidden Markov model). Of particular note is that we can give semantics for higher-order stochastic functions built using `partial` and `compose`.

4 Inference Composition

Consider running a probabilistic program f to draw a sample x from an unnormalized density $\gamma_f(x \mid y)$. We denote drawing a properly weighted sample x with weight w from $\gamma_f(x \mid y)$ via f as $(x, w) \leftsquigarrow f(y)$. Since proper weights are ratios of unnormalized densities, any joint distribution formed by a probabilistic program constitutes a proper weight. We can thus express as a properly weighted sampler any inference technique which only requires producing samples from programs.

Table 2 shows inference rule semantics for several inference combinators in terms of how they take properly weighted samplers as arguments and return properly weighted samplers in turn. Note that when q denotes a transition kernel that satisfies detailed balance, as used in Markov chain Monte Carlo methods, the new proper weight $w' = w$.

The proposed combinator framework is a natural fit for modern varational methods for training deep probabilistic models. Because the weight w is an unbiased estimator of the normalizer, we can use its logarithm as an evidence lower bound (ELBO) [2] or evidence upper bound (EUBO) [1] to perform

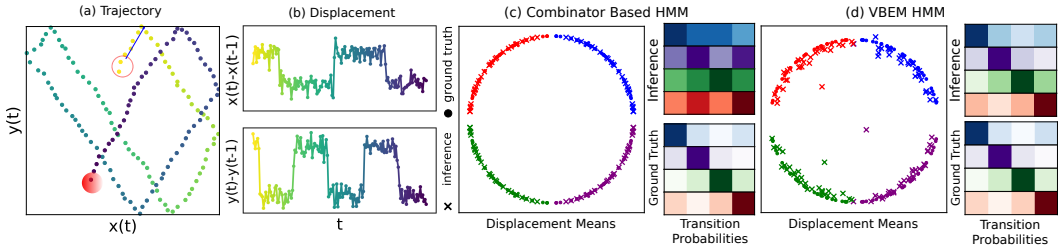

Figure 1: Combinator-based variational inference in hidden Markov models (HMM). a) A bouncing ball trajectory with initial velocity. b) The displacement along x and y axis, respectively. c) Inferred travel directions and transition probabilities from combinator-based wake-sleep Sequential Monte Carlo (SMC). d) Inferred travel directions and transition probabilities from Variational Bayesian Expectation Maximization (VBEM).

variational inference by automatic differentiation in PyTorch. For a parameterized density $\gamma_\theta(x \mid y)$, we can approximate the gradient $\nabla_\theta \log Z_\theta(y)$ using the Monte Carlo estimator

$$\sum_k \frac{w^k}{\sum_l w^l} \nabla_\theta \log \gamma_\theta(x^k \mid y).$$

When we sample from an inference model $q_\phi(x \mid y)$ we can perform wake-sleep style inference by minimizing the objective $\mathrm{KL}(\gamma_\theta(x\,;\,y)/Z_\theta(y) \,||\, q_\phi(x \mid y))$ using the estimator

$$-\sum_k \frac{w^k}{\sum_l w^l} \nabla_\phi \log q_\phi(x^k \mid y).$$

Note that the gradient w.r.t. θ computes $\nabla_\theta \log w$ whereas the gradient w.r.t. ϕ computes $-\nabla_\phi \log w$. In other words, we can perform variational inference in any properly weighted model by automatic differentiation on the importance weights.

5 Evaluation

Figure 1 shows inference results on simulated data. The data models a bouncing particle trajectory in a closed box (Fig. 1a). This trajectory has a piece-wise constant noisy velocity, which means that the displacements at each time step (Fig. 1b) can be described by an HMM with Gaussian observations, where each state's observation mean corresponds to the average velocity along one of four possible directions of motion. We compare wake-sleep SMC inference results for a combinator-based implementation (Fig. 1c) to those obtained using variational Bayesian expectation maximization (VBEM) (Fig. 1d), for a set of 30 time series that each contain 200 time points. VBEM optimizes the exclusive Kullback-Leibler (KL) divergence, $\mathcal{D}_{KL}(q \,||\, p)$, while in our combinator-based inference we optimized $\mathcal{D}_{KL}(p \,||\, q)$, approximating the posterior with greater variance. The combinator-based HMM implementation required 64 lines of model specific code.

6 Extensions and Future Work

In this work, we have considered the fewest assumptions possible about the structural form of the generative model, and so we believe these inference techniques to be the most suited to dynamic probabilistic programs with unbounded dimensionality. Our next step will be to apply combinators to modeling intuitive physics in perception. A variety of extensions are possible that make use of some information about the graphical structure of a model. One opportunity is to apply enumeration or belief propagation to components of the model that are amenable to such operations. Recent work by the Pyro team on sum-product implementations for deep probabilistic programs is relevant in this context [10]. Recent just-in-time compilation strategies [6] could be adapted to construct static graphs for such models, which an optional API could expose to implement inference optimizations.

Acknowledgments

The authors would like to thank David Tolpin for his early interest in this line of work, and the two anonymous reviewers at the Bayesian Nonparametrics workshop for their detailed feedback.

References

[1] Jörg Bornschein and Yoshua Bengio. Reweighted Wake-Sleep. *International Conference on Learning Representations*, 2015.

[2] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474, 2017.

[3] Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: A higher-order probabilistic programming platform with programmable inference. *arXiv*, pages 78–78, March 2014.

[4] Christian Naesseth, Fredrik Lindsten, and Thomas Schon. Nested sequential monte carlo methods. In *International Conference on Machine Learning*, pages 1292–1301, 2015.

[5] Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in Hakaru (system description). In *International Symposium on Functional and Logic Programming*, pages 62–79. Springer, 2016.

[6] PyTorch. Torch Script. https://pytorch.org/docs/master/jit.html.

[7] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient Estimation Using Stochastic Computation Graphs. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3528–3536. Curran Associates, Inc., 2015.

[8] Adam Ścibior, Ohad Kammar, and Zoubin Ghahramani. Functional programming for modular bayesian inference. *Proc. ACM Program. Lang.*, 2(ICFP):83:1–83:29, July 2018.

[9] Adam Ścibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. Denotational Validation of Higher-order Bayesian Inference. *Proc. ACM Program. Lang.*, 2(POPL):60:1–60:29, December 2017.

[10] Uber AI Labs. Pyro documentation. http://docs.pyro.ai/en/dev/ops.html#pyro.ops.contract.ubersum.

[11] Robert Zinkov and Chung-chieh Shan. Composing inference algorithms as program transformations. *Uncertainty in Artificial Intelligence*, 2017.