Effects in Bayesian inference

Adam Ścibior
University of Cambridge
ams240@cam.ac.uk

Ohad Kammar
University of Cambridge
ohad.kammar@cl.cam.ac.uk

Abstract

Recently there has been a lot of interest in the machine learning community in expressing Bayesian models as probabilistic programs in order to make them more reusable and compositional. Such programs have probabilistic effects and different ways of handling those effects correspond to different inference algorithms. Our goal is to present a problem that the HOPE community may find interesting and potentially propose solutions for it.

In this talk we discuss several state-of-the-art Particle Markov Chain Monte Carlo algorithms for inference in probabilistic programs. Our contribution is an implementation of those algorithms using algebraic effect handlers. We utilise an existing effect handler library[1] to obtain a very simple, flexible, and type safe probabilistic programming system. This is very much work in progress and the talk should be treated as a problem description rather than a solution.

Talk proposal

As probabilistic models get more complex and inference algorithms more sophisticated, implementing them becomes a significant software engineering effort. The machine learning community is currently investigating the use of abstraction where a probabilistic model is a program with probabilistic effects and an inference algorithm is a particular way of handling those effects (Goodman and Stuhlmüller 2014). Lots of different probabilistic programming systems were proposed, but in this talk we mostly relate to a recent one called Anglican (Wood et al. 2014).

Overall, the aim of the talk is to present the problem of doing Bayesian inference with effects in an efficient way and to show the sorts of implementation challenges faced by state-of-the-art algorithms. We believe that the machine learning community is lacking good tools that make such implementation easier and could use input from the HOPE audience.

We now introduce the effects required for writing probabilistic models as programs. We use the syntax of Haskell and an effect handler library written by Kammar et al. (2013). The starting point is a data structure `Dist` which encapsulates the common distributions such as Binomial or Normal that we use as basic building blocks. We assume that `Dist` implements the following:

```
sample :: StdGen -> Dist a -> a
pdf    :: Dist a -> a -> Double
```

The function `sample g d` generates a random variable from the distribution `d` using the random number generator `g`. The function `pdf d x` computes the probability density (or mass) function for a distribution `d` at point `x`.

We need two types of effects for Bayesian inference. Those are:

```
[operation| forall a. Draw    :: Dist a -> a       |]
[operation| forall a. Observe :: Dist a -> a -> () |]
```

Here `draw d` generates a random value from a distribution `d`, and `observe d x` asserts that a value `x`, drawn from a distribution `d`, was observed. The `observe` has an effect of modifying the probabilities of different outcomes in a monadic sequence according to the Bayes' rule (Barber 2012):

$$p(\theta|x) = \frac{1}{Z} p(x|\theta) p(\theta)$$

Informally, the prior $p(\theta)$ is defined by `draw`s and the likelihood $p(x|\theta)$ is defined by `observe`s.

A very simple but very popular model we can write as a probabilistic program is linear regression. Here the task is to find a straight line that best matches a set of points. We can write this model as a monadic computation with the two effects above.

```
lr = do
  a <- draw (Normal 0 1)
  b <- draw (Normal 0 1)
  observe (Normal (a*x + b) 1) y
  -- potentially more observations here
  -- can be written as a fold over the data points
  return (a,b)
```

This specification of a probabilistic model is very abstract and does not tie to any particular inference algorithm. It uses algebraic effects (Plotkin and Pretnar 2009) to define an abstract, modular interface for the probabilistic effects. Algebraic effects are an interesting alternative to monad stacks, since they do not require explicit lifting. We use effect handlers (Bauer and Pretnar 2012) to provide a concrete implementation of effects in a modular way.

There are several libraries implementing effect handlers, such as those written by Brady (2013), Kiselyov et al. (2013), and Kammar et al. (2013). In this talk we are using the last of those.

The rest of this proposal presents a simple example of implementing an inference algorithm known as rejection sampling using effect handlers. Then we outline implementation of more sophisticated algorithms and finally discuss some challenges for improving them.

A very basic algorithm is that of rejection sampling. We start by defining a new effect for rejection.

[1] `https://github.com/slindley/effect-handlers`

```
[operation|Reject :: () |]
```

Now we can handle `observe` effects by sampling from the given distribution and then comparing the result with the observed value. If the two are the same we continue, otherwise we reject.

```
[handler|
  forward h handles {Reject, Draw}.
    Rejection a :: a handles {Observe} where
      Return x     -> return x
      Observe d x k ->
        do {y <- draw d; if x == y then k ()
                         else reject >>= k;}
|]
```

Finally we handle rejections by restarting the program from the beginning.

```
[handler|
   forward h.
     Repeat a :: Comp (Repeat h a) a -> a handles {
         Reject} where
       Return x  _ -> return x
       Reject  k c -> repeat c c
|]
```

To complete the implementation we only require a handler for `draws`. We pass around a random number generator to actually obtain values.

```
[handler|
  Run a :: StdGen -> a handles {Draw} where
    Return x  g -> x
    Draw  d k g -> let (x,g') = sample g d in k x g'
|]
```

Now we can obtain an executable program by composing the handlers.

```
sampler :: StdGen -> (Double,Double)
sampler g = run g $ repeat d d where
  d = rejection lr
```

Unfortunately, rejection sampling is slow and it is not applicable to zero probability observations, such as drawing a particular value from a continuous distribution. In particular running the above program never terminates with probability 1. Nonetheless, rejection sampling can be used with discrete distributions, although it is still very slow.

In the talk we show a number of alternative, more efficient inference algorithms. They range from relatively simple importance sampling, where the probability densities for each `observe` are accumulated during the execution of the program and at the end used to weigh the samples, to Particle Markov Chain Monte Carlo (PMCMC), currently state-of-the-art for probabilistic programming. Implementing PMCMC methods is more challenging, since it requires pre-empting the execution of a program and potentially spawning several independent copies of it at a given point. Nonetheless, effect handlers make the task relatively easy.

We also indicate what are the possible ways to improve those algorithms and what implementation challenges they present. A popular alternative to PMCMC is the single-site Metropolis-Hastings (MH) algorithm introduced by Wingate et al. (2011). The key to achieving good performance with MH is being able to identify "the same" random choices in different runs. Formally this is not a well-defined problem, but in practice it seems that good heuristics can be found. It would be interesting to see what are some principled ways to derive such heuristics.

Other ideas for improving the inference algorithms include adapting proposals for `draws` in subsequent runs, hence generating more good samples, or combining several inference algorithms by composing their corresponding handlers. Overall, we believe that significant advances can be made in improving inference algorithms, but the speed with which that happens will greatly depend on the availability of good tools that make implementation easier and abstract away the irrelevant parts of the program.

References

D. Barber. *Bayesian Reasoning and Machine Learning*. CUP, 2012.

A. Bauer and M. Pretnar. Programming with algebraic effects and handlers. *CoRR*, abs/1203.1539, 2012.

E. Brady. Programming and reasoning with algebraic effects and dependent types. In *ICFP*. ACM, 2013.

N. D. Goodman and A. Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. `http://dippl.org`, 2014. Accessed: 2015-5-22.

O. Kammar, S. Lindley, and N. Oury. Handlers in action. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 145–158, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2326-0. . URL `http://doi.acm.org/10.1145/2500365.2500590`.

O. Kiselyov, A. Sabry, and C. Swords. Extensible effects: An alternative to monad transformers. In *Proceedings of the 2013 ACM SIGPLAN Symposium on Haskell*, Haskell '13, pages 59–70, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2383-3. . URL `http://doi.acm.org/10.1145/2503778.2503791`.

G. D. Plotkin and M. Pretnar. Handlers of algebraic effects. In *ESOP*. Springer-Verlag, 2009.

D. Wingate, A. Stuhlmueller, and N. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Proceedings of the 14th Intl. Conf. on Artificial Intelligence and Statistics*, page 131, 2011.

F. Wood, J. W. van de Meent, and V. Mansinghka. A new approach to probabilistic programming inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*, 2014.