The Support Method of Computing Expectations

Avi Pfeffer
Charles River Analytics
apfeffer@cra.com

1. INTRODUCTION

A standard method for computing the expectation $\int p(x)f(x)dx$ of a function f on a probabilistic program p is to sample values $x_1, \ldots, x_n$ from the generative model of p and approximate the integral by $\frac{1}{n}\sum_{i=1}^{n} f(x_i)$. An alternative approach is to deterministically enumerate a set of values $x_1, \ldots, x_n$ in the support of p, give each point x_i a weight w_i, and numerically approximate the integral by $\sum_{i=1}^{n} w_i p(x_i) f(x_i)$. If the points and weights are well chosen, this can lead to a more efficient computation than Monte Carlo sampling. For programs with small finite support, we can simply enumerate all values and give each a weight of 1 to obtain the standard definition of expectation. However, this method can also work well for programs with very large or infinite support.

Although this method is conceptually simple and well-known, it is surprisingly tricky to implement. It has a long history in search-based approaches for graphical models, going back to the work of Henrion [1], Poole [2], and Dechter and colleagues [3]. However, these methods have generally been restricted to discrete models and simpler languages than probabilistic programming. One challenge is to develop a good method for enumerating the support of continuous variables with infinite support. Another challenge is to avoid double counting values in models with multiple variables, as will be explained in Section 3.

2. CONTINUOUS VARIABLES

For continuous variables with infinite support, we generate an infinite stream of values. We have implemented this for normal and uniform random variables, but we anticipate that this method works for many kinds of continuous variables. We would like four properties to hold of this stream: (1) for any prefix, the values in the prefix should cover the range of the variable reasonably evenly; (2) no points should dominate the estimate, so given a prefix all points should have approximately equal weight; (3) in the limit, the points should cover the entire range of the variable; and (4) for any prefix, the weights should be such that $\sum_{i=1}^{n} w_i p(x_i)$ is approximately 1.

From properties (3) and (4), we can see that points in an earlier prefix will have higher weights than points in later prefix. But from property (2), we know that by the time the later prefix is generated, the weights of points earlier in the stream will have to be reduced so they don't dominate. This implies that as we add points to the stream, we have to reduce the weights of points already in the stream in a systematic and efficient way.

For uniform random variables, we use the following approach. We proceed in iterations, starting from iteration 0 with the midpoint, giving it weight 1. In each iteration n, we add 2^n points. At the end of iteration n, there are $2^{n+1} - 1$ evenly spaced points, with a gap of $\frac{u-l}{2^{n+1}}$ between them, where u and l are the upper and lower endpoints of the uniform variable. Properties (1) and (3) are satisfied by this construction at the end of an iteration. The weight of each point at the end of iteration n is equal to the gap, so that $\sum_{i=1}^{2^{n+1}-1} w_i p(x_i) = (2^{n+1} - 1)\frac{u-l}{2^{n+1}}\frac{1}{u-l}$, so property (4) is satisfied. For property (2), in each iteration, when we add 2^n new points with weight $\frac{u-l}{2^{n+1}}$, we subtract a weight of $\frac{u-l}{2^{n+1}}$ from the $2^n - 1$ existing points, which previously had weight of $\frac{u-l}{2^n}$.

We also need to make sure that the prefix remains balanced within an iteration. Our method is as follows. We add two points, one on either side of the midpoint. While doing that, we subtract $\frac{u-l}{2^{n+1}}$ once from the weight of the midpoint. Then we create two separate lists. In one, we add a new point to the left of each old point that was left of the midpoint, subtracting the appropriate weight from the old point. The second is symmetrical to the right of the midpoint. We interleave these two lists to obtain a balance of larger and smaller values. In our implementation, each of these lists is ordered from the middle out, which may lead to an imbalance of central values over extreme values during an iteration. It would be interesting to think about better methods for ordering each of these lists that can be done efficiently. Our method works in linear time in the length of the prefix, which is crucial.

We have a similar method for normal random variables. The main difference is that the range of normal variables is the entire real line, so we expand the interval covered by the prefix in each iteration. Since the interval is centered around the mean, we concentrate most on high-density points but also reach outliers quickly in case they contribute significantly to the expectation.

Figure 1 compares the support method to sampling for a probability query over a uniform random variable. We have obtained similar results for normal random variables. The horizontal axis shows the length of the prefix used for the support method while the vertical axis shows the ratio of the error under the sampling method to the error under the support method. Because the support method takes roughly twice as long to generate a point as sampling, we give the sampling method twice as many points. With just five points for the support method, the two methods are about equally good, but as the number of points increases, the support method becomes dramatically better, with an approximately linear increase with the number of points.

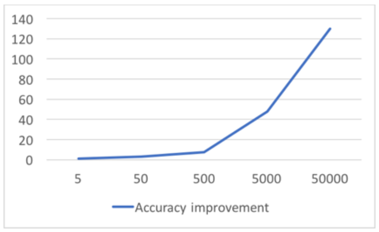

Figure 1: Improvement of support method over sampling

3. MULTIPLE VARIABLES

When we encounter models involving multiple variables, this simple approach runs into trouble. Suppose we have a variable x with model $p(x)$ and a variable y defined by a conditional model $p(y|x)$, so that $p(y) = \int p(x)p(y|x)dx$. For concreteness, consider the model

```
x = flip 0.7
y = if x then flip 0.2 else flip 0.6
```

The naïve approach is to enumerate the support of $p(x)$, and for each value x_i with weight w_i, enumerate the support of $p(y|x_i)$, giving value y_{ij} a weight v_{ij}. We then approximate the expectation by $\sum_i w_i \sum_j v_{ij} p(y_{ij}) f(y_{ij})$. The problem is that if the same value y_{ij} occurs for different values of x, we will double count its density. This problem shows up immediately in our example. For a flip random variable, the expectation will be approximated by the sum of four cases, corresponding to x and y each being true or false. The weights for flip are all 1, so in each of the two cases where $y = true$, we add $p(true)f(true)$, and similarly for $y = false$. Therefore, we overestimate the expectation by a factor of 2.

Our solution is to define each y_{ij} as providing a *contribution* c_{ij} to the estimate of the expectation, so that, for any value y_j, the total weighted density of y_j is the sum of the contributions, i.e., $v_j p(y_j) = \sum_{i:y_{ij}=y_j} c_{ij}$. We then approximate the expectation with $\sum_i w_i \sum_j c_{ij} f(y_{ij})$. This necessitates a fundamental change to the support method. Previously, we only enumerated values with weights and left density computation until expectations are computed. Now, density computation is embedded directly into support enumeration, where we are enumerating contributions to the expectation.

For a single variable x like the uniform and normal variables described earlier, the contribution of x_i is simply $w_i p(x_i)$. For the case of a variable y defined by a conditional distribution $p(y|x)$, the contribution c_{ij} of a value y_{ij} is $c_i v_{ij} p(y_{ij}|x_i)$. Let us assume inductively that $c_i = w_i p(x_i)$, and the w_i are such that $\sum_i w_i p(x_i) g(x_i)$ are a good approximation to the expectation of $g(x)$. Also, assume that the v_{ij} are such that $\sum_j v_{ij} p(y_{ij}) f(y_{ij})$ is a good approximation to the conditional expectation of $f(y)$ given x_i. Then, using the contributions c_{ij},

$$\begin{aligned}\sum_i \sum_j c_{ij} f(y_{ij}) &= \sum_i \sum_j c_i v_{ij} p(y_{ij}|x_i) f(y_{ij}) \\ &= \sum_i w_i p(x_i) \sum_j v_{ij} p(y_{ij}|x_i) f(y_{ij}) \\ &\approx \mathrm{E}_x\left[\sum_j v_j p(y_j|x) f(y_j)\right] \\ &\approx \mathrm{E}_x\left[\mathrm{E}_{y|x}[f(y)]\right] = \mathrm{E}_y[f(y)]\end{aligned}$$

In the case of the finite example above, the support method requires only four values of y_{ij} to fully enumerate the support, so it is exact and clearly superior to sampling. On a mixture model with uniform variables depending on a flip, the support method is also superior to sampling, obtaining comparable improvements to Figure 1.

4. DISCUSSION

The approach described in this paper bears some resemblance to Bayesian quadrature. Like our approach, quadrature estimates an integral by considering the densities of a number of points in the domain of the integral. However, there are a number of fundamental differences between the approaches. A minor difference is that Bayesian quadrature typically uses the generated points to represent a probability density over the domain, for example using a Gaussian process. In contrast, we use any choice of points and weights that conform to the guidelines presented in Section 2, without giving them an explicit probabilistic interpretation. More significantly, we provide an anytime approximation scheme for computing the expectation that iteratively refines the enumeration of the support. This scheme can actually involve the reduction or elimination of contribution of points in subsequent steps. Perhaps most of consequence is that we provide a general, composable method for using this approach with multiple variables.

It is natural to compare the support method to alternative sampling approaches. Forward sampling methods like importance sampling can often have problems when the posterior is very different from the prior. Because the support method enumerates the domain evenly, it may be less susceptible to this problem. For example, using our construction for Gaussian models, we enumerate regions distant from the mean in time linearly proportional to the distance, as opposed to exponential for forward sampling algorithms. We need to study how the approaches compare on some examples. We also need to study how the support method compares to MCMC on a variety of examples.

A point of possible concern is whether the support method scales to high dimensions. Instance-based methods like this often scale poorly. Our intuition is that this depends strongly on the structure of the model. For chain-like models, we expect that the dimensionality would essentially be reduced at every step. However, for wide models in which nodes depend on many parents, it might be difficult to explore the joint space of the parents effectively. This warrants further study.

REFERENCES

[1] M. Henrion, "Search-based methods to bound diagnostic probabilities in very large belief nets," in *Proc. Uncertainty in Artificial Intelligence*, 1991.

[2] D. Poole, "Probabilistic conflicts in a search algorithm for estimating posterior probabilities in Bayesian networks," *Artificial. Intelligence*, vol. 88, no. 1–2, pp. 69–100, 1996.

[3] R. Dechter and R. Mateescu, "AND/OR search spaces for graphical models," *Artificial. Intelligence*, vol. 171, no. 2–3, pp. 73–106, 2007.