Resampling with infinitesimal probabilities

Jules Jacobs

May 8, 2021

Abstract

In this note I explain why resampling works for infinitesimal probabilities. This question was raised by one of the reviewers of the paradoxes of probabilistic programming paper [Jac21]. Since at least one other person wondered the same, I put the answer I gave in this note.

SMC can be made to work quite generally, even when the distributions differ on different paths of execution, or when there are a different number of observes in different paths, if one uses probabilities rather than probability densities. Suppose for the moment that we have a probabilistic program where all observes are of finite non-zero measure (i.e. `observe(D,I)` for D discrete or D continuous but I an interval of finite width). Suppose that we are trying to estimate some expectation $\mathbb{E}[f(Y)]$ where Y is a random variable that represents the return value of the probabilistic program.

We run a trial of importance sampling, and suppose that in the middle of running the trial we suspect that this trial will contribute negligibly to the final weighted average. Rather than spending more time executing the rest of the trial, we'd like to cancel the trial and start over. Simply rejecting the trial doesn't work, because that will affect the estimate of $\mathbb{E}[f(Y)]$. Instead, what we do is flip a coin $X \sim \mathsf{Bernoulli}(p)$ with e.g. $p = 0.1$. If $X = 1$ we multiply the weight of the trial by $\frac{1}{p}$, and if $X = 0$ we reject the trial. Or more simply, we multiply the weight of the trial by $\frac{1}{p}X$. This will cancel the trial in 90% of the cases, so we save quite a bit of work. This type of resampling can also be done from within the probabilistic program, by inserting this somewhere in the program:

```
if(flip(p) == false){ reject(); }
weight *= 1/p;
```

This will not change the expectation value of the final contribution of the trial, because

$$\mathbb{E}[\frac{1}{p}Xf(Y)] = \frac{1}{p}\mathbb{E}[X]\mathbb{E}[f(Y)] = \frac{p}{p}\mathbb{E}[f(Y)] = \mathbb{E}[f(Y)]$$

Resampling is therefore a very general technique. We can choose p arbitrarily; we can even choose it as a function of the state of the program variables at that point, as long as the coin flip $X \sim \mathsf{Bernoulli}(p)$ itself is independent of Y.

In general it is hard to know whether a trial will eventually end up being negligible, so SMC approximates this by running multiple trials simultaneously and using their relative weights at synchronized points as its choice of p_i. After resampling has removed some trials, we can restore the same number of trials by copying a trial with weight w into n identical trials of weight w/n, which will also not change the expectation. Note that the technique of resampling is in principle valid at non-synchronized points. SMC could choose to arbitrarily run different trials for different number of steps, and then do a resampling step with p_i chosen arbitrarily (as long as $p_i > 0$). Doing this badly will ruin the speed of convergence, but the expectation value is still correct, so running a sufficiently large number of trials will still make the final result converge to the correct value.

With infinitesimal observes, the weights of the trials may become infinitesimal. Since we must have $p_i > 0$, we have to modify the heuristic that chooses the weight for the p_i, because even if some trial

is infinitesimally small compared to some other trial, we cannot simply throw it away with probability 1, because further execution might change this. We could however use the heuristic that we substitute $\epsilon = 0.1$ in order to compute the relative weights p_i, or we could run the trials forward until they have all acquired precisely the same powers of ϵ, and do resampling at that point.

Additional comments

The resampling I describe above is a special case. In general we have a whole batch of active executions $\vec{Y}$ with weights $\vec{w}$, and we randomly sample a new weight vector $\vec{w}' \sim R(\vec{w}, \vec{Y})$ from some distribution R that takes the old weights as input. In order for this operation to be correct, the expectation value must not change, *i.e.,*

$$\mathbb{E}[\sum_i w_i f(Y_i)] = \mathbb{E}[\sum_i w'_i f(Y_i)] \tag{1}$$

For which it suffices that w'_i is independent of Y_i and $\mathbb{E}[w'_i] = w_i$. We can further generalize to allow samples to be resampled to multiple copies. In that case the expectation of the *sum* of the new weights needs to be equal to the old weight.

Batches are not strictly necessary?

SMC uses batches to get its effect, but I think a similar effect can be achieved without them. We can execute samples sequentially as in importance sampling, and use the simple form of resampling when we suspect that a trial's contribution will be negligible. We can get the same effect as SMC by keeping track of the average weight of trials at specific checkpoints in the execution. Subsequent trials can then check if their particular weight value is much smaller than the average, and if so, decide to do a resampling step. If the weight is much larger than average, we could decide to do a splitting step, to reduce the variance. If we don't do any splitting steps, then this form of "sequential importance resampling" may be easier to implement, since you don't need to run different trials in parallel and have all the continuations machinery that Anglican uses. Not sure how effective that would be though! An alternative to still allow splitting might be to save the random seed at the start, and when we want to split, reduce the weight but insert the seed and the remaining weight into some todolist to execute later. We can then use that seed to run to exactly the same point again. Of course to actually reduce the bias we'd want to choose a new random seed after that point, so managing all this may still be a bit complicated.

References

[Jac21] Jules Jacobs. Paradoxes of probabilistic programming: and how to condition on events of measure zero with infinitesimal probabilities. *Proceedings of the ACM on Programming Languages*, 5(POPL):58:1–58:26, January 2021. doi:10.1145/3434339.