Particle Filters

Alex Pan and Alec Kosik

Abstract

Particle Filters allow us to model non-linear and non-Gaussian distributions through Monte Carlo methods. We first discuss HMM, and then discuss a recursive formulation of the distribution of $p(x_t|y_{1:n})$. We then discuss how this leads well to an algorithmic approach. Finally, we investigate Kalman Filters, Importance Sampling, and SIR. We discuss the motivations for and relevant problems with each approach.

1 Hidden Markov Models

A **Hidden Markov Model** (HMM) is defined as a Markov process with states that cannot be observed, but gives outputs that are dependent on the hidden state of the system. If we let the random variable X_i represent the i-th state of the system, and the random variable Y_i represent the i-th output of the system, we have:

$$X_1 \sim \mu(x_1) \text{ and } X_n|(X_{n-1} = x_{n-1}) \sim f(x_n|x_{n-1}) \tag{1}$$

$$Y_n|(X_n = x_n) \sim g(y_n|x_n) \tag{2}$$

where X_1 is defined by some initial conditions $\mu(x_1)$. Note how f obeys the Markov property, so the current state x_n is dependent only on the previous state x_{n-1}. Also note that g shows that the current observation y_n is only dependent on the current state x_n.

Let $x_{1:n}$ represent the ordered n-tuple of hidden states $(x_1, \ldots, x_n)$ where x_i is the i-th hidden state. Similarly, let $y_{1:n}$ represent the ordered n-tuple of observations $(y_1, \ldots, y_n)$. There are a few important distributions that follow directly from our HMM setting.

$$\begin{aligned} p(x_{1:n}) &= p(x_1, \ldots, x_n) \\ &= p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\ldots p(x_n|x_{n-1}, \ldots, x_1) \\ &= p(x_1)p(x_2|x_1)p(x_3|x_2)\ldots p(x_n|x_{n-1}) \\ &= \mu(x_1)\prod_{i=2}^{n} f(x_1|x_{i-1}) \end{aligned} \tag{3}$$

$$\begin{aligned} p(y_{1:n}|x_{1:n}) &= p(y_1, \ldots, y_n|x_{1:n}) \\ &= p(y_1|x_{1:n})p(y_2|x_{1:n}, y_1)(y_3|x_{1:n}, y_2, y_1)\ldots(y_n|x_{1:n}, y_{n-1}, \ldots, y_1) \\ &= p(y_1|x_1)p(y_2|x_2)\ldots(y_n|x_n) \\ &= \prod_{i=1}^{n} g(y_i|x_i) \end{aligned} \tag{4}$$

2 Particle Filters

2.1 General Goals

There are a few different uses of the particle filtering method. We could model the distribution $p(x_{1:n}|y_{1:n})$, the distribution $p(x_n|y_{1:n})$, or even the expected value of functions defined on these distributions. Here, our main goal is to use our observations to determine the hidden state of the system. That is, we want to figure out $p(x_n|y_{1:n})$. Recognizing that this is a marginal is useful because then we can just say:

$$p(x_n|y_{1:n}) = \int p(x_{1:n}|y_{1:n})dx_{1:n-1} \tag{5}$$

Setting our goal as a marginal of $p(x_{1:n}|y_{1:n})$ is helpful because we can refactor the full conditional distribution to see how it comes from our prior distributions derived in (3) and (4). Refactoring and using our initial HMM conditions, we can show:

$$
\begin{aligned}
p(x_{1:n}|y_{1:n}) &= \frac{p(x_{1:n}, y_{1:n})}{p(y_{1:n})} \\
&= \frac{p(x_{1:n})p(y_{1:n}|x_{1:n})}{p(y_{1:n})} \\
&= \frac{p(x_{1:n})p(y_{1:n}|x_{1:n})}{\int p(x_{1:n}, y_{1:n})dx_{1:n}} \\
&= \frac{\mu(x_1)\prod_{i=2}^{n} f(x_1|x_{i-1})\prod_{i=1}^{n} g(y_i|x_i)}{\int \mu(x_1)\prod_{i=2}^{n} f(x_1|x_{i-1})\prod_{i=1}^{n} g(y_i|x_i)dx_{1:n}}
\end{aligned} \tag{6}
$$

Thus,

$$
p(x_n|y_{1:n}) = \int \frac{\mu(x_1)\prod_{i=2}^{n} f(x_1|x_{i-1})\prod_{i=1}^{n} g(y_i|x_i)}{\int \mu(x_1)\prod_{i=2}^{n} f(x_1|x_{i-1})\prod_{i=1}^{n} g(y_i|x_i)dx_{1:n}} dx_{1:n-1} \tag{7}
$$

This looks really messy, but it's helpful to see that this is just integration of functions we already know: f and g. In other words, we already have all the information we need to arrive at the answer.

2.2 A Recursive Formulation

While we now have a nice closed-form solution, it would be really inefficient if we had to calculate this whole equation every single iteration of our system. Instead, we can formulate this recursively and see how some of the values carry through. We can show that:

$$
p(x_t|y_{0:t}) = \frac{p(x_t|y_{0:t-1})p(y_t|x_t)}{p(y_t|y_{0:t-1})} \tag{8}
$$

To see how this comes about, we will approach this from another angle using Bayes' Rule:

$$
\begin{aligned}
p(x_t|y_{0:t}) &= \frac{p(x_t, y_{0:t})}{p(y_{0:t})} \\
&= \frac{p(x_t, y_{0:t})}{p(y_t|y_{0:t-1})p(y_{0:t-1})}
\end{aligned} \tag{9}
$$

We can also use Baye's rule to get a better look at the recursive step:

$$
p(x_t|y_{0:t-1}) = \frac{p(x_t, y_{0:t-1})}{p(y_{0:t-1})} \tag{10}
$$

Plugging both of these back into our initial equation for $p(x_t|y_{0:t})$ gives us:

$$
\begin{aligned}
\frac{p(x_t, y_{0:t})}{p(y_t|y_{0:t-1})p(y_{0:t-1})} &= \frac{p(x_t, y_{0:t-1})}{p(y_{0:t-1})}\frac{p(y_t|x_t)}{p(y_t|y_{0:t-1})} \\
\implies p(x_t, y_{0:t}) &= p(x_t, y_{0:t-1})p(y_t|x_t)
\end{aligned} \tag{11}
$$

So now we want to show that this equation is true. Let's start by refactoring the right-side.

$$\begin{aligned}
p(x_t, y_{0:t-1})p(y_t|x_t) &= p(x_t|y_{0:t-1})p(y_{0:t-1})\frac{p(x_t, y_t)}{p(x_t)} \\
&= p(x_t|y_{0:t-1})p(y_{0:t-1})\frac{p(x_t|y_t)p(y_t)}{p(x_t)} \\
\text{Using the fact that: } p(x|yz) &= \frac{p(y)p(z)}{p(yz)}\frac{p(x|y)p(x|z)}{p(x)} \\
\text{Let: } x = x_t, y = y_{0:t-1}, z = y_t & \\
\text{We can see: } p(x_t|y_{0:t-1}, y_t) &= \frac{p(y_{0:t-1})p(y_t)}{p(y_{0:t-1}, y_t)}\frac{p(x_t|y_{0:t-1})p(x_t|y_t)}{p(x_t)} \\
\implies p(x_t|y_{0:t-1})p(y_{0:t-1})\frac{p(x_t|y_t)p(y_t)}{p(x_t)} &= p(x_t|y_{0:t})p(y_0:t) \\
&= p(x_t, y_{0:t})
\end{aligned} \tag{12}$$

...
Some stuff with integrals :)
...

The issue now is that this is sometimes intractable. If functions are nice, like guassians or linear distributions, this is integrable and we can arrive at a solution analytically. However, this is not always the case, which is why we sometimes need to derive an answer from numeric approximations.

3 Kalman Filter

This needs to be re-done because I've moved the discussion of the recursive form to the earlier section. Former code has been commented out.

4 Sequential Monte Carlo Methods

So what if our distribution isn't nice? We will use Monte Carlo Methods to approximate Eq. (5) and (6):

$$\text{Prediction step: } \pi_{x_{1:t}|y_{1:t-1}}(x_{1:t}|y_{1:t-1}) = f(x_t|x_{t-1})\pi_{x_{1:t-1}|y_{1:t-1}}(x_{1:t-1}|y_{1:t-1}) \tag{13}$$

$$\text{Update step: } \pi_{x_{1:t}|y_{1:t}}(x_{1:t}|y_{1:t}) \propto g(y_t|x_t)\pi_{x_{1:t}|y_{1:t-1}}(x_{1:t}|y_{1:t-1}) \tag{14}$$

5 SIS

To algorithmically find our target distribution, we sample from something simple like the Uniform or Gaussian distribution. We then weight these samples using our observations and g. Essentially, we iterate between (7) and (8), sampling new points and updating our weights each time. While this sounds okay, it turns out that we never use SIS in real life because of something called the "degeneracy problem".

6 Degeneracy Problem

When we are sampling from our reweighted distribution, it is possible that certain points are assigned very low weight. In future iterations, this is likely to compound, making their weights essentially negligible. This manifests as a handful of points having nearly all the weight while the rest of our points have "degenerated". This is a problem because our distribution essentially is now defined by just a few of the original points we started with, which is not enough to get an accurate representation of our target distribution.

7 Bootstrap (SIR)

1. *Initialization:*

   For $i = 1, \dots N$:

   Sample $x_0^{(i)} \sim \mu(x_0)$

   Assign weights $\widetilde{w}_0^{(i)} = g(y_0|x_0^{(i)})$

   Normalize weights $w_0^{(i)} = \frac{\widetilde{w}_0^{(i)}}{\sum_{i=1}^n \widetilde{w}_0^{(i)}}$

2. *Importance Sampling:*

   For $t = 1, \dots, T$:

   Sample $x_t^{(i)} \sim f(x_t|\widetilde{x}_{t-1}^i)$

   Assign weights $\widetilde{w}_t^{(i)} = g(y_t|x_t^{(i)})$

   Normalize weights $w_t^{(i)} = \frac{\widetilde{w}_t^{(i)}}{\sum_{i=1}^n \widetilde{w}_t^{(i)}}$

   Resample $\widetilde{x}_t^{(i)}$ from $x_t^{(i)}$ according to the weight distribution with replacement.

3. Return $\{x_T^{(i)}, w_T^{(i)}\}_{i=1}^N$

While this addresses the degeneracy problem, it turns out that resampling has it's own issue which we call the "sampling impoverishment problem". This comes about because our higher weighted points are more likely to be drawn multiple times. Instead of having many low-weight points spread out across the state-space, we will have lots of evenly-weighted points concentrated around areas where high-weight points originally occurred. There are methods that address both of these issues but they require fancy sounding things like the "Epanechnikov Kernel", which our outside the scope of our discussion. It turns out that SIR is generally good enough and is the algorithm that people generally use.