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 $p(x_t|y_{1:n})$ and 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 Introduction

Imagine that we have some outside information about a system and that we can generate observations which are dependent on the state of the system. However, the actual state of the system is hidden from us. How can we go about finding the hidden state of the system using our observations? It turns out we can approximate the hidden state through a process called **particle filtering**.

To give a toy example of this, imagine that we have a robot in a room. We know the layout of the room, but we do not know the position of our robot. However, the robot is able to take some sort of measurements with a sensor, for example its distance to walls. Using the sensor data and our existing information about the layout of the room, we use particle filtering to make predictions about where we could be in the room.

We will begin by formally defining the problem setting and then deriving particle filtering algorithms.

2 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.

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

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

3 Particle Filters

3.1 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.

3.2 A Recursive Formulation

While we now have a nice equation, 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. This recursiveness is described in two parts: the update equation and the prediction equation. We show the update equation first:

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

The derivation will be broken into 2 parts. First, we will show that (8) holds iff $p(x_t, y_{0:t}) = p(x_t, y_{0:t-1})p(y_t|x_t)$ holds. Then we will show that $p(x_t, y_{0:t}) = p(x_t, y_{0:t-1})p(y_t|x_t)$. We start by applying Bayes' rule to the left side of (8) and then moving the denominator to the right side:

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

Applying Bayes' rule to $p(x_t|y_{0:t})$:

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

Second, we show $p(x_t, y_{0:t}) = p(x_t, y_{0:t-1})p(y_t|x_t)$. Let's start by refactoring the right-side.

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

Using the fact that: $p(x|yz) = \frac{p(y)p(z)}{p(yz)}\frac{p(x|y)p(x|z)}{p(x)}$ Let: $x = x_t, y = y_{0:t-1}, z = y_t$

We can now see that: $p(x_t|y_{0:t-1}, y_t) = \frac{p(y_{0:t-1})p(y_t)}{p(y_{0:t-1}, y_t)}\frac{p(x_t|y_{0:t-1})p(x_t|y_t)}{p(x_t)}$

Thus,

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

We now show how to derive the prediction equation:

$$p(x_t|y_{0:t-1}) = \int f(x_t|x_{t-1})p(x_{t-1}|y_{0:t-1})dx_{t-1} \tag{9}$$

If we approach this as a marginal by integrating out x_{t-1} we normally would formulate this as:

$$\begin{aligned} p(x_t|y_{0:t-1}) &= \int p(x_{t-1:t}|y_{0:t-1})dx_{t-1} \\ &= \int p(x_t|x_{t-1}, y_{0:t-1})p(x_{t-1}|y_{0:t-1})dx_{t-1} \end{aligned}$$

We can make use of the Markov property to see:

$$p(x_t|x_{t-1}, y_{0:t-1}) = p(x_t|x_{t-1}) = f(x_t|x_{t-1})$$

The equations (9) and (8) give us what are called the *prediction* and *update* steps respectively. They also make clear an algorithmic approach: by iteratively repeating these steps, we can find $p(x_i|y_{0:i})$ for any i from our initial state up to our n-th state. This is also far more efficient computationally because we can save time by using information we already have from the previous iteration, instead of trying to perform a large integration every single step.

3.3 Kalman Filter

Setting different kinds of constraints on the problem leads to different kinds of solutions. It turns out that when $p(x_{t-1}, y_{0:t-1})$ is Gaussian, we can show that $p(x_t, y_{0:t})$ is Gaussian provided that f and g are linear functions on their inputs. In cases when these conditions hold, we can use the **Kalman Filter** to find $p(x_t, y_{0:t})$.

Essentially the Kalman Filter reformulates everything in terms of matrices since f and g are linear. An important thing to note is that the Kalman Filter typically extends our problem setting to include noise.

That is to say, our transition and observation distributions both have noise parameters. In a physical setting, this might look something like the wind blowing on the robot while it moves, which might affect its location as it moves, and uncertainty in our sensor data, which is what we would expect in any real life measurement.

We can then reformulate x_k and y_k as a system of matrix equations, and also show that (9) and (8) turn out to be Gaussian distributions. From there, we can derive another set of equations that give us the Kalman Filter, but we won't go through a statement or proof of these equations because some of the linear algebra goes outside the scope of this discussion.

4 Sequential Monte Carlo Methods

At the end of the day, our biggest constraint is the integrability of our distributions. For example, as the prediction step (Eq. (9)) states, we need to be able to perform an integration for this to work out. When integration is not possible, we will use use Monte Carlo Methods to approximate Eq. (5) and (6):

$$\text{Prediction step: } \pi_{x_{1:t}|y_{1:t-1}}(x_{1:t}|y_{1:t-1}) = f(x_t|x_{t-1})\pi_{x_{1:t-1}|y_{1:t-1}}(x_{1:t-1}|y_{1:t-1}) \tag{10}$$

$$\text{Update step: } \pi_{x_{1:t}|y_{1:t}}(x_{1:t}|y_{1:t}) \propto g(y_t|x_t)\pi_{x_{1:t}|y_{1:t-1}}(x_{1:t}|y_{1:t-1}) \tag{11}$$

5 SIS

5.1 SIS Algorithm

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 $x_t^{(i)} \sim f(x_t|x_{t-1}^i)$

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

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

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

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".

5.2 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.

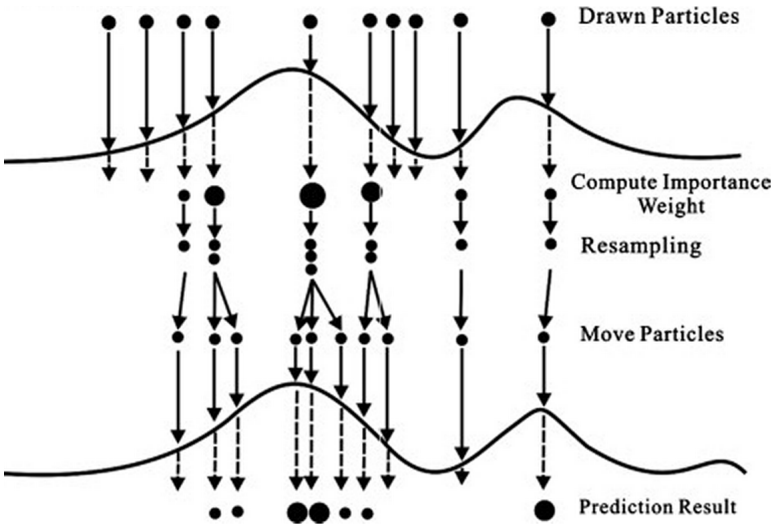

Figure 1: The SIR algorithm, adopted from [1].

6 Bootstrap (SIR)

We can effectively solve the degeneracy problem by resampling. After each iteration $t > 0$ we will resample all N particles from the weighted distribution defined in the previous time step. Thus particles with low-to-negligible weights at t will have a low probability of persisting to $t + 1$. See Figure 1 for a visualization of this process.

6.1 Bootstrap Algorithm

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

6.2 Sampling Impoverishment Problem

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—in the case of SIS and the degeneracy problem—we will have lots of evenly-weighted points concentrated around areas where high-weight points originally occurred. Intuitively, the degeneracy problem leaves you with a diverse scattering of point masses with a few ill-defined peaks while the sampling impoverishment problem leaves you with *only* well defined peaks. 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.

7 Conclusion

References

[1] I. Steinruecken, Christian, "Advanced sampling." http://www.inference.phy.cam.ac.uk/tcs27/talks/sampling.html.

[2] G. Doucet, Arnaud, de Freitas, Nando, *Sequential Monte Carlo Methods in Practice*. Springer-Verlag New York, 2001.

[3] J. Doucet, Arnaud, "A tutorial on particle filtering and smoothing: Fifteen years later." http://www.stats.ox.ac.uk/~doucet/doucet_johansen_tutorialPF2011.pdf, 2008.

[4] E. Orhan, "Particle filtering." http://www.cns.nyu.edu/~eorhan/notes/particle-filtering.pdf, 2012.