Notes: Gibbs Sampling of Images

CS 6190: Probabilistic Modeling

March 29, 2018

Gibbs sampling is a type of Metropolis-Hastings algorithm for sampling a joint distribution $p(x_1, x_2, \ldots, x_d)$. Instead of sampling directly from the joint distribution, the idea is to sample from just a single dimension at a time, using the conditional distribution $p(x_i|x_{-i})$, where x_{-i} are all the $d-1$ dimensions excluding the ith one.

For the homework, $x_1, x_2, \ldots, x_d$ are binary variables representing each pixel in an image[1], with values $-1, +1$. You are given a noisy image with real-valued pixels $y_1, y_2, \ldots, y_d$. Your goal is to use Gibbs sampling to sample from the posterior $p(x|y)$. First, remember you don't need to know how to compute the normalizing constant, so the unormalized pmf is given in the problem:

$$p(x|y) \propto p(x, y) \propto \exp\left(\alpha \sum_i x_i + \beta \sum_{\langle i,j \rangle} x_i x_j - \frac{1}{2\sigma^2} \sum_i (x_i - y_i)^2\right).$$

Here the notation $\langle i, j \rangle$ denotes an edge between the ith and jth nodes. This exists for the 4-neighbors of a pixel (up, down, left, right). The first thing that you want to do is write down the conditional distribution of a single pixel:

$$p(x_i|x_{-i}, y) = p(x_i|N(x_i), y_i),$$

where $N(x_i)$ are the four x neighbors of x_i. Notice this is a Bernoulli distribution - it is just a distribution on a single binary random variable! So, it is very easy to sample from once you have it. Also, notice this conditional distribution will be of the form:

$$p(x_i|N(x_i), y_i) = \frac{\exp(-U_i(x_i))}{\exp(-U_i(+1)) + \exp(-U_i(-1))},$$

where U_i is the part of the energy function U that only includes x_i terms. For this conditional distribution, it must be normalized to be a proper Bernoulli probability, but that only involves summing the two terms in the denominator.

[1] Note that I'm using "flat" indexing of the pixels here, but they of course will really be arranged in a matrix. Also, the notation x with no indices will indicate the vector of all pixels, i.e., $x = (x_1, x_2, \ldots, x_d)$.

The algorithm is finally:

Initialize first image $x^{(0)}$
Repeat for sample $k = 1, 2, \ldots, m$:
    For each pixel $x_i^{(k)}$:
        Sample $x_i^{(k)}$ from $p\left(x_i | N\left(x_i^{(k-1)}\right), y_i\right)$

Important Tricks:

1. **Order.** As you pass through the image, sampling a single x_i pixel at a time, you should not go in order. This can cause artifacts as the pixels left and above of x_i will change before it does, while the pixels right and below will not have changed. Instead, think of the pixel grid as a checkerboard and first pass over all of the "even" pixels (row + column is an even number) and then make a pass over all of the "odd" pixels (row + column is an odd number). You can also go in random order.

2. **Boundaries.** The boundary pixels are a special case because they have a different neighborhood structure (just 3 neighbors on the edge, just 2 neighbors at the corners). There are several ways to handle this. Probably the easiest is to "wrap" at the boundary, which means that the neighbors wrap around to the other side of the image. This involves a modulo width/height operation of the row/column indices. Another possibility is to just fix a one pixel boundary to a constant value and do not update it.

3. **Initialization.** You should be able to start with a completely random binary image (50/50 chance for each pixel)! You might also try thresholding the noisy image as an easier initialization.

4. **Parameters.** Changing the parameters $(\alpha, \beta, \sigma^2)$ will have an effect on your sampling. Remember, α affects the ratio of black and white pixels, $\beta > 0$ affects how strongly pixels want to be like there neighbors, and σ^2 affects how strongly an x pixel wants to be like its noisy y counterpart.

5. **Efficiency.** See if you can code this without using a loop over the pixel indices. You will need at least one loop to do the MCMC iterations.