CS688: Graphical Models - Spring 2016

Assignment 3

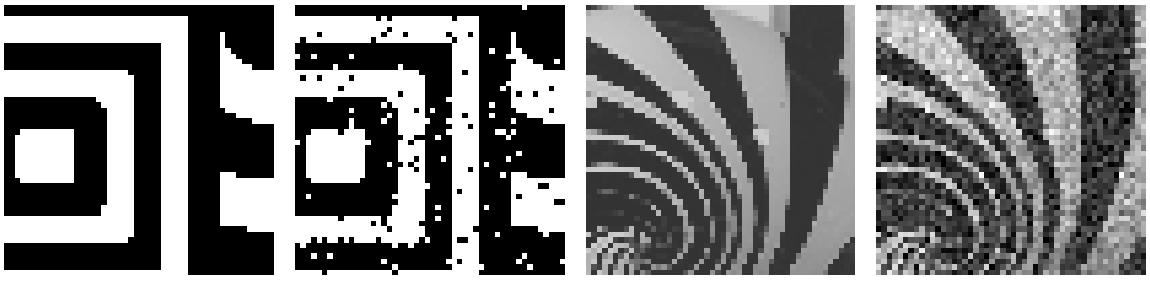
Assigned: Wednesday, Mar 23rd. Due: Wednesday, Apr 6th.

General Instructions: Submit a report with the answers to each question at the start of class on the date the assignment is due. You are encouraged to typeset your solutions. To help you get started, the full LaTeXsource of the assignment is included with the assignment materials. For your assignment to be considered "on time", you must upload a zip file containing all of your code to Moodle by the due date. Make sure the code is sufficiently well documented that it's easy to tell what it's doing. You may use any programming language you like. For this assignment, you may **not** use existing code libraries for inference and learning with CRFs or MRFs. If you think you've found a bug with the data or an error in any of the assignment materials, please post a question to the Moodle discussion forum. Make sure to list in your report any outside references you consulted (books, articles, web pages, etc.) and any students you collaborated with.

Academic Honesty Statement: Copying solutions from external sources (books, web pages, etc.) or other students is considered cheating. Sharing your solutions with other students is also considered cheating. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

Introduction: In this assignment, you will experiment with Monte Carlo image denoising using grid-structured conditional random field models. We will consider both discrete random fields for binary images and Gaussian random fields for real-valued images.

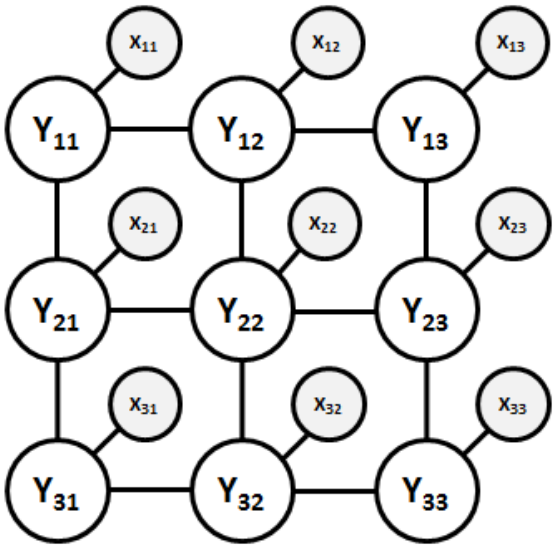
Data Set: The data for this assignment consists of two 50×50 pixel images. The *stripes* image is a binary image for use with the discrete model with pixel values that are either 0 or 1. The original image is *stripes.txt*, while the copy with noise added in *stripes-noise.txt*. The *swirl* image is a real-valued image with pixels values in the range $[0, 1]$ for use with the Gaussian model. The original image is *swirl.txt*, while the copy with noise added in *swirl-noise.txt*. The four images are shown below. We will refer to the images without noise as the "true" images and the images with noise as the "observed" or "noisy" images.



(a) stripes (b) stripes-Noise (c) swirl (d) swirl-Noise

Models: We will consider a grid-structured CRF model with two sets of variables $\mathbf{Y}$ and $\mathbf{X}$. X_{ij} will

represent the value of pixel (i, j) in the observed, noisy image. Y_{ij} will represent the de-noised value of pixel (i, j). We will denote the pixels of the true image by I_{ij}. We will let W represent the width of the image and H represent the height. The CRF model connects each pixel variable Y_{ij} in the de-noised image to it's nearest neighbors in the de-noised image. Its nearest neighbors are the pixels directly above, below, left and right of location (i, j). Pixels on the border of the image will have two or three neighbors, while pixels off the border of the image will have four neighbors. Each de-noised pixel variable Y_{ij} is also connected to the corresponding pixel in the noisy image X_{ij}. Each edge in the graph is associated with a pairwise potential. The graph structure is shown below.



We let V denote the set of pixel locations in the image and E denote the set of neighboring pixel locations. Since a pixel location is a pair (i, j), the set V contains pairs (i, j). E thus contains pairs of pixel locations $(i, j), (k, l)$ that are neighbors in the graph.

The conditional distribution over de-noised pixel values given noisy pixel values for the binary case is given below. The set $\mathcal{Y}$ is the joint domain of the de-noised pixel variables Y_{ij}. For a binary image of size $W \times H$, this is $\{0, 1\}^{W \times H}$.

$$P(\mathbf{Y} = \mathbf{y}|\mathbf{x}) = \frac{\exp\left(\sum_{(i,j),(k,l)\in E} W^P_{ijkl}[y_{ij} = y_{kl}] + \sum_{(i,j)\in V} W^L[y_{ij} = x_{ij}]\right)}{\sum_{\mathbf{y}\in\mathcal{Y}} \exp\left(\sum_{(i,j),(k,l)\in E} W^P_{ijkl}[y'_{ij} = y'_{kl}] + \sum_{(i,j)\in V} W^L[y'_{ij} = x_{ij}]\right)} \tag{1}$$

The conditional distribution over de-noised pixel values given the noisy pixel values for the Gaussian case is given next. The set $\mathcal{Y}$ is the joint domain of the de-noised pixel variables Y_{ij}. For an image of size $W \times H$ with real-valued pixels, this is $\mathbb{R}^{W \times H}$.

$$P(\mathbf{Y} = \mathbf{y}|\mathbf{x}) = \frac{\exp\left(-\sum_{(i,j),(k,l)\in E} W^P_{ijkl}(y_{ij} - y_{kl})^2 - \sum_{(i,j)\in V} W^L(y_{ij} - x_{ij})^2\right)}{\int_{\mathcal{Y}} \exp\left(-\sum_{(i,j),(k,l)\in E} W^P_{ijkl}(y'_{ij} - y'_{kl})^2 - \sum_{(i,j)\in V} W^L(y'_{ij} - x_{ij})^2\right) d\mathbf{y}'} \quad (2)$$

Both models trade off smoothness in the de-noised image (the first terms) against the degree to which the de-noised image looks like the noisy image (the second terms). When the pairwise weights W^P_{ijkl} are large, the model favors smoothness, when the local weight W^L is large, the model favors being close to the noisy image. In the most general case, we can consider a different amount of smoothing between every pair of de-noised variables Y_{ij} and Y_{kl} as given by W^P_{ijkl}. To begin, we will consider the special case where all the W^P_{ijkl}'s are equal to the same value W^P.

Algorithm: To de-noise the observed image $\mathbf{X}$, we perform inference in the CRF model to compute the posterior mean of each de-noised pixel variable Y_{ij} conditioned on the noisy pixel values $\mathbf{x}$. The graph is strongly loopy, so we will use a particle-based approximation. Specifically, we will use T iterations of the Gibbs sampler. The Gibbs sampler operates by cycling over each variable Y_{ij} and sampling a new value for that variable from its conditional distribution given the current values of the other variables in its Markov blanket. The Markov blanket of a variable Y_{ij} consists of it's neighbors Y_{kl} in the grid, and the noisy pixel variable X_{ij}.

For the binary model, the conditional distribution is easy to obtain from first principles. We use $\mathbf{A}_{ij}$ to denote the set of neighbors of Y_{ij} in the grid and $\mathbf{y}_{\mathbf{A}_{ij}}$ to denote the values of the neighbors.

$$P(Y_{ij} = 1|\mathbf{y}_{\mathbf{A}_{ij}}, x_{ij}) = \frac{\exp\left(\sum_{(k,l)\in A_{ij}} W^P[y_{kl} = 1] + W^L[x_{ij} = 1]\right)}{\sum_{y=0}^{1} \exp\left(\sum_{(k,l)\in A_{ij}} W^P[y_{kl} = y] + W^L[x_{ij} = y]\right)} \quad (3)$$

The Gibbs sampler starts by assigning an initial value to each Y_{ij}. We can simply set $y_{ij} = x_{ij}$. On each iteration t, the Gibbs sampler sweeps through all variables Y_{ij}, updating their values one at a time. The Gibbs sampler updates the value of Y_{ij} by drawing a random sample from the conditional distribution $P(Y_{ij} = 1|\mathbf{y}_{\mathbf{A}_{ij}}, x_{ij})$, which is a simple Bernoulli distribution. We can do this by drawing a uniform random number α on $[0, 1]$ and setting $y_{ij} = 1$ if $\alpha < P(Y_{ij} = 1|\mathbf{y}_{\mathbf{A}_{ij}}, x_{ij})$ and setting $y_{ij} = 0$ otherwise. After sweeping through all pixel locations on iteration t, we set $y^t_{ij} = y_{ij}$ to save the sample. The estimate for the de-noised pixel values after t iterations is the particle approximation to the posterior mean given by $\bar{y}_{ij} = \frac{1}{t}\sum_{s=1}^{t} y^s_{ij}$.

In the Gaussian case, the conditional distribution of Y_{ij} given $\mathbf{y}_{\mathbf{A}_{ij}}$ and x_{ij} can also be obtained in closed form, although the derivation is somewhat more difficult.

$$P(Y_{ij} = y|\mathbf{y}_{\mathbf{A}_{ij}}, x_{ij}) \propto \exp\left(-\sum_{(k,l)\in A_{ij}} W^P(y_{kl} - y)^2 - W^L(x_{ij} - y)^2\right) \quad (4)$$

We can see that up to normalization, the conditional distribution is the exp of the negative of a sum of several quadratic terms in involving y. This is exactly the form of an unnormalized univariate normal distribution, so we need only work out the mean and standard deviation of the distribution. We let N_{ij} denote the number of neighbors of Y_{ij} in the grid, expand the quadratic forms to identify the coefficients of y^2 and y, and drop additional terms that don't depend on y.

$$P(Y_{ij} = y|\mathbf{y}_{\mathbf{A}_{ij}}, x_{ij}) \propto \exp\left(-y^2(W^P N_{ij} + W^L) + 2y\left(W^L x_{ij} + \sum_{(k,l)\in A_{ij}} W^P y_{kl}\right)\right) \tag{5}$$

In a normal distribution with mean μ_{ij} and standard deviation σ_{ij}, we have that:

$$\mathcal{N}(y|\mu_{ij}, \sigma_{ij}) \propto \exp\left(-\frac{1}{2\sigma_{ij}^2}(y - \mu_{ij})^2\right) \propto \exp\left(-y^2\frac{1}{2\sigma_{ij}^2} + y\frac{1}{\sigma_{ij}^2}\mu_{ij}\right) \tag{6}$$

By matching the coefficients in the two distributions, we can see that:

$$-y^2\frac{1}{2\sigma_{ij}^2} = -y^2(W^P N_{ij} + W^L) \tag{7}$$

$$\sigma_{ij}^2 = \frac{1}{2(W^P N_{ij} + W^L)} \tag{8}$$

$$\tag{9}$$

$$y\frac{1}{\sigma_{ij}^2}\mu_{ij} = 2y\left(W^L x_{ij} + \sum_{(k,l)\in A_{ij}} W^P y_{kl}\right) \tag{10}$$

$$\mu_{ij} = 2\sigma_{ij}^2\left(W^L x_{ij} + \sum_{(k,l)\in A_{ij}} W^P y_{kl}\right) \tag{11}$$

$$= \frac{1}{W^P N_{ij} + W^L}\left(W^L x_{ij} + \sum_{(k,l)\in A_{ij}} W^P y_{kl}\right) \tag{12}$$

We apply the Gibbs sampler to the Gaussian CRF model in exactly the same way that we apply it to the discrete CRF model. We initialize all of the Y variables to $y_{ij} = x_{ij}$. On each iteration t we sweep through all pixel locations (i, j) in the image. For each pixel location, we sample y_{ij} from its conditional distribution given the current values of the neighboring variables in the grid y_{kl}, $(k, l) \in A_{ij}$ and the noisy observed pixel x_{ij}. As shown above, this conditional distribution on y_{ij} is a univariate normal distribution. We use the formulas above to compute the mean μ_{ij} and variance σ_{ij}^2 of this conditional distribution. To sample a value for y_{ij}, we first use a library function to draw a random standard normal value $z \sim \mathcal{N}(z|0, 1)$. We then set $y_{ij} = \mu_{ij} + z\sqrt{\sigma_{ij}^2}$, which is a draw from $\mathcal{N}(y_{ij}|\mu_{ij}, \sigma_{ij}^2)$. After sweeping through all pixel locations on iteration t, we set $y_{ij}^t = y_{ij}$ to save the sample. Our estimate for the de-noised pixel values after t iterations is again the particle approximation to the posterior mean given by $\bar{y}_{ij} = \frac{1}{t}\sum_{s=1}^{t} y_{ij}^s$.

1. (*5 points*) **Derivation for Binary Image Model:** For the discrete binary model, derive Equation (3) from Equation (1). In other words, derive the Gibbs sampling conditional probability from the full model.

2. (*45 points*) **Monte Carlo De-noising for Binary Images:** Implement the Gibbs sampling procedure for the binary CRF model as described above and use your implementation to perform the following exercises. We recommend creating a "single Gibbs sweep" function that iterates once through the pixels, resampling each one in turn based on the parameter settings and the noisy image matrix. To debug this, run it once, then plot the current sample, and do this a few more times to make sure it's updating as you expect. (e.g. *plt.imshow(curY, cmap='gray')* in Python.)

(a) Qualitative analysis: quickly play around with the model. This helps intuition and debugging. (Jupyter/IPython notebook or another interactive environment may be useful for this.)
Initialize with the noisy image and run one sweep (or a few sweeps if you like). Describe qualitatively what happens under different parameter settings, show an example image output, and explain *why* it's happening. Try at least these different scenarios. Report the parameter weights with each image output.

- What happens when the parameters are zero?
- What happens when both are positive?
- What happens when one or both are negative? (You should be able to make some crazy-looking ones by doing this.)

(b) Posterior marginal decoding. Compute the posterior mean image $\bar{y}$ using the $T = 50$ samples from your Gibbs sampler. Compute the mean absolute error between the pixels in the posterior mean image $\bar{y}$ and the pixels in the true *stripes* image I. The MAE is defined as $(1/HW) \sum_{(i,j)} |\bar{y}_{ij} - I_{ij}|$.

Try different values of W^P and W^L and determine values that give good performance in terms of MAE. Report the values of the parameters W^P and W^L that you find and the corresponding MAE, and also report the MAE baseline of just predicting the noisy image. (Make sure you can beat this baseline.)
Display the best posterior mean image $\bar{y}$ that you find, along with posterior mean images for settings of W^P and W^L that resulted in over and under-smoothing.
(Implementation note: you can make a running sum at each iteration, or, since this problem is so small, you can just save the entire image sample at each iteration and analyze it afterwards: make your "run the sampler for a bunch of iterations" function return a list of the entire sample history.)

(c) Using the best parameters you found in part **(b)**, again run the Gibbs sampler until the MAE converges. On each iteration, compute the t-step posterior mean image $\bar{y}$ using samples 1 to t. Compute the MAE between the t-step posterior mean image and the true image I. Produce a plot showing the MAE of the t-step posterior mean images versus t. How many iterations does it take for the posterior mean's MAE to converge?

(d) Initialization and traceplots. We'll make the following type of traceplot: For every iteration, plot the fraction of pixels that are turned on. Eventually this should "converge", meaning it bounces around a region corresponding to the Markov chain steady state.
For a single setting of parameters, run two MCMC chains: one initialized with the noisy image, and another initialized with all zeros. Plot these two different timeseries as two different lines on the same graph. They should start in different positions, then MCMC theory says they should converge to bouncing around in the same area.
Find two settings of parameters: one where the different initializers make a big difference in convergence time, and one where initialization does not make as large a difference. Show the two traceplots. There

should one graph for each parameter setting, and within each graph, two lines for the two different chains. (We've found this is a reasonable way to visually compare, but feel free to use a different plotting approach if you can illustrate the effects more clearly.)

Explain why they have different sensitivity to initialization.

(e) Burn-in: As we discussed, a way to get rid of the bias in the initialization is to throw out the results of the first B (burn-in) iterations of an MCMC sampler. Is this important for this model and why?

3. (*45 points*) **Monte Carlo De-noising for Real-Valued Images:** Implement the Gibbs sampling procedure for the Gaussian CRF model as described above, and:

(a) Apply your Gibbs sampler to the *swirl-noise* image and identify values for W^P and W^L that give good de-noising performance in terms of MAE. Report the parameters and the corresponding MAE value. Display the posterior mean image $\bar{y}$ found using these parameters.

(b) As we will see shortly, the general case of a different pairwise weight W^P_{ijkl} for every pair of neighboring pixel locations (i, j) and (k, l) is very useful. Re-derive the mean and variance of the conditional distribution on y_{ij} using the more general form of the weights W^P_{ijkl}. Report the modified mean and variance formulas. Show some work.

(c) One of the main problems with the basic Gaussian model is that it smooths strong image edges along with noise. Intuitively, when the squared difference between the noisy pixel values $(x_{ij} - x_{kl})^2$ is large, we don't want the model to assert that $(y_{ij} - y_{kl})^2$ should be small. We can achieve this effect by setting the general pairwise weights W^P_{ijkl} to $\frac{W^P}{0.01+(x_{ij}-x_{kl})^2}$, where W^P is a global pairwise weight parameter. Implement the Gibbs sampler using general pairwise weights W^P_{ijkl} using the mean and variance you found in **(b)** Explore values of W^P and W^L, computing the general pairwise weights W^P_{ijkl} using the above formula with each value of W^P. Identify values of W^P and W^L that perform well on *swirl-noise* in terms of MAE. Report the MAE, the values of the weights and display the de-noised image. Do you obtain better MAE with the basic or more general model? Which model leads to de-noised images that are visually less noisy?

4. (*5 points*) **Parameter learning pseudocode:** How could you use the Gibbs sampler to support parameter learning in one of the above models? Assume you have access to a training set so you can do supervised learning. We've learned many different high level algorithms for this at this point for parameter learning; each of these would use the Gibbs sampler you already wrote within the inner loop.

Write out pseudocode to parameter learning for two different algorithms: (a) MLE with gradient descent, and (b) Bayesian learning with MCMC (for example, Metropolis-Hastings with a random walk proposal, or use another MCMC algorithm if you like). In both cases, the innermost part of the algorithm calls out to the Gibbs sampler as a black-box. Be clear what are the inputs and outputs of the pseudocode functions that you define and call. When you introduce new variables not already defined in this assignment, define them in a reasonably precise manner.

We don't care about convergence (just run a fixed number of iterations). We do care about when the sampler gets called, what gets returned from or is calculated from the sampler, how the parameter updates get calculated, and what the algorithm returns as a final result.