Graphic Models Homework 3

Monte Carlo CRF image de-noising

Joshua Trask

March 16, 2012

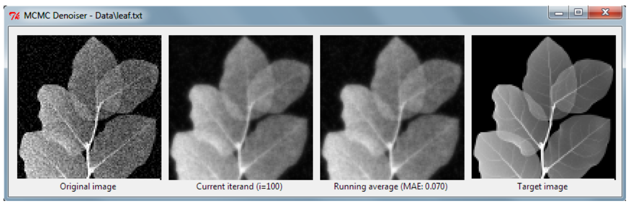

Figure 1: Screenshot from the companion software. We smooth the noisy image (far left) to reproduce the target (far right.) The estimate (second from right) is given after 100 samples.

In this report, we consider a CRF model for removing noise from 2D images. Each pixel depends on its adjacent neighbors, so the resulting Markov network is extremely loopy. Thus we consider an MCMC implementation for approximate inference, using the Gibbs sampler.

We first use a simple model for the discrete case in which each pixel takes on one of only two values (black or white,) and then generalize to the continuous greyscale case to evaluate two additional models. This third model is an extension to the basic continuous model which uses different pairwise weights at each transition so as to avoid smoothing between pairs of pixels which were not close together in color originally.

The companion Python software animates the progress of the de-noising algorithm and reports the error against the target image.

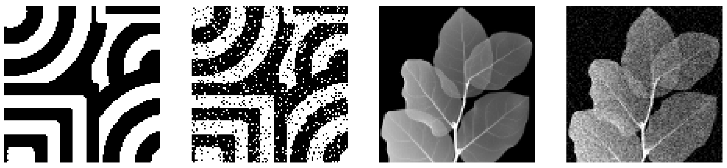

Figure 2: Test images, left-to-right: black-and-white 'stripes' image (target, noisy,) greyscale 'leaf' image (target, noisy.) The original images have MAE of 0.096 and 0.047, respectively.

1 Monte Carlo De-noising for Binary Images

We attain excellent results for the 'stripes' image in the discrete case using parameters $W^P = 4.84$ and $W^L = 3.55$.[1] This corresponds to an invocation of the companion software by

```
denoise.py Data/stripes.txt -m Discrete -P 4.84 -L 3.55
```

The result, with mean average error 0.011, is shown in Figure 3 alongside some results from less successful parameter selections.

t-step convergence results for these parameter selections are presented in Figure 4. We can see that the under-smoothing process converges almost immediately to an MAE of 0.047, since the model distribution is dominated by the pixel factors and little consideration is made for the changes from the previous iteration. The 'good' results improve dramatically for the first ten iterations, and then slowly converge to a lower MAE. The over-smoothed results also improve quickly at first as noise is aggressively smoothed from the image, but after this initial improvement, the MAE actually increases as blurring artifacts accumulate.[2] The MAE in fact does not converge in 100 iterations, but continues degrading for several thousand iterations. A drastically over-smoothed iterand is shown in Figure 5.

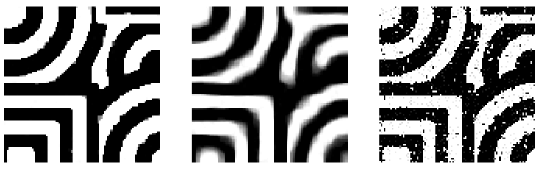

Figure 3: Left-to-right: high-quality results ($MAE = 0.011$) taking $W^P = 4.84$ and $W^L = 3.55$; over-smoothed results ($MAE = 0.105, W^P = 100, W^L = 0.001$;) under-smoothed results ($MAE = 0.047, W^P = 3, W^L = 10$.) Both the over- and under-smoothed results are significantly less noisy than the original image, but the under-smoothed results still show some noise, and the over-smoothed results introduce dramatic blurring artifacts, so that the resulting image in fact has higher MAE than the original.

[1]The note in Appendix B explains how these and other high-quality parameters were found.

[2]We might hypothesize that there is a common cause for the stabilization of the 'good' results after approximately ten iterations and for the increasing error in the over-smoothed case which also begins after about ten iterations. In the first iterations we can think of probabilistic influence as starting to radiate from each pixel, so it is unsurprising that quality would improve at first (since this influence is the basis for our de-noising,) but then degrade as the influence 'reaches' further in precisely the model which over-values its contribution.

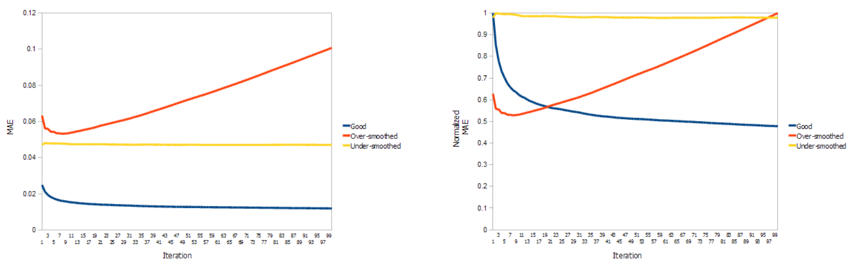

Figure 4: *Left:* t-step MAE for the parameters used in Figure 3, *Right:* t-step MAE, normalized to the maximum iterand MAE.

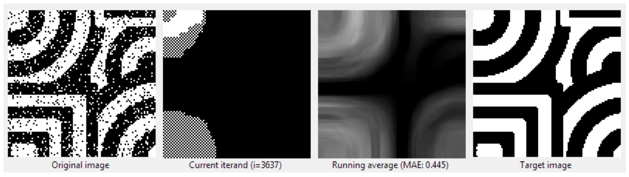

Figure 5: 3637-step iterand for the over-smoothing parameters.

2 Monte Carlo De-noising for Real-Valued Images

We also find improved image quality in the continuous case, which in the companion software can be invoked by

```
denoise.py Data/leaf.txt -m Continuous1 -P 1 -L 200
```

using 'Continuous1' for the basic Gaussian de-noiser and 'Continuous2' for the modified version that tries to respect hard edges. Derivations of the mean and variance for the modified 'Continuous2' model are presented in Appendix A.

High-quality parameters were found for the basic Gaussian model as described in Appendix B, and the de-noised image results for these parameters are presented for both continuous models in Figure 6. Figure 7 shows the de-noised images from both models when we instead use parameters which minimize error in the modified model. We are able to attain a better MAE in the modified model,[3] and although the modified model performs significantly worse (in terms of MAE) using the parameters optimized for the basic model, the image quality actually seems much better visually. One advantage we can see in the modified model is near-perfect de-noising of the black background, which is still noisy in the basic model.

[3]But again, please see the note in Appendix B.

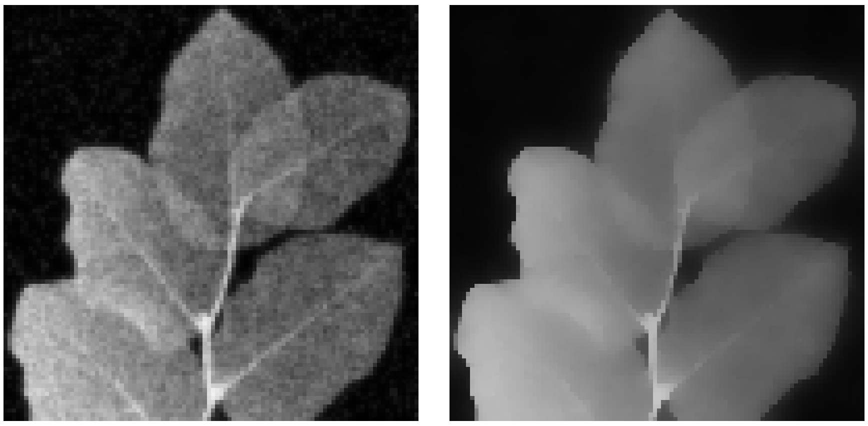

Figure 6: De-noised results for the basic Gaussian model (left, $MAE = 0.036$) and modified model (right, $MAE = 0.047$) using parameters $W^P = 5213$ and $W^L = 14374$ optimized for the basic model.

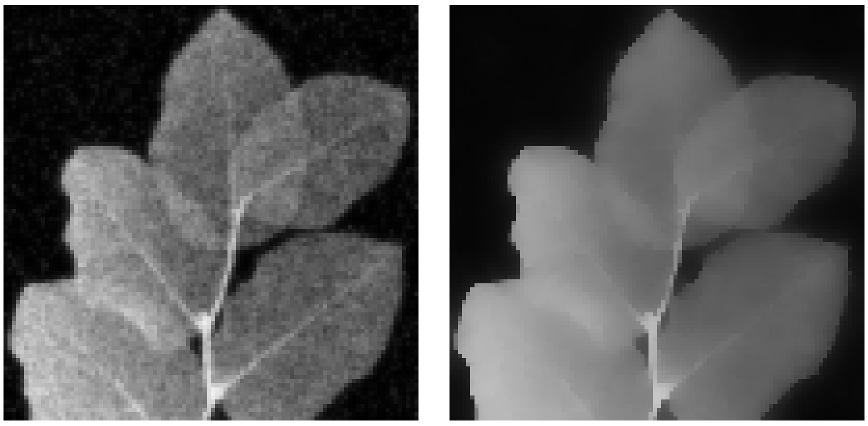

Figure 7: De-noised results for the basic Gaussian model (left, $MAE = 0.044$) and modified model (right, $MAE = 0.030$) using parameters $W^P = 863$ and $W^L = 47809$ optimized for the modified model.

A Derivation of modified Gaussian parameters

The derivations in this section were checked verbatim by e-mail on March 13.

We can generalize the parameterization of our pairwise weights W^P to use different values W^P_{ijkl} for each pair of neighboring pixels (i,j) and (k,l). Then we can give the conditional distribution of Y_{ij} given its neighbors $\mathbf{y_{A_{ij}}}$ by modifying Equation (4) from the assignment sheet:

$$P(Y_{ij} = y|\mathbf{y_{A_{ij}}}) \propto \exp\left(-\sum_{(k,l)\in A_{ij}} W^P_{ijkl}(y - y_{kl})^2 - W^L(y - x_{ij})^2\right). \tag{1}$$

Again we can group terms as coefficients on a second-order polynomial in y, and drop terms which do not depend on y:

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

Matching with terms in the normal distribution, we must have

$$-y^2 \frac{1}{2\sigma^2_{ij}} = -y^2(W^L + \sum_{(k,l)\in A_{ij}} W^P_{ijkl}) \tag{3}$$

and

$$y\frac{1}{\sigma^2_{ij}}\mu_{ij} = 2y(W^L x_{ij} + \sum_{(k,l)\in A_{ij}} W^P_{ijkl} y_{kl}). \tag{4}$$

From Equation (3) we obtain

$$\sigma^2_{ij} = \frac{1}{2(W^L + \sum_{(k,l)\in A_{ij}} W^P_{ijkl})}, \tag{5}$$

and from Equation (4) we have

$$\begin{aligned}\mu_{ij} &= 2\sigma^2_{ij}(W^L x_{ij} + \sum_{(k,l)\in A_{ij}} W^P_{ijkl} y_{kl}) \\ &= \frac{1}{W^L + \sum_{(k,l)\in A_{ij}} W^P_{ijkl}}\left(W^L x_{ij} + \sum_{(k,l)\in A_{ij}} W^P_{ijkl} y_{kl}\right).\end{aligned} \tag{6}$$

B A note on parameter optimization

After trying several sets of parameters by hand, we investigated automated approaches from SciPy for optimizing the parameters in each model. Due to operator inexperience, the automated techniques described below are fairly alchemical, but are included here for completeness.

We first tried standard local search techniques including downhill simplex and conjugate gradient (with estimated derivatives.) These search strategies found high-quality parameter selections, but seemed to identify nearly-level sets where the MAE could be decreased very slightly as the parameters moved further from zero. The resulting large parameter values may cause issues with numerical accuracy, and do not instill confidence that they are truly optimal.

Much better results were obtained using global search techniques, and in particular the results found by simulated annealing gave comparable performance using much more reasonable-looking parameters. By observing the iterand performance of a simulated annealing session, we can compare the optimal results to many other samples in the parameter space. Finally, we see some improvement by running a downhill simplex algorithm to tune the parameters starting at the best choice found by simulated annealing. We should not expect that these parameters are truly optimal; rather, they are among the higher-quality results found during the searches described, and they use parameter values that do not seem too excessive. However, this does mean that a truly optimal set of parameters may allow the basic Gaussian model to outperform the modified model.

One aspect of this process I was uncertain about is the behavior of these algorithms when optimizing a stochastic function. In particular, this seemed likely to throw off gradient estimation for local search methods. We can see that in the 'optimized' parameters the variance is extremely low, and in fact it is even lower in the divergent parameters identified by local search. We hypothesize that this may confer better-behaved estimates for the optimizer, since it reduces randomness. We also hypothesize that this decrease in randomness is likely to overfit our models to the respective images. It may be possible to limit randomness that reaches the optimizer by averaging together multiple MAE samples for each parameter evaluation, but we should still expect the optimizer to prefer overfitting.

Inspired by this observation, we sought a final set of parameters to minimize the combined MAE when applied to both models. At the time of writing, suitable parameters had not been found for this experiment — we note, however, that the optimizer was (as hypothesized) dwelling at parameter settings with much lower variance, and among nearly all such settings, the modified model had lower MAE.