CS228 Homework 3 Solutions

Instructor: Stefano Ermon – ermon@stanford.edu

Available: 02/03/2017; Due: 02/17/2016

---

1. **[4 points]** *(MAP and MPE)* Show that marginal MAP assignments do not always match the MPE assignments (Most Probable Explanation). I.e., construct a Bayes net such that the most likely configuration of all variables does not agree with the most likely assignment to a single variable based on its marginal probability (the value that maximizes the marginal probability of a single variable, i.e. after marginalizing out the remaining ones).

2. **[15 points]** *(Variable Elimination)* Suppose we wish to perform exact inference over a chain Markov Random Field given by $X_1 - X_2 - ... - X_n$. Assume that each variable X_i has $|Val(X_i)| = d$.

   (a) **[5 points]** Derive an $O(n^3d^3)$ algorithm for computing marginals $P(X_i, X_j)$ over all n^2 variable pairs X_i,X_j.

   Answers: Suppose $j > i$. To compute $P(X_i, X_j)$, eliminate variables $1..i$, with cost $O(id^2)$. then eliminate variables $j..n$, with cost $O((n-j)d^2)$. then eliminate variables $i..j$, with cost $O((j-i)d^3)$. The largest factor generated has three nodes. Therefore, the complexity for each query is $O(nd^3)$, assuming each of $X_1, \cdots, X_n$ has domain size d. Since we need to run the query $n = O(n^2)$ times, the total time complexity is $O(n^3d^3)$.

   (b) **[3 points]** Since we are computing marginals for all variable pairs, we can store computations done for the previous pairs and use them to save time for the computations of the remaining pairs. The key recursive relationship that makes this work is the following equation (for $i < j - 1$):

$$P(X_i, X_j) = \sum_{X_{j-1}} P(X_i, X_{j-1}) P(X_j | X_{j-1}) \tag{1}$$

   Prove that the equation above holds.

   Answers: Due to the conditional independence properties implied by the network, we have that, for $i < j - 1$,

$$\begin{aligned} P(X_i, X_j) &= \sum_{X_{j-1}} P(X_i, X_{j-1}, X_j) \\ &= \sum_{X_{j-1}} P(X_i, X_{j-1}) P(X_j | X_i, X_{j-1}) \\ &= \sum_{X_{j-1}} P(X_i, X_{j-1}) P(X_j | X_{j-1}) \end{aligned} \tag{2}$$

   (c) **[7 points]** Construct a dynamic programming algorithm that computes marginals over all n^2 variable pairs based on the recursive relation in (1), and achieves a running time that is asymptotically faster than $O(n^3d^3)$. Describe the time complexity of your algorithm in terms of n and d. Note: Make sure to clearly specify how each of the probabilities P you use are computed.

   Answers: The term $P(X_j|X_{j-1})$ can be computed directly from the marginals in the calibrated clique tree, while $P(X_i, X_{j-1})$ is computed and stored from a previous step if we arrange the computation in a proper order.

3. **[8 points]** (**Clique tree calibration**) Suppose that we have a clique tree over a set of factors $\mathcal{F}$ with cliques $C_1, ..., C_N$, which we have calibrated using sum-product message propagation so that we have all messages $\delta_{i \to j}$.

   (a) **[4 points]** If we modify a factor in some clique C_i, which message updates do we have to perform to recalibrate the tree?

   Answers In order to recalibrate the tree, we have to recalculate all of the messages from C_i out to the leaves of the tree (i.e. the distribute pass, if we think of C_i as the root of our clique tree, and the incoming messages as the result of the collect pass.). Note that the messages passed to C_i need not change.

   (b) **[4 points]** If we modify a factor in some clique C_i, but we just want the marginal over a single pre-specified variable X_k, which message updates do we have to perform?

   Answers One can think of this in two cases (although the second one sufficient):

   i. $X_k \in C_i$**:** In this case we don't need to do any message passing updates; we need only divide out the old factor form the beliefs, multiply in the new factor, and marginalize.

   ii. $X_k \notin C_i$**:** Here, we have to pass messages from C_i to the first clique containing X_k. Once this clique gets the message, we can marginalize to get X_k

4. **[18 points]** (**Importance Sampling**) Suppose we have a distribution $P(\boldsymbol{X}, \boldsymbol{E})$ over two sets of variables $\boldsymbol{X}$ and $\boldsymbol{E}$. Our distribution is represented by a nasty Bayes Net with very dense connectivity, and our sets of variables $\boldsymbol{X}$ and $\boldsymbol{E}$ are spread arbitrarily throughout the network. In this problem our goal is to use the sampling methods we learned in class to estimate the posterior probability $P(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{E} = \boldsymbol{e})$. More specifically, we will use a tree-structured Bayes Net as the proposal distribution for use in the importance sampling algorithm.

   (a) **[4 points]** For a particular value of $\boldsymbol{x}$ and $\boldsymbol{e}$, can we compute $P(\boldsymbol{x} \mid \boldsymbol{e})$ exactly, in a tractable way? Can we sample directly from the distribution $P(\boldsymbol{X} \mid \boldsymbol{e})$? Can we compute $P'(\boldsymbol{x} \mid \boldsymbol{e}) = P(\boldsymbol{x}, \boldsymbol{e})$ exactly, in a tractable way? For each question, provide a Yes/No answer and a single sentence explanation or description.

   Answers No, No, Yes

   (b) **[14 points]** Now, suppose your friendly TAs have given you a tree network (each variable besides the root has exactly one parent) that defines a distribution Q. They tell you that $Q(\boldsymbol{X}, \boldsymbol{E})$ is "close" to the distribution $P(\boldsymbol{X}, \boldsymbol{E})$ of the nasty network. You now want to use *the posterior* in Q as your proposal distribution for importance sampling. You now must perform the two steps of importance sampling:

   i. Show how to sample from the posterior in Q. More specifically, describe an algorithm for drawing samples $\boldsymbol{x}[m]$ *exactly* from $Q(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$ using only tractable techniques. You answer should be at the level of pseudocode, and should indicate the specific distribution that each variable will be sampled from, and an explanation of how you computed that distribution.

   ii. Now you must reweight the samples according to the rules of importance sampling. You want your weighted samples to accurately represent the actual posterior in the original network $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$. Show precisely how you determine the weights $w[m]$ for the samples.

   iii. Show the form of the final estimator $\hat{P}(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{E} = \boldsymbol{e})$ for $P(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{E} = \boldsymbol{e})$, in terms of the samples from part i, and the weights from part ii.

   Answers Create a clique tree where each clique is a pair of variables (child and parent). Multiply in the indicator functions for the evidence $\boldsymbol{E} = \boldsymbol{e}$. The distribution across the tree now represents $Q(\boldsymbol{X}, \boldsymbol{E} \mid \boldsymbol{E} = \boldsymbol{e})$. Calibrate the tree. Now, the belief at a clique over $(X, \mathbf{Pa}_X)$ is proportional to $Q(X, \mathbf{Pa}_X \mid \boldsymbol{E} = \boldsymbol{e})$. From this belief, we can easily compute $Q(X \mid \mathbf{Pa}_X, \boldsymbol{E} = \boldsymbol{e})$ (use Bayes' Rule). Using these CPDs, we can now forward sample directly from the posterior in Q (sample the first variable, then instantiate it in neighboring cliques, repeating to forward sample).

   To get weights:

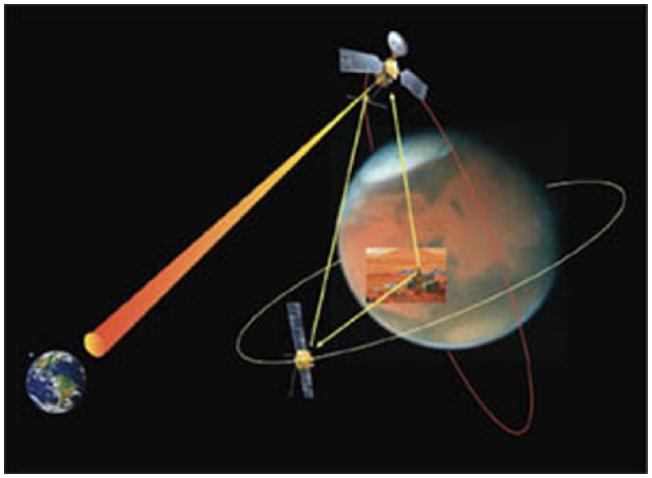
$$w[m] = \frac{P(\boldsymbol{x}[m], \boldsymbol{e}[m])}{Q(\boldsymbol{x}[m], \boldsymbol{e}[m] \mid \boldsymbol{e})}$$

The estimator:

$$\hat{P}(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{E} = \boldsymbol{e}) = \frac{\sum_{m=1}^{M} w[m] 1\{\boldsymbol{x}[m] = \boldsymbol{x}\}}{\sum_{m=1}^{M} w[m]}$$

[55 points] Programming Assignment [1]

In this programming assignment, you will design algorithms for reliable communication in the presence of noise. We will learn state-of-the-art techniques that some NASA missions use to communicate from deep space, and in the process help The Martian return home. [2]



The Mars Rover is trying to communicate with mission headquarters on Earth. The message the Mars Rover wants to transmit is a binary sequence $X \in \mathbf{B}^N$ of N bits, representing an image. Here $\mathbf{B} = \{0, 1\}$.

To deal with transmission noise, the Mars Rover appends N redundant bits to each message to allow for error correction (using an error-correcting code). The redundant bits are chosen using a clever scheme: the message is *encoded* as $Y = GX$ through a special matrix $G \in \mathbf{B}^{2N \times N}$, a generator matrix [3] that you can treat as a "constant" for the purposes of this assignment. G is chosen such that the first N bits of Y are equal to X, while the remaining N bits are redundant "parity checks" (here GX is computed modulo 2, so that $Y \in \mathbf{B}^{2N}$). The *encoded* message $Y \in \mathbf{B}^{2N}$ obtained this way is special because it is a *codeword*: the Mars Rover and mission headquarters have agreed that all *valid messages or codewords* are such that $HY = \mathbf{0}$ where H is a pre-specified binary *parity check matrix* $H \in \mathbf{R}^{N \times 2N}$. The matrices G and H are paired, and chosen so that multiplying by G creates valid messages (codewords), and H can be used to check for errors in a received message (on Earth). Mathematically, $HG = 0$, so that $HY = HGX = \mathbf{0}$ for any input message X.

This codeword $Y \in \mathbf{B}^{2N}$ is then transmitted through deep space back to the mission control on Earth who then receives it as the noisy $\widetilde{Y}$. The decoding process refers to the procedure of recovering the ground truth codeword Y (and thus also X, the first N bits of Y) from the noisy version $\widetilde{Y}$. We will focus on error correcting codes based on highly sparse, low density parity check (LDPC)[4] matrices H, and use the sum-product variant of the loopy belief propagation (BP) algorithm to estimate partially corrupted message bits[5], and to bring our Martian safely back home.

To represent the problem using an undirected graphical model, there are two sets of factors you need to consider. The first are the unary factors associating Y_i with $\widetilde{Y}_i$ (messages that are similar to the one received are more likely). You also need to include the parity checks, which are factors defined on Y (assigning zero probability to messages that are not valid codewords) which depend on H.

LDPC codes are specified by a binary parity check matrix $H \in \mathbf{R}^{N \times 2N}$, whose columns correspond to codeword bits, and rows to parity check constraints. We define $H_{ij} = 1$ if parity check P_i depends on

[1] Assignment adapted from Brown University CS242 instructed by Erik Sudderth

[2] https://scienceandtechnology.jpl.nasa.gov/research/research-topics-list/communications-computing-software/deep-space-communications

[3] Generated by Neal's LDPC software `http://www.cs.utoronto.ca/~radford/ldpc.software.html`

[4] For optional background information on LDPC codes, see Chap. 47 of MacKay's *Information Theory, Inference, and Learning Algorithms*, which is freely available online: `http://www.inference.phy.cam.ac.uk/mackay/itila/`.

[5] If you are curious, Chapter 47 also provides some ideas to speed up loopy belief propagation updates for LDPC codes (Equations 47.9 and 47.10). See also lecture 4.

codeword bit Y_j, and $H_{ij} = 0$ otherwise. Valid codewords are those for which the sum of the bits connected to each parity check, as indicated by H, equals zero in modulo-2 addition (i.e., the number of "active" bits must be even). Equivalently, the modulo-2 product of the parity check matrix with the $2N$-bit codeword vector must equal a N-bit vector of zeros. As illustrated in Fig. 1, we can visualize these parity check constraints via a corresponding factor graph. The parity check matrix H can then be thought of as an adjacency matrix, where rows correspond to factor (parity) nodes P, columns to variable (message bit) nodes Y, and ones to edges linking factors to variables.

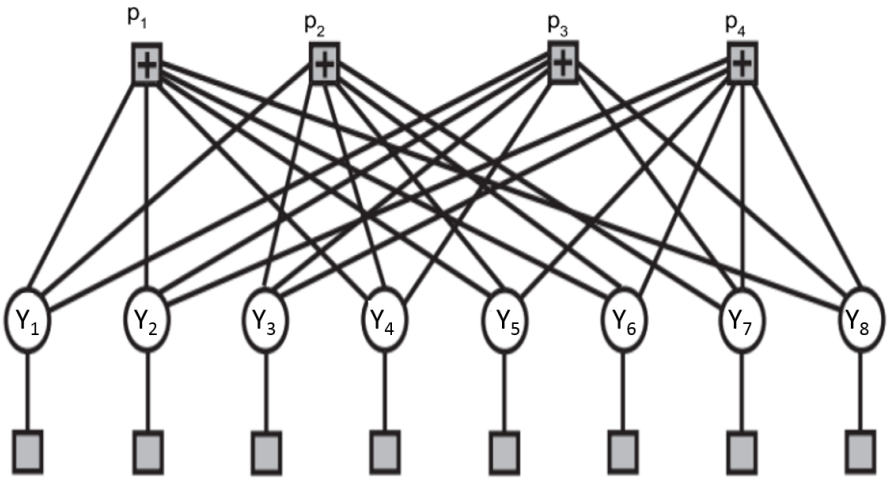


Figure 1: A factor graph representation of a LDPC code linking four factor (parity constraint) nodes to eight variable (message bit) nodes. The unary factors encode noisy observations of the message bits from the output of some communications channel.

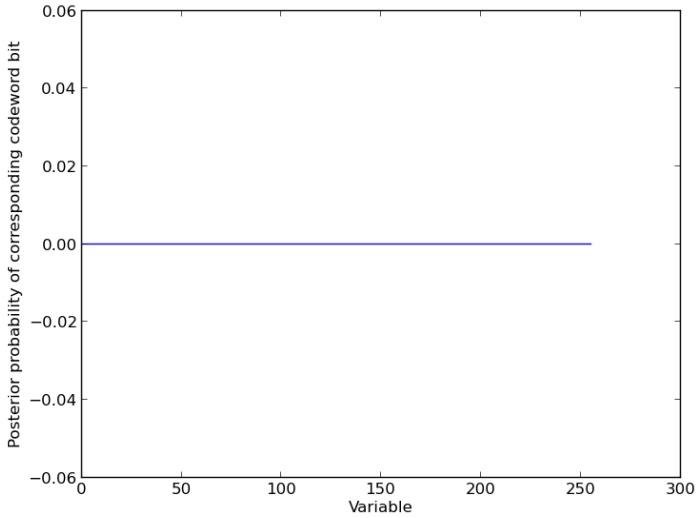
(a) **[9 points]** Implement code that, given an arbitrary parity check matrix H, constructs a corresponding factor graph. The parity check factors should evaluate to 1 if an even number of adjacent bits are active (equal 1), and 0 otherwise. For a given H matrix, define a small test case, and verify that your graphical model assigns zero probability to invalid codewords.

(b) Implement loopy belief propagation (sum product) for the factor graphs you generate in Part a.

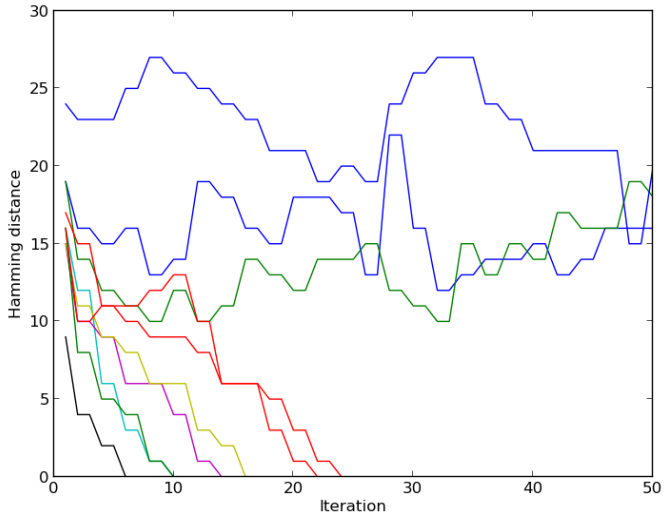
(c) **[12 points]** Load the $N = 128$-bit LDPC code provided in ***ldpc36-128.mat***. To evaluate decoding performance, we assume that the all-zeros codeword Y is sent, which always satisfies any set of parity checks. Using the **rand** method, simulate the output $\widetilde{Y}$ of a binary symmetric channel: each transmitted bit is flipped to its complement with error probability $\epsilon = 0.05$, and equal to the transmitted bit otherwise. Define unary factors for each variable node Y_i which equal $1 - \epsilon$ if that bit equals the "received" bit at the channel output, and ϵ otherwise. Run loopy belief propagation for 50 iterations of a parallel message update schedule (update all messages in each iteration using BP equations, based on the messages from the previous iteration), initializing by setting all variable-to-factor messages to be constant. After the final iteration, plot the estimated posterior probability (conditioned on the received, noisy message) that each codeword bit equals one. If we decode by setting each bit to the maximum of its corresponding marginal, would we find the right codeword?

We can see that the marginal probabilities of every bit being one converge to 0. So, we can find the right codeword by setting each bit to the maximum of its corresponding marginal.

(d) **[8 points]** Repeat the experiment from part (b) for 10 random channel noise realizations with error probability $\epsilon = 0.06$. For each trial, run sum-product for 50 iterations. After each iteration, estimate the codeword by taking the maximum of each bit's marginal distribution, and evaluate the Hamming



distance (number of differing bits) between the estimated and true (all-zeros) codeword. On a single plot, display 10 curves showing Hamming distance versus iteration for each trial. Is BP a reliable decoding algorithm?



Though BP makes the estimated codeword converge to its actual value for some trials, for others it fails to converge. So, BP is not a reliable decoding algorithm. This supports the fact that loopy BP is only an approximate inference algorithm.

(e) **[8 points]** Repeat part (c) with two higher error probabilities, $\epsilon = 0.08$ and $\epsilon = 0.10$. Discuss any qualitative differences in the behavior of the loopy BP decoder.

With higher ϵ values, fewer of the iterations converge to the true value of the codeword.

(f) **[12 points]** Load the $N = 1600$-bit LDPC code provided in ***ldpc36-1600.mat***. Using this, we will replicate the visual decoding demonstration from MacKay's Fig. 47.5. Start by converting a 40×40 binary image to a 1600-bit message vector; you may use the logo image we provide, or create your own. Encode the message using the provided generator matrix G, and add noise with error

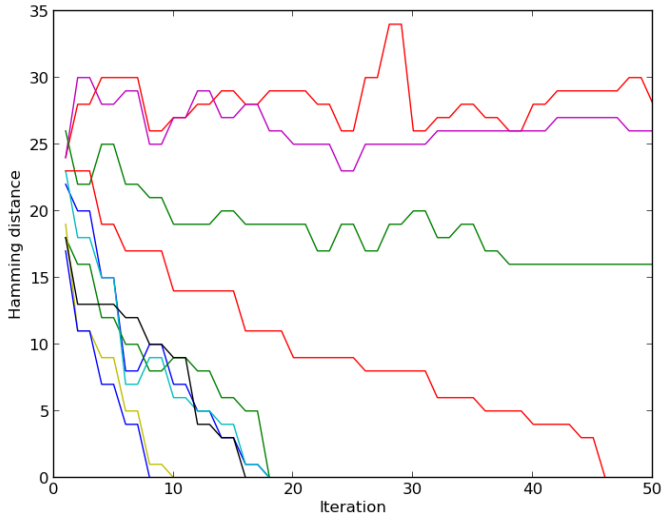


Figure 1: Convergence for $\epsilon = 0.08$

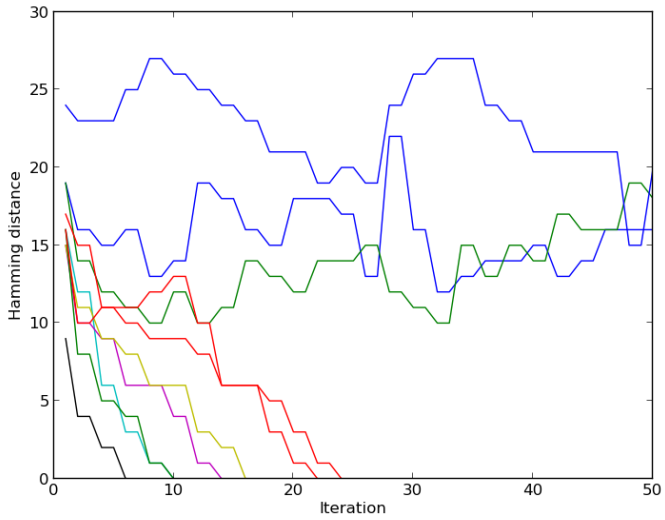


Figure 2: Convergence for $\epsilon = 0.1$

probability $\epsilon = 0.06$ (flip each bit with that probability). For this input, plot images showing the output of the sum-product decoder after 0, 1, 2, 3, 5, 10, 20, and 30 iterations. The **rem** method may be useful for computing modulo-2 sums. You can use the **reshape** method to easily convert between images and rasterized message vectors.

We can see the image getting progressively denoised with the number of iterations.

(g) **[6 points]** Repeat part (e) with a higher error probability of $\epsilon = 0.10$, and discuss differences.

We can see the image getting progressively denoised with the number of iterations. However, the noise is significant even after 20 iterations, due to the higher ϵ value.

Notes:

- This problem requires substantial computing time, so start early. And also you may choose not to

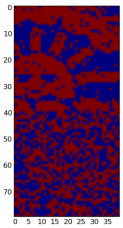


Figure 3: Original

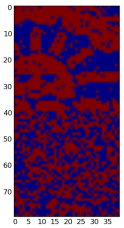


Figure 4: After 1 iteration

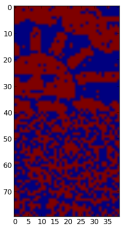


Figure 5: After 2 iterations

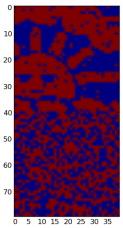


Figure 6: After 3 iterations

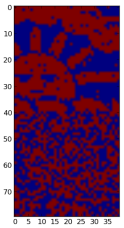


Figure 7: After 5 iteration

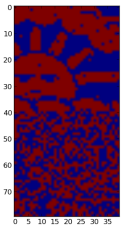


Figure 8: After 10 iterations

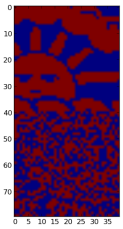


Figure 9: After 20 iterations

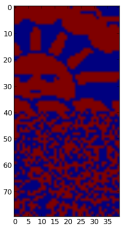


Figure 10: After 30 iteration

use our provided classes or starter code.

- Error correcting codes are *everywhere*! This file is very likely stored on your computer in some memory (disk, RAM, ..) that uses an error correcting code. LDPC codes (like the one you implemented) in particular are used among other things for deep space communications, for 10GBase-T Ethernet and are also part of the Wi-Fi 802.11 standard. Loopy belief propagation is essentially a state-of-the-art decoding algorithm.

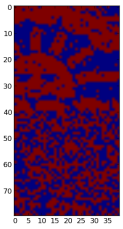


Figure 11: Original

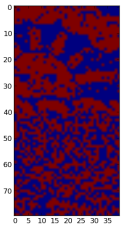


Figure 12: After 1 iteration

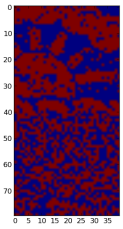


Figure 13: After 2 iterations

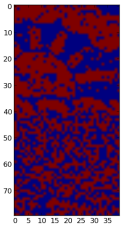


Figure 14: After 3 iterations

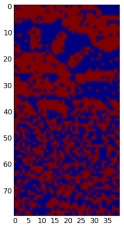


Figure 15: After 5 iteration

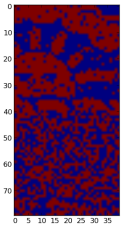


Figure 16: After 10 iterations

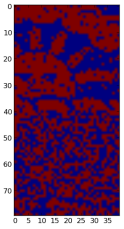


Figure 17: After 20 iterations

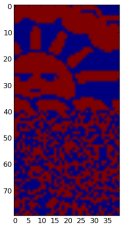


Figure 18: After 30 iteration