Problem Set 2

CS 6347

Due: 3/2/2019 by 11:59pm

Note: all answers should be accompanied by explanations for full credit. Late homeworks cannot be accepted. All submitted code **MUST** compile/run.

Problem 1: MAP IP vs. LP (20 pts)

Given a graph $G = (V, E)$, a valid vertex coloring of a graph with a budget of $k > 0$ colors is an assignment of a color to each vertex of the graph so that no two neighboring vertices receive the same color. We can represent each possible vertex coloring of the graph by creating a random variable x_i for each edge $i \in V$ that takes values in the set $\{1, \ldots, k\}$, indicating which of the k possible colors is selected for the vertex i. The uniform probability distribution over vertex colorings is then given by

$$p(x) = \frac{1}{Z} \prod_{(i,j) \in E} \mathbf{1}_{x_i \neq x_j}$$

where Z is the normalizing constant. In this problem, you will examine non-uniform coloring distributions. That is, each color will be associated with a weight $w_a \in \mathbb{R}$ for each $a \in \{1, \ldots, k\}$. For each vertex $i \in V$ we will add a new potential function $\phi_i(x_i) = \exp(w_{x_i})$.

1. Suppose that G is a cycle with an even number of vertices. For each $k > 0$, what is the MAP assignment in the case that $w_a = a$ for all $a \in \{1, \ldots, k\}$?

2. Write down the MAP IP and LP for a cycle on three variables with $k = 3$ and weights $w_1 = 1$, $w_2 = 2$, $w_3 = 3$. What is the optimal value of the MAP IP? What is the optimal value of the MAP LP?

Problem 2: Loopy Belief Propagation (80 pts)

In this problem, you will attempt to approximate the maximizing assignment and the partition function using *loopy* belief propagation for the non-uniform vertex coloring in Problem 1. Recall that, for tree structured graphs, we can use belief propagation (i.e., variable elimination starting from the leaves of the tree) to compute these quantities exactly. Unlike belief propagation on a tree, loopy belief propagation initializes the messages to some fixed value and then iteratively updates the messages until (hopefully) this process converges. There is one important caveat to make this process work. In general, you will need to rescale the messages (or you will end up with an overflow very quickly). That is, for some joint distribution $p(x) = \frac{1}{Z} \prod_{i \in V} \phi_i(x_i) \prod_{C \in \mathcal{C}} \psi_C(x_C)$, the message

updates should be of the following form. The messages at step $t > 1$ for the loopy sum-product algorithm should be computed from the messages at time $t-1$ via the following updates.

$$m^t_{C\to i}(x_i) = \eta^t_{C\to i} \sum_{x_{C\setminus i}} \left[\psi_C(x_C) \prod_{k\in C\setminus i} m^{t-1}_{k\to C}(x_k)\right]$$

$$m^t_{i\to C}(x_i) = \eta^t_{i\to C}\phi_i(x_i) \prod_{C'\in\mathcal{C}\setminus C st. i\in C'} m^{t-1}_{C'\to i}(x_i),$$

where $\eta^t_{C\to i}$ and $\eta^t_{i\to C}$ are scaling factors used to keep the messages from becoming too large. The approximate marginals at time t are computed, as before,

$$b^t_i(x_i) = \gamma^t_i \phi_i(x_i) \prod_{C\in\mathcal{C} st. i\in C} m^t_{C\to i}(x_i)$$

$$b^t_C(x_C) = \gamma^t_C \psi_C(x_C) \prod_{k\in C} m^t_{k\to C}(x_k)$$

where γ^t_i and γ^t_C are normalizing constants. These approximate marginals are called beliefs.

1. Implement the sum-product algorithm for the coloring problem in MATLAB. Your solution should take as input an $n \times n$ matrix A corresponding to the adjacency matrix of the graph G, a vector $w \in \mathbb{R}^k$ representing the weights, and `its` which is an input that controls the number of iterations of sum-product. The function should return the approximate partition function attained after `its` iterations by plugging the appropriately normalized beliefs into the Bethe free energy.

   ```
   function Z = sumprod(A, w, its)
   ```

2. Implement the max-product algorithm for the coloring problem in MATLAB. Your solution should take as input an $n \times n$ matrix A corresponding to the adjacency matrix of the graph G, a vector $w \in \mathbb{R}^k$ representing the weights, and `its` which is an input that controls the number of iterations of max-product. The function should return the approximate maximum weight coloring attained after `its` iterations. To do this, simply return the assignment that maximizes each singleton belief as a vector whose i^{th} entry corresponds to the maximizing assignment of the belief for the vertex i. If the solution is not unique, return 0 for that component. *Hint: most of the code is identical to the sum-product case.*

   ```
   function x = maxprod(A, w, its)
   ```

You should check that your code produces the correct answer when the graph is a tree!