Advanced Probabilistic Machine Learning and Applications

Caterina De Bacco and Martina Contisciani

1 Tutorial 10: Belief Propagation

Exercise 1: implementing BP for graph coloring

In tutorial 9 we saw the coloring problem of graph theory.
Given a (unweighted, undirected) graph $G(V,E)$ a coloring $M \subseteq E$ is an assignment of labels, called colors, to the vertices of a graph such that no two adjacent vertices share the same color.

We saw the probability distribution is:

$$P(\mathbf{s}) = \frac{1}{Z}\prod_{(ij)\in \tilde{F}} \psi_{ij}(s_i,s_j) = \frac{1}{Z}\prod_{(ij)\in E} \mathrm{e}^{-\beta\, \mathbb{I}(s_i=s_j)} \quad ,$$

where we used the soft constraint $\psi_{ij}(s_i,s_j) = \mathrm{e}^{-\beta\,\mathbb{I}(s_i=s_j)}$, and then we let $\beta \to \infty$.
This resulted in BP updates:

$$\chi_{s_j}^{j\to(ij)} = \frac{1}{\tilde{Z}^{j\to i}} \prod_{k\in\partial j\setminus i} \left[1-\left(1-\mathrm{e}^{-\beta}\right)\chi_{s_j}^{k\to(kj)}\right] \quad , \tag{1}$$

where we can interpret:

- $1-\left(1-\mathrm{e}^{-\beta}\right)\chi_{s_j}^{k\to(kj)}$ as the probability that neighbor k is fine with j taking color s_j.
- $\prod_{k\in\partial j\setminus i}\left[1-\left(1-\mathrm{e}^{-\beta}\right)\chi_{s_j}^{k\to(kj)}\right]$ as the probability that all the neighbors are fine that j taking color s_j (if i was excluded and (ij) erased).

Notice that we only distinguish whether a variable takes the same color of its neighbor or not, but we do not distinguish what color among the different ones.

The one-point and two-point marginals are:

$$\chi_{s_j} = \frac{1}{Z^{(i)}}\prod_{k\in\partial j}\left[1-\left(1-\mathrm{e}^{-\beta}\right)\chi_{s_j}^{k\to(kj)}\right] \tag{2}$$

$$\chi_{s_i,s_j} = \frac{1}{Z^{(ij)}}\psi_{ij}(s_i,s_j)\chi_{s_j}^{j\to(ij)}\,\chi_{s_i}^{i\to(ij)} \tag{3}$$

which are valid at convergence.

In this tutorial we will code the belief propagation for Erdős-Rényi graphs coloring for $\beta \to \infty$.

(a) Initialize BP close to be uniform fixed point, i.e. $1/q + \epsilon_s^{j\to i}$ and iterate the equations until convergence. Define converge as the time when the

$$err < \tilde{\epsilon}$$

where $err = \frac{1}{2q|E|}\sum_{(ij)\in E}\sum_s |\left(\chi_s^{i\to(ij)}(t+1) - \chi_s^{i\to(ij)}(t)\right)|$, with suitably chosen small $\tilde{\epsilon}$.

(b) Check how the behavior depends on the order of update, i.e. compare what happens if you update all messages at once or randomly one by one.

(c) For parameters where the update converges, plot the convergence time as a function of the average degree c. Do this on as large graphs as is feasible with your code.

(d) Assign one color to each node at convergence, based on the argmax of the one-point marginals. Compute the fraction of violations you get over $N_{real} = 5$ random initializations of the graph and plot it as a function of c. Repeat and plot the fraction of violations as a function of the number of colors q.

(e) Check how the behavior depends on the initialization. What if initial messages are random? What if they are all points towards the first color?

[Solution]

(a) In the code, we generate an Erdős-Rényi graph with $N = 100$ nodes and average degree $c = 5$. We fix the number of colors $q = 3$ and $\beta = 2$. We use the directed version of the generated graph, so that each message is saved as an attribute of a single edge. The BP messages are initialized as $1/q + \epsilon_s^{j\to i}$ where $\epsilon_s^{j\to i} \sim \text{Uniform}[-\alpha/q, \alpha/q]$, with $\alpha = 0.1$. Since the entries of the vector with the messages represent probabilities, they have to be normalized so as to sum to 1. $\tilde{\epsilon}$ is chosen to be 10^{-4}, and in the Jupyter this is codified by the variable abs_tol. As a remind, $|E|$ is the number of edges of the original graph, so we have to divide by 2 the number of edges of the directed graph.

(b) If we update BP messages parallelly, BP will not converge in $max\_it = 1000$ iterations, but if we update BP messages randomly one by one, BP converges in around 30 iterations. Figure 1 shows the err as function of the number of iterations for the parallel and the random update. The parallel version updates all messages at once and doesn't use the improvements for computing the messages at time $(t+1)$. The plot on the left reflects this phenomena showing how the err doesn't decrease as the number of iterations increases. On the other hand, the randomized version includes the update version of some messages for computing the update of the remained ones, and this accelerate the convergence. In the code, we consider a permutation of the edges in order to start the randomization from different messages.

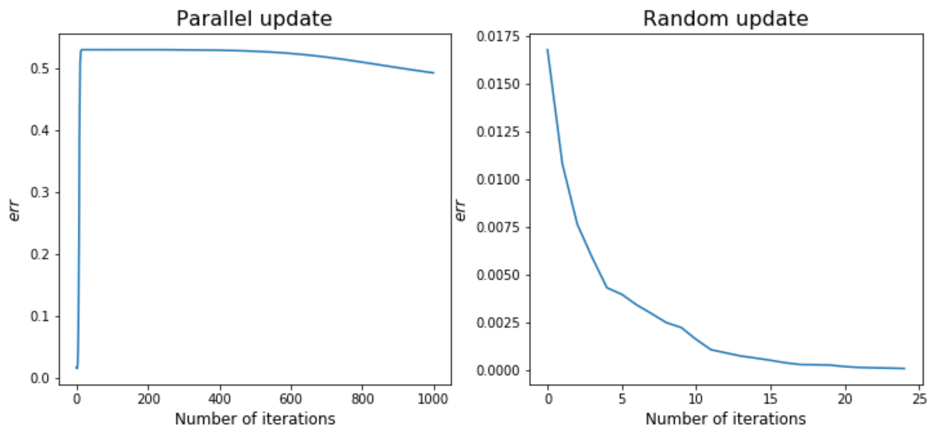

Figure 1: err value under different BP update for a graph with $N = 100$ and $c = 5$; $q = 3$ and $\beta = 2$.

(c) We use $q = 3$, $\beta = 2$ and $N \in [50, 100, 200]$. We choose 50 values of $c \in [0.1, 7]$ and we draw 5 random graphs for each c. The update of BP is randomly one by one and for each c we save the median of the converge number of iterations. Figure 2 shows the median converge time of these five trials as a function of the average degree for all choices of N. As the average degree increases, the problem becomes more difficult to solve and the BP doesn't converge. This is reflected by the jump around $c = 6$. Remember that we are using 3 colors, and when the average degree is higher then q, the constraint of having neighbours with a different colors is much more difficult to satisfy. Moreover, as N increases, the jump appears earlier because the problem is more difficult.

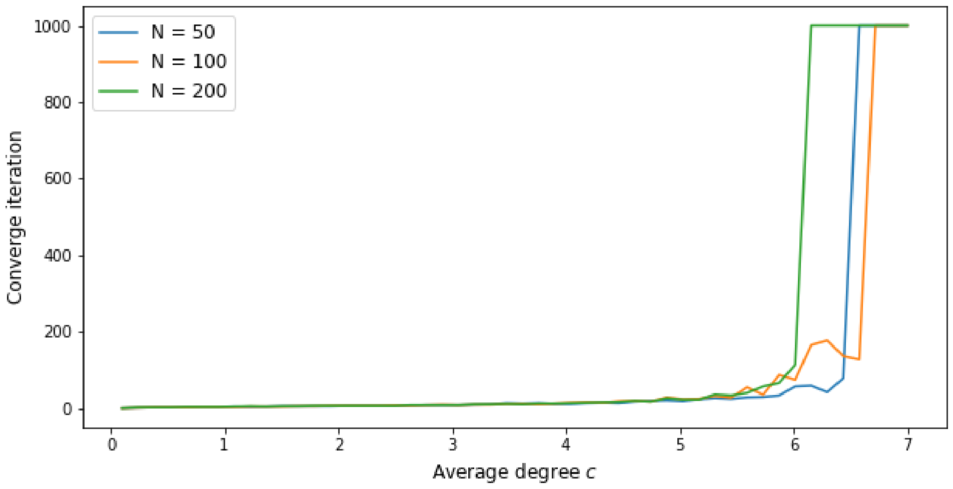

Figure 2: Convergence time vs. c for $N = 50, 100, 200$, $q = 3$ and $\beta = 2$.

(d.1) We use $q = 3$, $\beta = 2$, $N = 500$ and we choose 30 values of $c \in [2, 7]$. For each c we draw 5 random graphs and we update BP randomly one by one. A violation represents the case when two neighbours have the same color, and the total number of errors is normalized over the number of edges of the undirected graph for allowing the comparison between graphs with different number of edges. We save the mean and the standard deviation of the fraction of violations and Figure 3 shows the result. As expected, the fraction of violations increases as the average degree increases, because the number of neighbours becomes bigger than the number of colors.

(d.2) We use $c = 5$, $\beta = 2$, $N = 500$ and we choose values of $q \in [2, 12]$. For each q we draw 5 random graphs and we update BP randomly one by one. Figure 4 shows the mean of the fraction of violations vs the number of colors, and the vertical bars represent the standard deviation over the trials. The fraction of violations decreases as the number of colors increases, because the problem becomes easier as the choices of colors increases and the average number of neighbours remains constant.

(e) We fix $N = 1000$, $c = 5$, $q = 3$ and $\beta = 2$. We use the generated graph with three different initializations: small perturbation around the fixed-solution $1/q$, uniformly random and point mass on first color. Figure 5 shows the err as a function of the number of iterations for all cases. If we start around the fix point $1/q$, BP converges faster and the err is quite small. The random initialization is the slowest to converge and the first-color curve has the biggest jump.

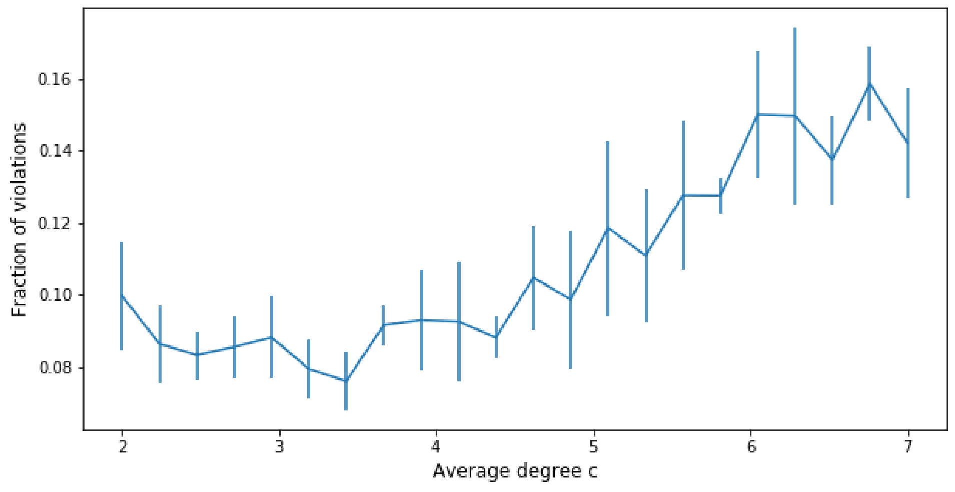

Figure 3: Fraction of violations vs. c for $N = 500$, $q = 3$ and $\beta = 2$. We plot the mean over 5 independent realizations and the bars represent the standard deviation.

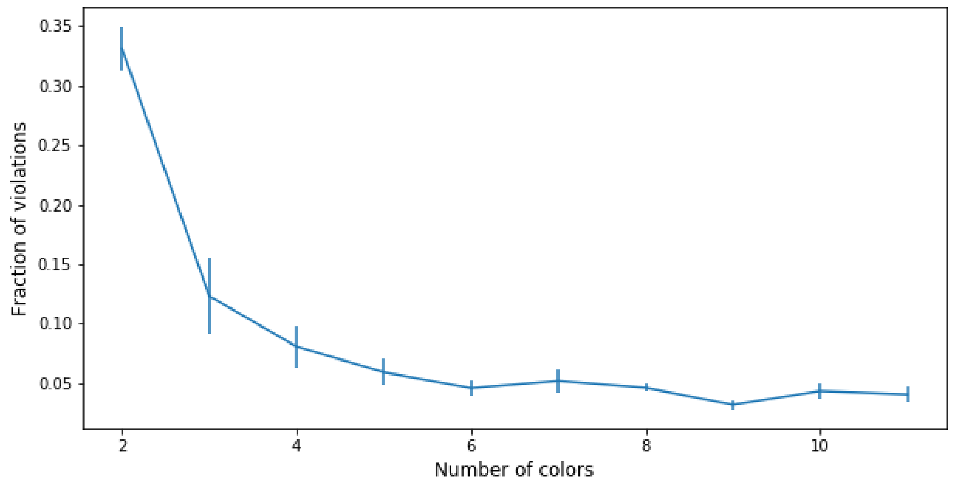

Figure 4: Fraction of violations vs. q for $N = 500$, $c = 5$ and $\beta = 2$. We plot the mean over 5 independent realizations and the bars represent the standard deviation.

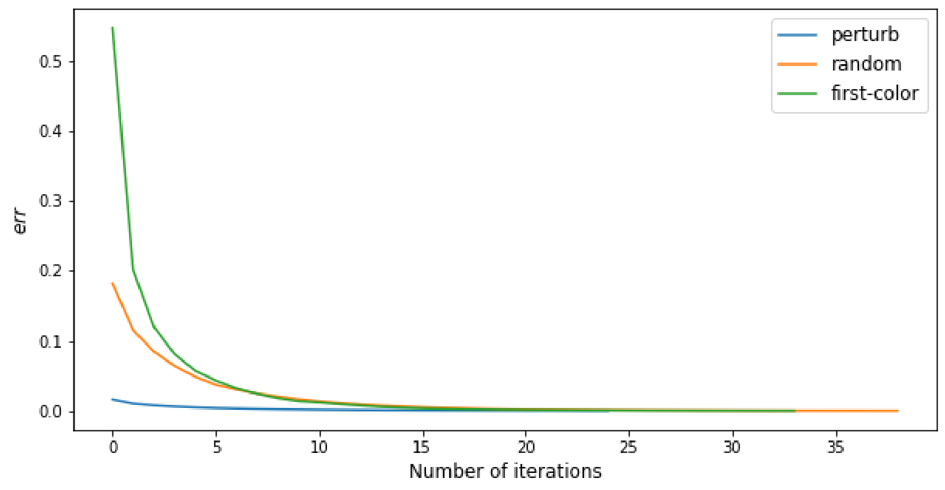

Figure 5: err value the BP with three different initializations.