Week 4: Inference on graphs continued

Sum-product on trees

Consider the basic example considered in the lecture.

To have concrete numbers, suppose all variables are binary $\{0,1\}$ and take $\psi_i(x_i) \equiv 1$ with

$$\psi_{12} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \quad \psi_{13} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \psi_{34} = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}, \quad \psi_{35} = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}.$$

We have

$$p(x_1,x_2,x_3,x_4,x_5) = \frac{1}{Z}\prod_{i=1}^{5}\psi_i(x_i)\psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{34}(x_3,x_4)\psi_{35}(x_3,x_5).$$

Let's fix the values of three variables: $\bar{x}_2 = 1$, $\bar{x}_4 = 1$, $\bar{x}_5 = 0$ and we get

$$p(x_1,1,x_3,1,0) \;=\; \frac{1}{Z}\psi_{12}(x_1,1)\psi_{13}(x_1,x_3)\psi_{34}(x_3,1)\psi_{35}(x_3,0).$$

This gives

$$\begin{aligned} p(0,1,0,1,0) &= \frac{1}{Z}2\cdot 2\cdot 1\cdot 1 = \frac{4}{Z} \\ p(0,1,1,1,0) &= \frac{1}{Z}2\cdot 1\cdot 2\cdot 1 = \frac{4}{Z} \\ p(1,1,0,1,0) &= \frac{1}{Z}1\cdot 1\cdot 1\cdot 1 = \frac{1}{Z} \\ p(1,1,1,1,0) &= \frac{1}{Z}1\cdot 2\cdot 2\cdot 1 = \frac{4}{Z} \end{aligned}$$

From this, we easily get the conditional distribution

$$\begin{aligned} p(x_1,x_3|x_2=1,x_4=1,x_5=0) &= \frac{p(x_1,1,x_3,1,0)}{\sum_{x_1',x_3'=0}^{1} p(x_1',1,x_3',1,0)} \\ &= \frac{\frac{1}{Z}\psi_{12}(x_1,1)\psi_{13}(x_1,x_3)\psi_{34}(x_3,1)\psi_{35}(x_3,0)}{\frac{1}{Z}(4+4+1+4)} \\ &= \frac{\psi_{12}(x_1,1)\psi_{13}(x_1,x_3)\psi_{34}(x_3,1)\psi_{35}(x_3,0)}{13} \\ &= \frac{1}{13}\begin{bmatrix} 4 & 4 \\ 1 & 4 \end{bmatrix}. \end{aligned}$$

Suppose that we are interedted in the marginal distributions of x_1 and of x_3. We compute the message passing formulas as in the lecture. Because x_2, x_4, x_5 are fixed, we get

$$m_{2\to1}(x_1) = \psi_2(1)\psi_{12}(x_1, 1) = \begin{bmatrix}2\\1\end{bmatrix}$$
$$m_{4\to3}(x_3) = \psi_4(1)\psi_{34}(x_3, 1) = \begin{bmatrix}1\\2\end{bmatrix}$$
$$m_{5\to3}(x_3) = \psi_5(0)\psi_{35}(x_3, 0) = \begin{bmatrix}1\\1\end{bmatrix}$$

Since x_3 is not observed we have

$$m_{3\to1}(x_1) = \sum_{x_3} \psi_3(x_3)\psi_{13}(x_1, x_3)m_{4\to3}(x_3)m_{5\to3}(x_3) = \begin{bmatrix}4\\5\end{bmatrix}.$$

From this we get

$$b(x_1) = p(x_1|\bar{x}_2 = 1, \bar{x}_4 = 1, \bar{x}_5 = 0) \propto \psi_1(x_1)m_{2\to1}(x_1)m_{3\to1}(x_1) = \begin{bmatrix}8\\5\end{bmatrix}$$

and so $p(x_1 = 1|\bar{x}_2 = 1, \bar{x}_4 = 1, \bar{x}_5 = 0) = \frac{5}{13}$.
To compute $b(x_3) = p(x_3|\bar{x}_2 = 1, \bar{x}_4 = 1, \bar{x}_5 = 0)$ we need to compute the message $m_{1\to3}$

$$m_{1\to3}(x_3) = \sum_{x_1} \psi_1(x_1)\psi_{13}(x_1, x_3)m_{2\to1}(x_1) = \begin{bmatrix}5\\4\end{bmatrix}$$

This gives

$$b(x_3) \propto \psi_3(x_1)m_{1\to3}(x_3)m_{4\to3}(x_3)m_{5\to3}(x_3) = \begin{bmatrix}5\\8\end{bmatrix}$$

giving that $p(x_3 = 1|\bar{x}_2 = 1, \bar{x}_4 = 1, \bar{x}_5 = 0) = \frac{8}{13}$.

By the way, this simple Python code computes the joint distribution for any choice of potential functions. With this you can directly check all our calculations. It first computes the unnormalized quantities. For example, with the data above, we get

```
import numpy as np

def joint_distribution(psi_nodes, psi_edges):
    # Initialize joint distribution
    joint_dist = np.zeros((2, 2, 2, 2, 2))

    # Iterate over all possible combinations of binary variables
    for x1 in [0, 1]:
        for x2 in [0, 1]:
            for x3 in [0, 1]:
                for x4 in [0, 1]:
                    for x5 in [0, 1]:
                        # Calculate joint distribution
                        joint_dist[x1, x2, x3, x4, x5] = (
                            psi_nodes[0][x1] * psi_nodes[1][x2] *
psi_nodes[2][x3] * psi_nodes[3][x4] * psi_nodes[4][x5] *
                            psi_edges[0][x1, x2] * psi_edges[1][x1, x3] *
psi_edges[2][x3, x4] * psi_edges[3][x3, x5]
                        )



    return joint_dist

# Define node potentials
psi_nodes = [
    {0: 0.6, 1: 0.4},  # psi_1(x1)
    {0: 0.7, 1: 0.3},  # psi_2(x2)
    {0: 0.5, 1: 0.5},  # psi_3(x3)
    {0: 0.8, 1: 0.2},  # psi_4(x4)
    {0: 0.9, 1: 0.1}   # psi_5(x5)
]

# Define edge potentials
psi_edges = [
    {(0, 0): 0.5, (0, 1): 0.5, (1, 0): 0.5, (1, 1): 0.5},  # psi_12(x1, x2)
    {(0, 0): 0.4, (0, 1): 0.6, (1, 0): 0.7, (1, 1): 0.3},  # psi_13(x1, x3)
    {(0, 0): 0.8, (0, 1): 0.2, (1, 0): 0.6, (1, 1): 0.4},  # psi_34(x3, x4)
    {(0, 0): 0.9, (0, 1): 0.1, (1, 0): 0.2, (1, 1): 0.8}   # psi_35(x3, x5)
]
```

To compute the normalizing constant and the corresponding distribution we run the following code.

```
# compute the unnormalized distribution
joint_dist_notnormalized = joint_distribution(psi_nodes, psi_edges)

#print the normalizing constant
Z=np.sum(joint_dist_notnormalized)
print(Z)

# Print the joint distribution
print(np.divide(joint_dist_notnormalized,Z))
```

From this we get that for this choice of potential functions $Z = 162$. We can also confirm directly all the previous calculations. For example, to check that $p(x_3 = 1|\bar{x}_2 = 1, \bar{x}_4 = 1, \bar{x}_5 = 0) = \frac{8}{13}$, we run the following code

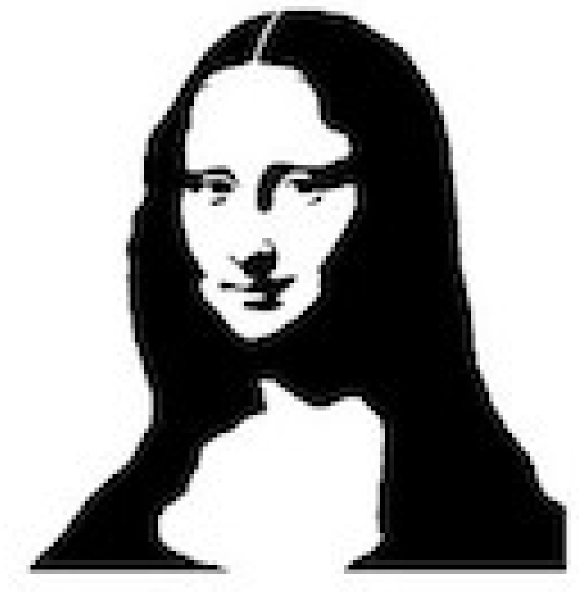
```
pz1 = np.sum(joint_dist_notnormalized[:, 1, 1, 1, 0])
pz0 = np.sum(joint_dist_notnormalized[:, 1, 0, 1, 0])
print(np.divide(pz1,pz0+pz1))
print(8/13)
```

Image Denoising with BP

Ising model for images

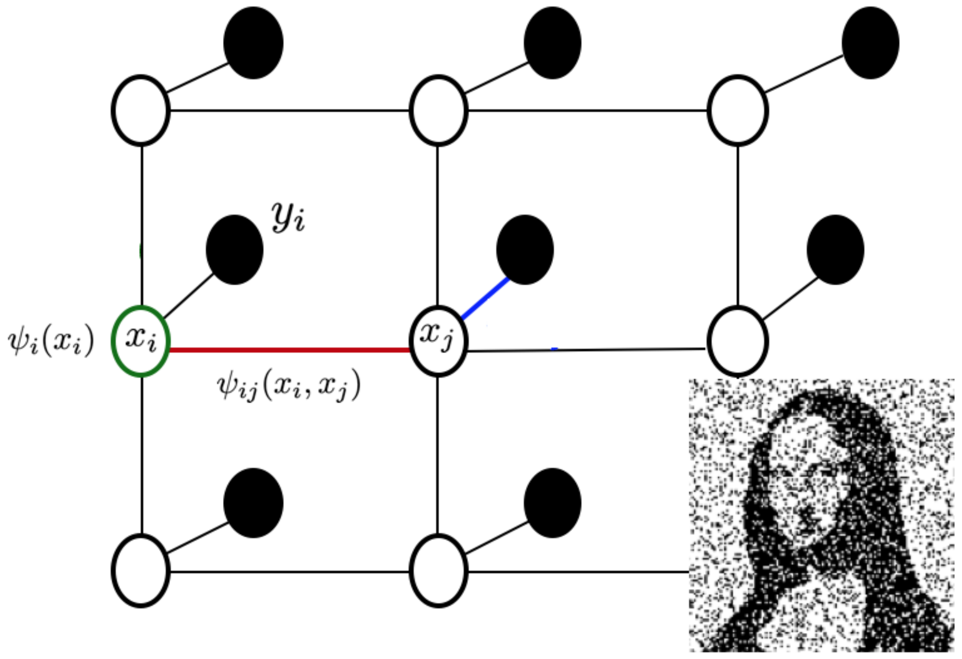
A binary image is a $n \times n$ matrix where each entry is either +1 or -1, i.e. $x_s \in \{-1, +1\}$. We vectorize this matrix row by row and denote the image as $x \in \mathbb{R}^{n^2}$.

For example, the Mona Lisa below is 128 x 128 image, vectorized to be $x \in \mathbb{R}^{16384}$.



Assume that one of your friends sent this image to you through a noisy channel. During transmission, each of its pixels may be flipped with a small probability, say ϵ. This means that with probability ϵ, there is error in the transmitted pixel.

The resulting noisy image can be denoised by using the Ising model. The following is based on $\epsilon = 0.2$.



Each unobserved node in this Ising model (MRF) corresponds to the original uncorrupted image. However, what you observed is the corrupted version $y \in \mathbb{R}^{n^2}$ with

$$\begin{aligned}\mathbb{P}(y_s|x_s) &= (1-\epsilon)^{\frac{1+y_s x_s}{2}} \epsilon^{\frac{1-y_s x_s}{2}} \quad \text{for all } s. \\ &= \exp\left\{\frac{1+y_s x_s}{2}\log(1-\epsilon) + \frac{1-y_s x_s}{2}\log(\epsilon)\right\} \\ &\propto \exp\{y_s x_s \frac{1}{2}\log(\frac{1-\epsilon}{\epsilon})\} \\ &= \exp\{y_s x_s \theta\} \quad \text{where } \theta = \frac{1}{2}\log(\frac{1-\epsilon}{\epsilon}).\end{aligned}$$

Loopy BP for image denoising

To estimate the true image, aka image denoising, we want to approximate the posterior distribution $p(x|y)$, which is the distribution of the true image given that we observed the noisy version. Our prior on the true image is based on the Ising model

$$p(x) \propto \prod_{s \sim t} \psi_{st}(x_s, x_t)$$

where the pairwise clique potentials are given as

$$\psi_{st}(x_s, x_s) = \begin{pmatrix} e^J & e^{-J} \\ e^{-J} & e^J \end{pmatrix}.$$

Here, J is the coupling strength between nodes s and t and the notation $s \sim t$ means there is an edge between s and t, i.e., $(s,t) \in \mathcal{E}$. We assume for simplicity that all these coupling strengths are equal.

The posterior can be easily written as

$$\begin{aligned}p(x|y) &\propto p(y,x)\\ &= p(x)\prod_s p(y_s|x_s)\\ &= \exp\{J\sum_{s\sim t} x_s x_t + \lambda \sum_s y_s x_s\}\\ &= \prod_{s\sim t}\psi_{st}(x_s,x_t)\prod_s \psi_s(x_s)\end{aligned}$$

Now it is clear that the node potentials are given as (note that y is a fixed vector here)

$$\psi_s(x_s) = \exp\{\lambda y_s x_s\}.$$

Now that we know both the clique as well as the node potentials, the problem reduces to running loopy BP on this problem. We recall the loopy BP updates

Loopy BP:

1. Initialize all messages uniformly $m_{i\to j}(x_j) = 1/k$ where k is the number of states x_j can take (in our case $k = 2$).
2. Keep doing BP updates until it (nearly) converges:

   $$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i,x_j) \prod_{k\in N(j)\neq i} m_{k\to j}(x_j)$$

   and normalize messages for stability $m_{j\to i}(x_i) = m_{j\to i}(x_i)/\sum_{x_i} m_{j\to i}(x_i)$.
3. It will often not converge, but that's generally ok.
4. Compute beliefs after message passing is done.

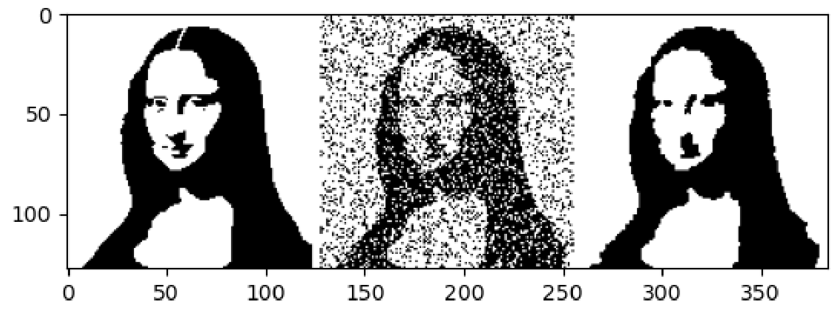
   $$b(x_i) \propto \psi_i(x_i)\prod_{j\in\mathcal{N}(i)} m_{j\to i}(x_i).$$

Here, each message $m_{j\to i}(x_i)$ is stored as a 2-dimensional vector, where its first coordinage $m_{j\to i}(+1)$ and its second coordinate $m_{j\to i}(-1)$. Similarly, beliefs $b(x_i)$ are also 2-dimensional vectors with first and second coordiants given by $b(+1)$ and $b(-1)$ respectively.

Even though loopy BP may not converge, even 10-20 iterations suffices to perform approximate inference on the posterior. The beliefs that we compute correspond to the marginals $p(x_i|y)$, thus using a naive decision rule

$$\hat{x}_i = \arg\max_{x_i} b(x_i)$$

we can estimate the true image. The result is remarklable:



Momentum

It is often the case in any optimization algorithm that we don't want to go all-in on the current iteration, and would like to be a bit conservative. This is achieved through the following method that adds *inertia* to the BP updates.

The new messages are computed as before

$$m^{+}_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)\neq i} m_{k\to j}(x_j)$$

but we update the old message with some momentum parameter $\gamma \in [0,1]$

$$m_{j\to i}(x_i) = \gamma\, m^{+}_{j\to i}(x_i) + (1-\gamma)\, m_{j\to i}(x_i)$$

and normalize messages for stability $m_{j\to i}(x_i) = m_{j\to i}(x_i)/\sum_{x_i} m_{j\to i}(x_i)$.

This will make the algorithm more stable.