Probabilistic Graphical Models in Computer Vision (IN2329)

Csaba Domokos

Summer Semester 2017
8. Belief Propagation
Overview of the course

- Sum–product algorithm
- Max–sum algorithm
- Loopy belief propagation

Graphical models:
- Bayesian network
- MRF
- CRF

Inference:
- Probabilistic inference
- MAP inference

Learning:
- Probabilistic parameter learning
- Loss–minimizing parameter learning

Computer Vision:
- Image segmentation
- Object detection
- Stereo matching
- Human–pose estimation

IN2329 - Probabilistic Graphical Models in Computer Vision
Inference means the procedure to estimate the probability distribution, encoded by a graphical model, for a given data (or observation).

Assume we are given a factor graph $G = (\mathcal{V}, \mathcal{E}', \mathcal{F})$ and the observation $x$.

- **Maximum A Posteriori (MAP) inference**: find the state $y^* \in \mathcal{Y}$ of maximum probability,

$$
    y^* \in \argmax_{y \in \mathcal{Y}} p(y \mid x) = \arg\min_{y \in \mathcal{Y}} E(y; x).
$$

- **Probabilistic inference**: find the value of the partition function $Z(x)$ and the marginal distributions $\mu_F(y_F)$ for each factor $F \in \mathcal{F}$,

$$
    Z(x) = \sum_{y \in \mathcal{Y}} \exp(-E(y; x)),
$$

$$
    \mu_F(y_F) = p(y_F \mid x).
$$
Today we are going to learn about belief propagation to perform exact inference on graphical models having tree structure.

Reminder: a tree is a connected and acyclic graph.

- Probabilistic inference: Sum-product algorithm
- MAP inference: Max-sum algorithm

We also extend belief propagation for general factor graphs, which results in an approximate inference.
Sum–product algorithm
Assume that we are given the following factor graph and a corresponding energy function $E(y)$, where $Y = Y_i \times Y_j \times Y_k \times Y_l$.

We want to compute $p(y)$ for any $y \in Y$ by making use of the factorization

$$p(y) = \frac{1}{Z} \exp(-E(y)) = \frac{1}{Z} \exp(-E_A(y_i, y_j)) \exp(-E_B(y_j, y_k)) \exp(-E_C(y_k, y_l)).$$

**Problem:** we also need to calculate the partition function

$$Z = \sum_{y \in Y} \exp(-E(y)) = \sum_{y_i \in Y_i} \sum_{y_j \in Y_j} \sum_{y_k \in Y_k} \sum_{y_l \in Y_l} \exp(-E(y_i, y_j, y_k, y_l)),$$

which looks expensive (the sum has $|Y_i| \cdot |Y_j| \cdot |Y_k| \cdot |Y_l|$ terms).
We can expand the partition function as

\[
Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E(y_i, y_j, y_k, y_l))
\]

\[
= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp\left( - \left( E_A(y_i, y_j) + E_B(y_j, y_k) + E_C(y_k, y_l) \right) \right)
\]

\[
= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E_A(y_i, y_j)) \exp(-E_B(y_j, y_k)) \exp(-E_C(y_k, y_l))
\]

\[
= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) \sum_{y_l \in \mathcal{Y}_l} \exp(-E_C(y_k, y_l)).
\]
Note that we can successively eliminate variables, that is

\[ Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) \sum_{y_i \in \mathcal{Y}_i} \exp(-E_C(y_k, y_l)) \]

\[ = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) r_{C \rightarrow Y_k}(y_k) \]

\[ = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) r_{B \rightarrow Y_j}(y_j) \]

\[ = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) r_{A \rightarrow Y_i}(y_i) = \sum_{y_i \in \mathcal{Y}_i} r_{A \rightarrow Y_i}(y_i). \]
Now we are assuming a tree-structured factor graph and are applying the same elimination procedure as before.

\[
Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) \sum_{y_l \in \mathcal{Y}_l} \exp(-E_C(y_j, y_l))
\]
\[
= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \underbrace{r_{B \rightarrow Y_j}(y_j)}_{qY_j \rightarrow A(y_j)} \underbrace{r_{C \rightarrow Y_j}(y_j)}_{qY_j \rightarrow A(y_j)}
\]
\[
= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) qY_j \rightarrow A(y_j)
\]
Now we are assuming a tree-structured factor graph and applying the same elimination procedure as before.

\[
Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) q_{Y_j \rightarrow A}(y_j) r_{A \rightarrow Y_i}(y_i)
\]

\[
= \sum_{y_i \in \mathcal{Y}_i} r_{A \rightarrow Y_i}(y_i).
\]
**Message**: pair of vectors at each factor graph edge \((i, F) \in \mathcal{E}'\).

1. **Variable-to-factor** message \(q_{Y_i \rightarrow F} \in \mathbb{R}^{\mathcal{Y}_i}\) is given by

\[
q_{Y_i \rightarrow F}(y_i) = \prod_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i),
\]

where \(M(i) = \{F \in \mathcal{F} : (i, F) \in \mathcal{E}'\}\) denotes the set of factors adjacent to \(Y_i\).

2. **Factor-to-variable** message: \(r_{F \rightarrow Y_i} \in \mathbb{R}^{\mathcal{Y}_i}\).
2. **Factor-to-variable** message $r_{F \rightarrow Y_i} \in \mathbb{R}^{Y_i}$ is given by

\[
r_{F \rightarrow Y_i}(y_i) = \sum_{y'_F \in Y_F, \ y'_i = y_i} \left( \exp(-E_F(y'_F)) \prod_{l \in N(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y'_l) \right),
\]

where $N(F) = \{ i \in V : (i, F) \in \mathcal{E}' \}$ denotes the set of variables adjacent to $F$. 

![Diagram](image)
One can note that the message updates depend on each other.

\[
    r_{F \rightarrow Y_i}(y_i) = \sum_{y'_F \in \mathcal{Y}_F, y'_i = y_i} \left( \exp(-E_F(y'_F)) \prod_{l \in N(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y'_l) \right) \tag{1}
\]

\[
    q_{Y_i \rightarrow F}(y_i) = \prod_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i) \tag{2}
\]

The messages that do not depend on previous computation are the following.

- The factor-to-variable messages in which no other variable is adjacent to the factor, that is the product in (1) will be empty.
- The variable-to-factor messages in which no other factor is adjacent to the variable, that is the product in (2) is empty and the message will be one.
For tree-structured factor graphs there always exist at least one such message that can be computed initially, hence all the dependencies can be resolved.

1. Select one variable node as root of the tree (e.g., $Y_m$)
2. Compute leaf-to-root messages (e.g., by applying depth-first-search)
3. Compute root-to-leaf messages (reverse order as before)

Partition function is evaluated at the (root) node \( i \)

\[
Z = \sum_{y_i \in \mathcal{Y}_i} \prod_{F \in M(i)} r_{F \rightarrow Y_i}(y_i)
\]
The *marginal distribution* for each factor can be computed as

$$\mu_F(y_F) \triangleq \sum_{y' \in \mathcal{Y}, \ y'_F = y_F} p(y) = \sum_{y' \in \mathcal{Y}, \ y'_F = y_F} \frac{1}{Z} \exp(-\sum_{H \in \mathcal{F}} E_H(y'_H)) $$

$$= \frac{1}{Z} \exp(-E_F(y_F)) \sum_{y' \in \mathcal{Y}_H, \ H \in \mathcal{F}\setminus\{F\}} \exp(-\sum_{H \in \mathcal{F}\setminus\{F\}} E_H(y'_H)) $$

$$= \frac{1}{Z} \exp(-E_F(y_F)) \prod_{i \in N(F)} \prod_{i \in N(F)} q_{Y_i \rightarrow F}(y_i) .$$
Assume a *tree-structured* factor graph. If the messages are computed based on *depth-first search order* for the *sum–product algorithm*, then it converges after $2|\mathcal{V}|$ iterations and provides the **exact** marginals.

If $|\mathcal{Y}_i| \leq K$ for all $i \in \mathcal{V}$, then the complexity of the algorithm $O(|\mathcal{V}| \cdot K^L)$, where $L = \max_{F \in \mathcal{F}} |N(F')|$.

\[
r_{F \rightarrow Y_i}(y_i) = \sum_{y'_F \in \mathcal{Y}_F, y'_i = y_i} \left( \exp(-E_F(y'_F)) \prod_{j \in N(F') \setminus \{i\}} q_{Y_j \rightarrow F}(y'_j) \right).
\]

Note that the complexity of the naïve way is $O(K|\mathcal{V}|)$.

*Reminder:* Assuming $f, g : \mathbb{R} \rightarrow \mathbb{R}$, the notation $f(x) = O(g(x))$ means that there exists $C > 0$ and $x_0 \in \mathbb{R}$ such that $|f(x)| \leq C|g(x)|$ for all $x > x_0$. 

*Sum–product algorithm*  *Max–sum algorithm*  *Loopy belief propagation*
Max–sum algorithm
MAP inference

\[ y^* \in \arg\max_{y \in \mathcal{Y}} p(y) = \arg\max_{y \in \mathcal{Y}} \frac{1}{Z} \tilde{p}(y) = \arg\max_{y \in \mathcal{Y}} \tilde{p}(y). \]

Similar to the sum–product algorithm one can obtain the so-called max–sum algorithm to solve the above maximization.

By applying the \( \ln \) function, we have

\[
\ln \max_{y \in \mathcal{Y}} \tilde{p}(y) = \max_{y \in \mathcal{Y}} \ln \tilde{p}(y)
\]

\[
= \max_{y \in \mathcal{Y}} \ln \prod_{F \in \mathcal{F}} \exp(-E_F(y_F))
\]

\[
= \max_{y \in \mathcal{Y}} \sum_{F \in \mathcal{F}} -E_F(y_F).
\]
Now we are assuming a tree-structured factor graph and applying an elimination procedure as before.

\[
\max_{y\in\mathcal{Y}} \sum_{F\in\mathcal{F}} -E_F(y_F) = \max_{y\in\mathcal{Y}} -E_A(y_i, y_j) - E_B(y_j, y_k) - E_C(y_j, y_l)
\]

\[
= \max_{y_i, y_j} -E_A(y_i, y_j) + \max_{y_k} -E_B(y_j, y_k) + \max_{y_l} -E_C(y_j, y_l)
\]

\[
= \max_{y_i, y_j} -E_A(y_i, y_j) + \underbrace{r_{B\rightarrow Y_j}(y_j)}_{q_{Y_j\rightarrow A}(y_j)} + \underbrace{r_{C\rightarrow Y_j}(y_j)}_{q_{Y_j\rightarrow A}(y_j)}
\]

\[
= \max_{y_i, y_j} -E_A(y_i, y_j) + r_{B\rightarrow Y_j}(y_j) + r_{C\rightarrow Y_j}(y_j)
\]
Now we are assuming a tree-structured factor graph and applying an elimination procedure as before.

The solution is then obtained as:

\[
\begin{align*}
    y_i^* &\in \arg\max_{y_i} r_{A\rightarrow Y_i}(y_i), \\
    y_j^* &\in \arg\max_{y_j} -E_A(y_i^*, y_j) + q_{Y_j\rightarrow A}(y_j), \\
    y_k^* &\in \arg\max_{y_k} -E_B(y_j^*, y_k), \\
    y_l^* &\in \arg\max_{y_l} -E_C(y_j^*, y_l).
\end{align*}
\]
The messages become as follows

\[ q_{Y_i \rightarrow F}(y_i) = \sum_{F' \in \mathcal{M}(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i) \]

\[ r_{F \rightarrow Y_i}(y_i) = \max_{y'_F \in \mathcal{Y}_F, \ y'_i = y_i} \left( -E_F(y'_F) + \sum_{l \in \mathcal{N}(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y'_l) \right). \]

The \textit{max-sum algorithm} provides \textbf{exact} MAP inference for tree-structured factor graphs.
Choosing an optimal state

After calculating the messages, the following **back-tracking** algorithm is applied for choosing an optimal \( y^* \).

1. Initialize the procedure at the root node \((Y_i)\) by choosing any

\[
y_i^* \in \arg\max_{y_i \in \mathcal{Y}_i} \max_{y'_i \in \mathcal{Y}, y_i' = y_i} \tilde{p}(y'),
\]

and set \( \mathcal{I} = \{i\} \).

2. Based on (reverse) **depth-first search order**, for each \( j \in \mathcal{V} \setminus \mathcal{I} \)

   (a) choose a configuration \( y_j^* \) at the node \( Y_j \) such that

   \[
y_j^* \in \arg\max_{y_j \in \mathcal{Y}_j} \max_{y'_j \in \mathcal{Y}, y_j' = y_j, y_j^* = y_i} \tilde{p}(y'),
   \]

   (b) update \( \mathcal{I} = \mathcal{I} \cup \{j\} \).
Sum–product algorithm

\[ q_{Y_i \rightarrow F}(y_i) = \prod_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i) \]

\[ r_{F \rightarrow Y_i}(y_i) = \sum_{y_{F'}^{'} \in \mathcal{Y}_{F'}, \ y_i^{'} = y_i} \left( \exp(-E_F(y_{F'}^{'})) \prod_{l \in N(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y_{l}^{'} \right) \]

Max–sum algorithm

\[ q_{Y_i \rightarrow F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i) \]

\[ r_{F \rightarrow Y_i}(y_i) = \max_{y_{F'}^{'} \in \mathcal{Y}_{F'}, \ y_i^{'} = y_i} \left( -E_F(y_{F'}^{'}) + \sum_{l \in N(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y_{l}^{'} \right) \]
Let us consider the following factor graph with binary variables:

Let us chose the node $Y_i$ as root. We calculate the messages for the max-sum algorithm from leaf–to–root direction in a depth-first search order as follows.

1. $q_{Y_l \rightarrow C}(0) = q_{Y_l \rightarrow C}(1) = 0$
2. $r_{C \rightarrow Y_k}(0) = \max_{y_l \in \{0,1\}} \{-E_{C}(0, y_l) + q_{Y_l \rightarrow C}(0)\} = \max_{y_l \in \{0,1\}} -E_{C}(0, y_l) = 0$
   
   $r_{C \rightarrow Y_k}(1) = \max_{y_l \in \{0,1\}} \{-E_{C}(1, y_l) + q_{Y_l \rightarrow C}(1)\} = \max_{y_l \in \{0,1\}} -E_{C}(1, y_l) = 0$

3. $r_{B \rightarrow Y_k}(0) = -1$
   
   $r_{B \rightarrow Y_k}(1) = -0.5$

4. $q_{Y_k \rightarrow A}(0) = r_{B \rightarrow Y_k}(0) + r_{C \rightarrow Y_k}(0) = -1 + 0 = -1$
   
   $q_{Y_k \rightarrow A}(1) = r_{B \rightarrow Y_k}(1) + r_{C \rightarrow Y_k}(1) = -0.5 + 0 = -0.5$
Example (cont.)

5. \( q_{Y_j \rightarrow A}(0) = q_{Y_j \rightarrow A}(1) = 0 \)

6. \[
\begin{align*}
    r_{A \rightarrow Y_i}(0) &= \max_{y_j, y_k \in \{0, 1\}} \{-EA(0, y_j, y_k) + q_{Y_j \rightarrow A}(y_j) + q_{Y_k \rightarrow A}(y_k)\} = -0.5 \\
    r_{A \rightarrow Y_i}(1) &= \max_{y_j, y_k \in \{0, 1\}} \{-EA(1, y_j, y_k) + q_{Y_j \rightarrow A}(y_j) + q_{Y_k \rightarrow A}(y_k)\} = 0.5
\end{align*}
\]

In order to calculate the maximal state \( y^* \) we apply back-tracking

1. \( y_i^* \in \text{argmax}_{y_i \in \{0, 1\}} r_{A \rightarrow Y_i}(y_i) = \{1\} \)

2. \( y_j^* \in \text{argmax}_{y_j} \max_{y_j, y_k \in \{0, 1\}} \{-EA(1, y_j, y_k) + q_{Y_k \rightarrow A}(y_k)\} = \{0\} \)

3. \( y_k^* \in \text{argmax}_{y_k \in \{0, 1\}} \{-EA(1, 0, y_k) + r_{B \rightarrow Y_k}(y_k) + r_{C \rightarrow Y_k}(y_k)\} = \{1\} \)

4. \( y_l^* \in \text{argmax}_{y_l \in \{0, 1\}} \{-EC(1, y_l) + r_{C \rightarrow Y_k}(1)\} = \{1\} \)

Therefore, the optimal state \( y^* = (y_i^*, y_j^*, y_k^*, y_l^*) = (1, 0, 1, 1) \).
Loopy belief propagation
When the graph has cycles, then there is no well-defined *leaf–to–root* order. However, one can apply message passing on cyclic graphs, which results in **loopy belief propagation**.

1. Initialize all messages as constant 1
2. Pass factor–to–variables and variables–to–factor messages alternately until convergence
3. Upon convergence, treat beliefs $\mu_F$ as approximate marginals

The **factor-to-variable messages** $r_{F \rightarrow Y_i}$ remain well-defined and are computed as before:

$$
r_{F \rightarrow Y_i}(y_i) = \sum_{y'_F \in \mathcal{Y}_F, y'_i = y_i} \left( \exp(-E_F(y'_F)) \prod_{j \in N(F) \setminus \{i\}} q_{Y_j \rightarrow F}(y'_j) \right).$$

The **variable-to-factor messages** are simply normalized at every iteration as follows:

$$q_{Y_i \rightarrow F}(y_i) = \frac{\prod_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i)}{\sum_{y'_i \in \mathcal{Y}_i} \prod_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y'_i)}.$$
The approximate marginals, i.e. beliefs, are computed as before, but now a factor-specific normalization constant $z_F$ is also used.

The factor marginals are given by

$$
\mu_F(y_F) = \frac{1}{z_F} \exp(-E_F(y_F)) \prod_{i \in N(F)} q_{Y_i \rightarrow F}(y_i),
$$

where the factor specific normalization constant is given by

$$
z_F = \sum_{y_F \in \mathcal{Y}_F} \exp(-E_F(y_F)) \prod_{i \in N(F)} q_{Y_i \rightarrow F}(y_i).
$$
In addition to the factor marginals the algorithm also computes the **variable marginals** in a similar fashion.

\[
\mu_i(y_i) = \frac{1}{z_i} \prod_{F' \in M(i)} r_{F' \rightarrow Y_i}(y_i),
\]

where the normalizing constant is given by

\[
z_i = \sum_{y_i \in Y_i} \prod_{F' \in M(i)} r_{F' \rightarrow Y_i}(y_i).
\]

Since the local *normalization constant* \(z_F\) differs at each factor for loopy belief propagation, the exact value of the normalizing constant \(Z\) **cannot** be directly calculated. Instead, an **approximation to the partition function** can be computed.
Loopy belief propagation is very popular, but has some problems:

- It might not converge (e.g., it can oscillate).
- Even if it does, the computed probabilities are only approximate.
- If there is a single cycle only in the graph, then it converges.
We have discussed **exact inference** methods on *tree-structured* graphical models

- Probabilistic inference: *Sum-product algorithm*
- MAP inference: *Max-sum algorithm*

For *general* factor graphs: *Loopy belief propagation*

In the **next lecture** we will learn about

- Human-pose estimation

**Mean-field approximation**: probabilistic inference via optimization (a.k.a. variational inference)

