

# Exact Inference in Probabilistic Graphical Models

# 4

## 4.1 Introduction

In the context of inference, our task is, given a joint distribution  $p(x) = p(x_1, \dots, x_n)$ , to compute marginals or conditionals of this distribution.

We formalize this by considering two disjoint subsets of r.v.:  $y \subseteq x, z \subseteq x$  s.t.  $y \cap z = \emptyset$  and  $y \cup z \subseteq x$ .

Given that we observe  $y$  (this is denoted by  $y = \bar{y}$ ), we want to compute the conditional

$$p(z|y = \bar{y}) = \sum_{x'} p(z, x'|y = \bar{y}) \text{ discrete r.v.}$$

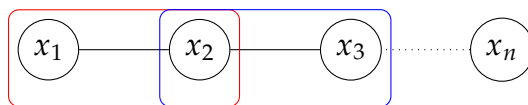
$$= \int p(z, x'|y = \bar{y}) dx' \text{ continuous r.v.}$$

being  $x'$  s.t.  $x' \cup y \cup z = x$ .

**Remark** this summation can be of the order of  $O(\mathcal{K}^m)$  ( $|x'| = m$ ), hence obtaining this marginalization is computationally challenging!

Our goal is to carry out this computation efficiently, by leverage the factorization implied by the PGM.

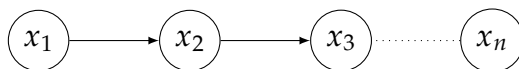
For example, consider a Markov Random Field where variables are connected in a chain (colored rectangles represent the cliques):



The factorization implied by this PGM is:

$$p(x) = \frac{1}{Z} \cdot \Psi_{1,2}(x_1, x_2) \cdot \Psi_{2,3}(x_2, x_3) \cdot \dots \cdot \Psi_{x_{n-1}, x_n}(x_{n-1}, x_n)$$

Consider now the equivalent model in case of a Bayesian Network:



The factorization in this case reads as:

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_2) \cdot \dots \cdot p(x_n|x_{n-1})$$

- 4.1 Introduction . . . . . 35
- 4.2 Factor Graphs . . . . . 37
- 4.2.1 From Bayesian Networks to Factor Graphs . . . . . 38
- 4.2.2 From Markov Random Fields to Factor Graphs . . . . . 39
- 4.3 Sum Product algorithm . . . . . 39
- 4.4 Max Plus algorithm . . . . . 45
- 4.5 Inference in general Probabilistic Graphical Models . . . . . 48

**Note:** chain structures like the previous are common because they represent temporal series (more specifically, they are Markov processes).

Suppose that, given the model above, we want to compute the marginal

$$p(x_k) = \sum_{x_1, \dots, x_{k-1}, x_{k+1}, x_n} p(x_1, \dots, x_n)$$

with  $1 < k < n$ . In principle this has complexity  $O(\mathcal{X}^{n-1})$ .

However, plugging the factorization implied by the Markov Random Field, we get (using the distributive laws of sum and product):

$$\begin{aligned} p(x_k) &= \sum_{x_k} \frac{1}{Z} \Psi_{1,2}(x_1, x_2) \cdots \Psi_{n-1,n}(x_{n-1}, x_n) = \\ &= \frac{1}{Z} \sum_{x_{k-1}} \Psi_{k-1,k}(x_{k-1}, x_k) \cdots \left[ \sum_{x_2} \Psi_{2,3}(x_2, x_3) \left[ \sum_{x_1} \Psi(x_1, x_2) \right] \right] + \\ &+ \sum_{x_{k+1}} \Psi_{k,k+1}(x_k, x_{k+1}) \cdots \left[ \sum_{x_{n-1}} \Psi_{n-2,n-1}(x_{n-2}, x_{n-1}) \left[ \sum_{x_n} \Psi_{n-1,n}(x_{n-1}, x_n) \right] \right] \end{aligned}$$

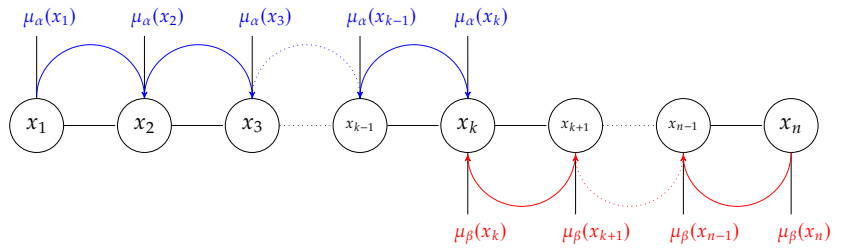
Note that complexity is now  $O(n \cdot k)$ , i.e. linear in  $n$ .

We can define a dynamic programming algorithm, called **message-passing algorithm**, in which the information from the graph is summarized by local edge information.

We break the chain in two parts, the past and the future w.r.t.  $x_k$ , and we define the following:

- ▶ forward message:  $\mu_\alpha(x_k) = \sum_{x_{k-1}} \Psi_{k-1,k}(x_{k-1}, x_k) \cdot \mu_\alpha(x_{k-1})$ ,  
with base case  $\mu_\alpha(x_1) = 1$
- ▶ backward message:  $\mu_\beta(x_k) = \sum_{x_{k+1}} \Psi_{k,k+1}(x_k, x_{k+1}) \cdot \mu_\beta(x_{k+1})$ ,  
with base case  $\mu_\beta(x_n) = 1$

In this algorithm, each node has a message:



Once we have both  $\mu_\alpha(x_k)$  and  $\mu_\beta(x_k)$ , we can observe that:

$$p(x_k) = \frac{1}{Z_k} \cdot \mu_\alpha(x_k) \mu_\beta(x_k) \propto \mu_\alpha(x_k) \mu_\beta(x_k)$$

with the normalization constant  $Z_k$  computed as  $Z_k = \sum_{x_k} \mu_\alpha(x_k) \mu_\beta(x_k)$ .

Summarizing, the idea behind this algorithm is to start from the extremes of the chain and pass messages (forward and backward) until the

other end is reached. Once there, messages are combined to obtain an (un)normalized marginal distribution.

## 4.2 Factor Graphs

Our goal in what follows is to extend what we have done for chains to more general graph structures, i.e. trees and politrees.

In order to do so, we need to introduce the formalism of **Factor Graphs**.

The idea behind Factor Graphs is to expose in a more clear way what are the factors and what are the variables.

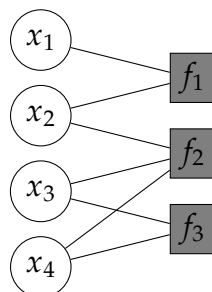
For this reason, Factor Graphs are bipartite graphs (i.e. nodes are divided in two classes), in which we have:

- ▶ variable nodes, denoted by  $x$
- ▶ factor nodes, denoted by  $f$

Consider the distribution (with  $f_i$  factors and  $x_i$  variables):

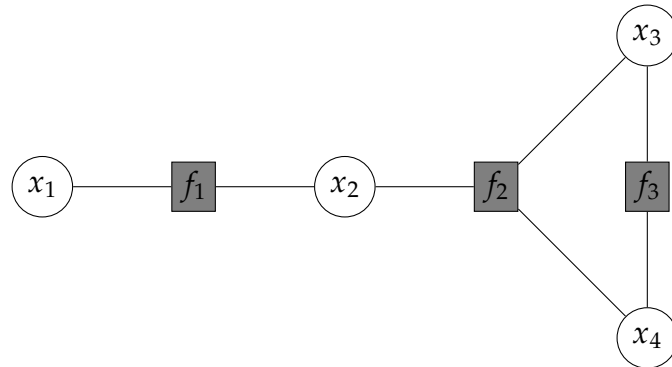
$$p(x) \propto f_1(x_1, x_2) f_2(x_2, x_3, x_4) f_3(x_3, x_4)$$

The canonical way of represent bipartite graphs is the following:



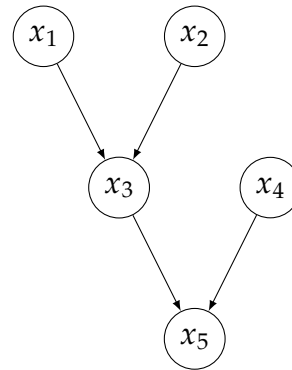
That is, all variable nodes are put on the left and all factor nodes on the right.

A more convenient, equivalent, representation could be:



### 4.2.1 From Bayesian Networks to Factor Graphs

Consider the following Bayesian network:



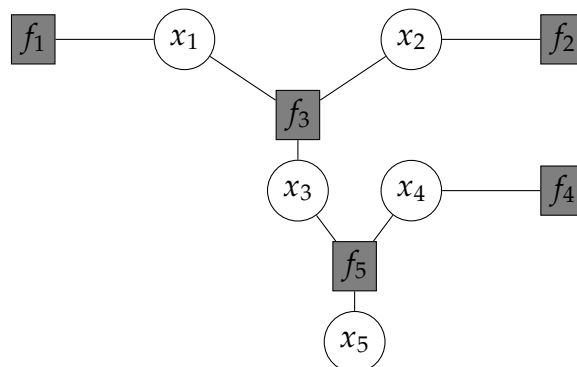
Which corresponds to the factorization:

$$\begin{aligned}
 p(x) &= p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4)p(x_5|x_3, x_4) \\
 &\doteq f_1(x_1)f_2(x_2)f_3(x_1, x_2, x_3)f_4(x_4)f_5(x_3, x_4, x_5)
 \end{aligned}$$

Fundamentally the idea is to have a factor node (connected to the proper variable nodes) for each term of the factorization implied by the Bayesian Network.

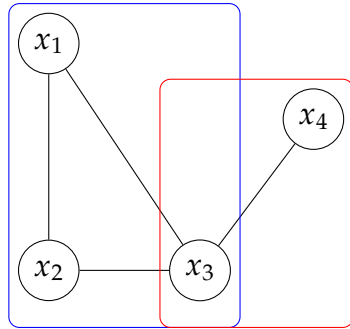
Formally:  $\forall p(x_j|pa_j) \rightarrow f_j \rightsquigarrow \{x_j\} \cup pa_j$

Hence the Factor Graph equivalent to the Bayesian Network above is:



### 4.2.2 From Markov Random Fields to Factor Graphs

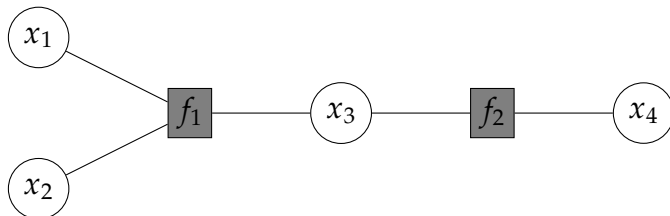
Consider the following Markov Random Field (rectangles correspond to the cliques):



Which implies the factorization:

$$p(x) = \frac{1}{Z} \Psi_1(x_1, x_2, x_3) \Psi_2(x_3, x_4) = \frac{1}{Z} f_1(x_1, x_2, x_3) f_2(x_3, x_4)$$

This leads to the following Factor Graph:

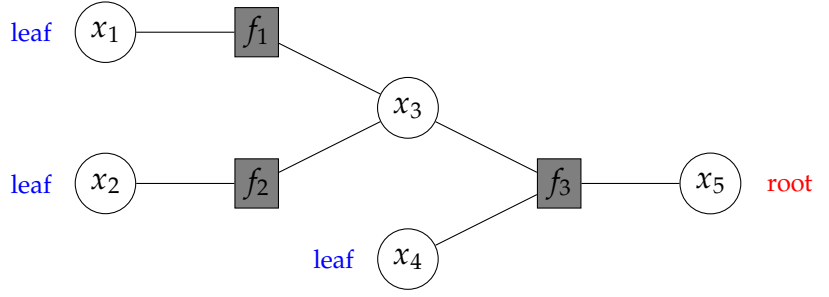


Hence, the conversion from Markov Random Fields to Factor Graphs works as:  $\forall c \in \mathcal{C} \rightsquigarrow f_c(x_c) \approx \Psi_c(x_c)$  (equality does not hold because of the normalization constant).

So, whether we are starting from a Bayesian Network or from a Markov Random Field, we can convert our PGM into a Factor Graph and perform message passing on it, without loss of generality.

## 4.3 Sum Product algorithm

Our goal is now to do inference in Factor Graphs, generalizing to more general graph structures what we did with chains previously. Consider the following Factor Graph:



The joint probability distribution implied by this Factor Graph is:

$$p(x) = f_1(x_1, x_3)f_2(x_2, x_3)f_3(x_3, x_4, x_5)$$

Since the above graph is actually a tree, we can identify the **root** node (choice is arbitrary, here it is  $x_5$ ) and consequently the **leaves** (in the example,  $x_1, x_2, x_4$ ). Recall that there is a unique path from the root to any of the leaves and that following all those paths we visit all the nodes in the tree.

The message passing algorithm that we are going to detail is called **Belief Propagation** and works in Factor Graphs which are trees (i.e., without any loop).

Assume that the root is the node of which we want to compute the marginal (this is not necessarily the case, indeed after performing forward and backward pass we have sufficient information to compute the marginal w.r.t. each node in graph).

So, let's consider the marginal probability w.r.t.  $x_5$ :

$$\begin{aligned} p(x_5) &= \sum_{x_1, x_2, x_3, x_4} p(x_1, x_2, x_3, x_4, x_5) = \\ &= \sum_{x_3, x_4} f_3(x_3, x_4, x_5) \left[ \sum_{x_1} f_1(x_1, x_3) \right] \left[ \sum_{x_2} f_2(x_2, x_3) \right] \end{aligned}$$

Take  $\sum_{x_1} f_1(x_1, x_3)$ : we can see this as a message going from factor  $f_1$  to variable  $x_3$ . We denote it by  $\mu_{f_1 \rightarrow x_3}(x_3)$ .

Analogously,  $\sum_{x_2} f_2(x_2, x_3) \doteq \mu_{f_2 \rightarrow x_3}(x_3)$  is a message from factor  $f_2$  to variable  $x_3$ .

Their product is instead a message flowing from variable  $x_3$  to factor  $f_3$ , denoted by  $\mu_{x_3 \rightarrow f_3}(x_2, x_3) \doteq \sum_{x_1} f_1(x_1, x_3) \cdot \sum_{x_2} f_2(x_2, x_3)$

Finally, the full summation can be seen as a message going from factor  $f_3$  to variable  $x_5$ , i.e.  $\mu_{f_3 \rightarrow x_5}(x_5) \doteq \sum_{x_3, x_4} f_3(x_3, x_4, x_5) [\sum_{x_1} f_1(x_1, x_3)] [\sum_{x_2} f_2(x_2, x_3)]$

We can observe the following:

- messages from factors to variables include a summation (an integral in the case of continuous r.v.) over all the variables on which the factor depends, except for the one we are sending the message to;

- ▶ messages from variables to factors collect via multiplication all the incoming messages from the other factors, except from the one we are sending the message to.

Formalizing, we call:

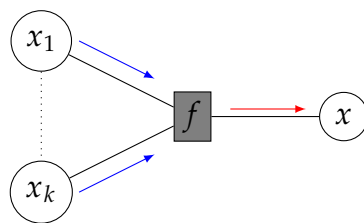
- ▶ set of neighboring nodes of variable  $x$ :

$$\mathcal{N}(x) = \{f_1, \dots, f_k | f_i \text{ is connected to } x\}$$

- ▶ set of neighboring nodes of factor  $f$ :

$$\mathcal{N}(f) = \{x_1, \dots, x_k | x_i \text{ is connected to } f\}$$

The scenario is the following:

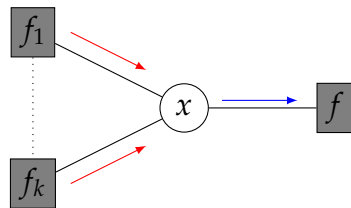


The red arrow in the figure above corresponds to the message  $\mu_{f \rightarrow x}(x)$ . Following what we have seen before, this is computed as:

$$\mu_{f \rightarrow x}(x) = \sum_{x_1, \dots, x_k \in \mathcal{N}(f) \setminus x} f(x, x_1, \dots, x_k) \cdot \prod_{x_i \in \mathcal{N}(f) \setminus x} \mu_{x_i \rightarrow f}(x_i)$$

( $\mu_{x_i \rightarrow f}(x_i)$  are the blue arrows in the figure above).

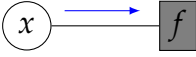
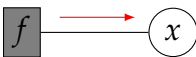
Consider now the symmetric situation:



In this case:

$$\mu_{x \rightarrow f}(x) = \prod_{f_i \in \mathcal{N}(x) \setminus f} \mu_{f_i \rightarrow x}(x)$$

Base cases are defined as:

- ▶  $x$  leaf:   $\mu_{x \rightarrow f}(x) = 1$
- ▶  $f$  leaf:   $\mu_{f \rightarrow x}(x) = f(x)$

The **Sum-product** algorithm works in two steps:

- ▶ forward pass: messages are sent from the leaves to the root;
- ▶ backward pass: messages are sent from the root back to the leaves.

In this way each edge will be traversed both by a forward and a backward message.

After these two passages, all possible messages are computed and we can obtain marginals and conditionals.

Consider the problem of computing the marginals, there are two cases in this scenario:

- computation of the marginal of a variable node  $x$ :

$$p(x) = \prod_{f \in \mathcal{N}(x)} \mu_{f \rightarrow x}(x)$$

- computation of the marginal of a factor node  $f(\bar{x})$  (i.e.,  $p(\bar{x})$  where  $\bar{x} = \mathcal{N}(f)$ ):

$$p(\bar{x}) = f(\bar{x}) \cdot \prod_{x \in \mathcal{N}(f)} \mu_{x \rightarrow f}(x)$$

Note that both these computations can be carried out once we have all the messages.

Moreover, the sum-product algorithm works also when the joint distribution is not normalized, in that case the obtained marginals have to be normalized.

Consider now the problem of computing conditionals on  $y = \hat{y}$ ,  $y \subseteq \{x_1, \dots, x_n\}$ , i.e.  $p(x|y = \hat{y})$ .

The first step is **clamping**  $y$  to  $\hat{y}$ , that is  $p(x, x_1, \dots, x_k, y = \hat{y})$  (i.e. we fix  $y = \hat{y}$  in the joint).

Then we run the sum-product algorithm with  $y_j$  fixed to  $\hat{y}_j \forall y_j \in y$  (practically, when running the message passing algorithm, whenever we have a variable in  $y$  we consider its corresponding state  $\hat{y}$ , instead of summing over it):

$$p(x, y = \hat{y}) = \sum_{x_1, \dots, x_k} p(x, x_1, \dots, x_k, y = \hat{y})$$

Finally, we compute the desired conditional as:

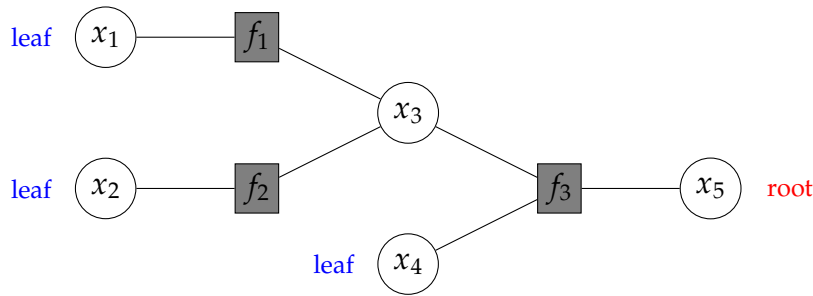
$$p(x|y = \hat{y}) = \frac{p(x, y = \hat{y})}{p(\hat{y})}$$

with  $p(\hat{y}) = \sum_x p(x, y = \hat{y})$ .

In this context, we can see  $p(x, y = \hat{y})$  as an unnormalized conditional and  $p(\hat{y})$  as its normalization constant.

**Example:** consider the following factor graph:





that represents the unnormalized joint distribution

$$p(x_1, x_2, x_3, x_4, x_5) \propto f_1(x_1, x_3)f_2(x_2, x_3)f_3(x_3, x_4, x_5)$$

. Variables are binary, i.e.  $x_i \in \{0, 1\}$  and factors are defined as:

$$f_1(x_1, x_3) = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \text{ where } (x_1, x_3) = \begin{matrix} (0, 0) \\ (0, 1) \\ (1, 0) \\ (1, 1) \end{matrix}$$

$$f_2(x_2, x_3) = \begin{bmatrix} 0.1 \\ 0.5 \\ 0.2 \\ 0.2 \end{bmatrix} \text{ where } (x_2, x_3) = \begin{matrix} (0, 0) \\ (0, 1) \\ (1, 0) \\ (1, 1) \end{matrix}$$

$$f_3(x_3, x_4, x_5) = \begin{bmatrix} 0.1 \\ 0 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0 \\ 0.2 \\ 0.4 \end{bmatrix} \text{ where } (x_3, x_4, x_5) = \begin{matrix} (0, 0, 0) \\ (0, 0, 1) \\ (0, 1, 0) \\ (0, 1, 1) \\ (1, 0, 0) \\ (1, 0, 1) \\ (1, 1, 0) \\ (1, 1, 1) \end{matrix}$$

After arbitrarily choosing  $x_5$  as root node, we can start by computing forward messages edge by edge:

### Forward pass

$$\mu_{x_1 \rightarrow f_1}(x_1) = 1$$

$$\mu_{x_2 \rightarrow f_2}(x_2) = 1$$

$$\mu_{f_1 \rightarrow x_3}(x_3) = \sum_{x_1} f_1(x_1, x_3)\mu_{x_1 \rightarrow f_1}(x_1) = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}$$

$$\mu_{f_2 \rightarrow x_3}(x_3) = \sum_{x_2} f_2(x_2, x_3)\mu_{x_2 \rightarrow f_2}(x_2) = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}$$

$$\mu_{x_3 \rightarrow f_3}(x_3) = \mu_{f_1 \rightarrow x_3}(x_3)\mu_{f_2 \rightarrow x_3}(x_3) = \begin{bmatrix} 0.12 \\ 0.42 \end{bmatrix}$$

$$\mu_{x_4 \rightarrow f_3}(x_4) = 1$$

$$\begin{aligned} \mu_{f_3 \rightarrow x_5}(x_5) &= \sum_{x_3, x_4} f_3(x_3, x_4, x_5) \mu_{x_3 \rightarrow f_3}(x_3) \mu_{x_4 \rightarrow f_3}(x_4) = \\ &= \begin{bmatrix} 0.12 \cdot (0.1 + 0.1) + 0.42 \cdot (0.1 + 0.2) \\ 0.12 \cdot (0 + 0.1) + 0.42 \cdot (0 + 0.4) \end{bmatrix} = \begin{bmatrix} 0.15 \\ 0.18 \end{bmatrix} \end{aligned}$$

### Backward pass

$$\mu_{x_5 \rightarrow f_3}(x_5) = 1$$

$$\begin{aligned} \mu_{f_3 \rightarrow x_4}(x_4) &= \sum_{x_3, x_5} f_3(x_3, x_4, x_5) \mu_{x_5 \rightarrow f_3}(x_5) \mu_{x_3 \rightarrow f_3}(x_3) = \\ &= \begin{bmatrix} 0.12 \cdot (0.1 + 0) + 0.42 \cdot (0.1 + 0) \\ 0.12 \cdot (0.1 + 0.1) + 0.42 \cdot (0.2 + 0.4) \end{bmatrix} = \begin{bmatrix} 0.054 \\ 0.276 \end{bmatrix} \end{aligned}$$

$$\mu_{f_3 \rightarrow x_3}(x_3) = \sum_{x_4, x_5} f_3(x_3, x_4, x_5) \mu_{x_5 \rightarrow f_3}(x_5) \mu_{x_4 \rightarrow f_3}(x_4) = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}$$

$$\mu_{x_3 \rightarrow f_1}(x_3) = \mu_{f_3 \rightarrow x_3}(x_3) \mu_{f_2 \rightarrow x_3}(x_3) = \begin{bmatrix} 0.3 \cdot 0.3 \\ 0.7 \cdot 0.7 \end{bmatrix} = \begin{bmatrix} 0.09 \\ 0.49 \end{bmatrix}$$

$$\mu_{x_3 \rightarrow f_2}(x_3) = \mu_{f_3 \rightarrow x_3}(x_3) \mu_{f_2 \rightarrow x_3}(x_3) = \begin{bmatrix} 0.3 \cdot 0.4 \\ 0.7 \cdot 0.6 \end{bmatrix} = \begin{bmatrix} 0.12 \\ 0.42 \end{bmatrix}$$

$$\mu_{f_1 \rightarrow x_1}(x_1) = \sum_{x_3} f_1(x_1, x_3) \mu_{x_3 \rightarrow f_1}(x_3) = \begin{bmatrix} 0.09 \cdot 0.3 + 0.49 \cdot 0.2 \\ 0.09 \cdot 0.1 + 0.49 \cdot 0.4 \end{bmatrix} = \begin{bmatrix} 0.125 \\ 0.205 \end{bmatrix}$$

$$\mu_{f_2 \rightarrow x_2}(x_2) = \sum_{x_3} f_2(x_2, x_3) \mu_{x_3 \rightarrow f_2}(x_3) = \begin{bmatrix} 0.12 \cdot 0.1 + 0.42 \cdot 0.5 \\ 0.12 \cdot 0.2 + 0.42 \cdot 0.2 \end{bmatrix} = \begin{bmatrix} 0.222 \\ 0.108 \end{bmatrix}$$

Having computed all the messages, we can calculate the marginal for all the nodes, for example:

$$p(x_5) \propto \mu_{f_3 \rightarrow x_5}(x_5) = \begin{bmatrix} 0.15 \\ 0.18 \end{bmatrix} \propto \begin{bmatrix} 0.45 \\ 0.55 \end{bmatrix}$$

$$\begin{aligned} p(x_3) \propto \mu_{f_1 \rightarrow x_3}(x_3) \mu_{f_2 \rightarrow x_3}(x_3) \mu_{f_3 \rightarrow x_3}(x_3) &= \begin{bmatrix} 0.4 \cdot 0.3 \cdot 0.3 \\ 0.6 \cdot 0.7 \cdot 0.7 \end{bmatrix} = \begin{bmatrix} 0.036 \\ 0.294 \end{bmatrix} \propto \\ \begin{bmatrix} 0.11 \\ 0.89 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} p(x_2, x_3) \propto f_2(x_2, x_3) \mu_{x_2 \rightarrow f_2}(x_2) \mu_{x_3 \rightarrow f_2}(x_3) &= \begin{bmatrix} 0.1 \cdot 0.12 \\ 0.5 \cdot 0.42 \\ 0.2 \cdot 0.12 \\ 0.2 \cdot 0.42 \end{bmatrix} = \begin{bmatrix} 0.012 \\ 0.21 \\ 0.024 \\ 0.084 \end{bmatrix} \propto \\ \begin{bmatrix} 0.04 \\ 0.64 \\ 0.07 \\ 0.25 \end{bmatrix} \end{aligned}$$

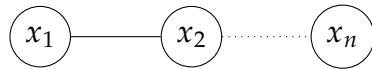
Notice that all marginals were normalized using the same constant  $Z = 0.33$ .

## 4.4 Max Plus algorithm

In what follows our task will be, given a joint density  $p(x)$ , to find the tuple  $x^M = \operatorname{argmax}_x p(x)$  (i.e. find a setting of the variables that has the largest probability and the value of that probability).

To do so, we will use the **max-plus algorithm**.

As an example, consider a chain structure described by a Markov Random Field:



whose factorization reads as

$$p(x) = \frac{1}{Z} \Psi_{1,2}(x_1, x_2) \cdots \Psi_{n-1,n}(x_{n-1}, x_n)$$

Our goal is to maximize  $p(x)$  w.r.t  $x_n$ .

It is possible to distribute the factorization so that only local computations are required:

$$\begin{aligned} \max_x p(x) &= \max_{x_1} \dots \max_{x_n} p(x) = \\ &= \frac{1}{Z} \max_{x_1} \max_{x_2} \Psi_{1,2}(x_1, x_2) \cdots \max_{x_{n-1}} \Psi_{n-2,n-1}(x_{n-2}, x_{n-1}) \max_{x_n} \Psi_{n-1,n}(x_{n-1}, x_n) \end{aligned}$$

This happens because of the distributive properties of the max, indeed it holds that, given  $a > 0$ :

- ▶ the max distributes over the product

$$\max(ab, ac) = a \cdot \max(b, c)$$

- ▶ the max distributes over the sum

$$\max(a + b, a + c) = a + \max(b, c)$$

This allows us to take an approach similar to the one used in the sum-product algorithm.

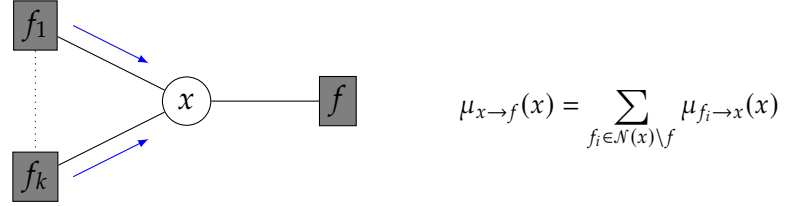
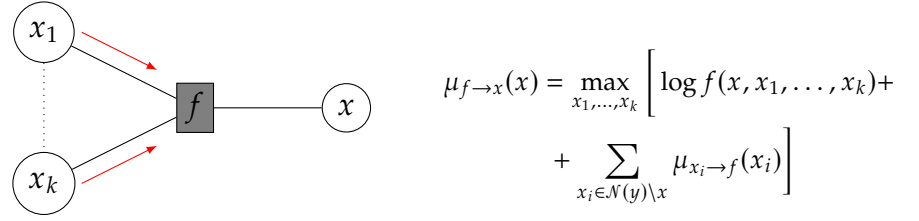
Actually we will maximize  $\log p(x)$ , hence turning the product into sum of logarithms (and this justifies the name max-plus), i.e. our objective is to maximize

$$\log p(x) = \sum_i \log \Psi_{i,i+1}(x_i, x_{i+1}) - \log Z$$

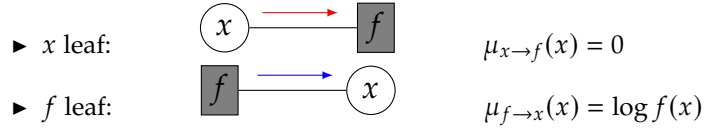
This saves us from product of factors that are possibly very small, since they are probabilities.

Max-plus algorithm is very similar to sum-product: essentially max replaces sum and plus replaces product. This leads to the following

scenarios:



With base cases:



In order to find the maximum of the joint distribution, that is

$$p_{max} = \max_{x_{root}} \left[ \sum_{f \in \mathcal{N}(x_{root})} \mu_{f \rightarrow x_{root}}(x_{root}) \right]$$

we just need to run the forward pass of the algorithm (i.e. we propagate messages from the leaves up to the root, as in the sum-product algorithm).

**Remark:** the result will be the same irrespective of which node is chosen as the root.

However, we want to find also the configuration of the variables for which this maximum is achieved. This can be done by applying a slight variation in the forward pass, meaning that we also need to keep track of which values of the variables gave rise to the maximum state of each variable. So during the forward pass we will also store:

$$\Phi_{f \rightarrow x}(x) = \operatorname{argmax}_{x_1, \dots, x_k \in \mathcal{N}(f) \setminus x} \left[ \log f(x, x_1, \dots, x_k) + \sum_{x_i \in \mathcal{N}(f) \setminus x} \mu_{x_i \rightarrow f}(x_i) \right]$$

Hence, since at the end of the forward pass we know the most probable value of the root node

$$x_{root}^{max} = \operatorname{argmax}_{x_{root}} \left[ \sum_{f \in \mathcal{N}(x_{root})} \mu_{f \rightarrow x_{root}}(x_{root}) \right]$$

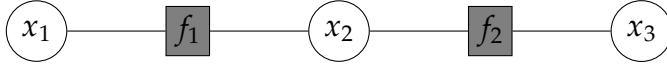
we can exploit the additional information that we stored to retrieve the most probable state of the internal nodes by **back-tracking**, i.e. by following  $\Phi$ . For example, assuming the variable nodes connected to  $f$

are  $x_1, \dots, x_k, x_{root}$ , in the first backtracking step we will fix the value of  $x_1, \dots, x_k$  according to

$$\Phi_{f \rightarrow x_{root}}(x_{root}^{max}) = \begin{cases} x_1 \rightarrow x_1^{max} \\ \dots \\ x_k \rightarrow x_k^{max} \end{cases},$$

and then propagate backwards following  $\Phi$  from the other factor nodes connected to  $x_1, \dots, x_k$ .

**Example:** consider the following chain:



in which variables are binary, i.e.  $x \equiv x_1, x_2, x_3 \in \{0, 1\}$  and factors are defined as:

$$f_1(x_1, x_2) = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \text{ where } (x_1, x_2) = \begin{matrix} (0, 0) \\ (0, 1) \\ (1, 0) \\ (1, 1) \end{matrix}$$

$$f_2(x_2, x_3) = \begin{bmatrix} 0.1 \\ 0.5 \\ 0.2 \\ 0.2 \end{bmatrix} \text{ where } (x_2, x_3) = \begin{matrix} (0, 0) \\ (0, 1) \\ (1, 0) \\ (1, 1) \end{matrix}$$

We start by computing forward messages edge by edge:

$$\mu_{x_1 \rightarrow f_1}(x_1) = 0$$

$$\mu_{f_1 \rightarrow x_2}(x_2) = \max_{x_1} [\log f_1(x_1, x_2) + \mu_{x_1 \rightarrow f_1}(x_1)] = \begin{bmatrix} \log 0.3 \\ \log 0.4 \end{bmatrix} \text{ where } x_2 = \begin{matrix} 0 \\ 1 \end{matrix}$$

$$\mu_{x_2 \rightarrow f_2}(x_2) = \mu_{f_1 \rightarrow x_2}(x_2)$$

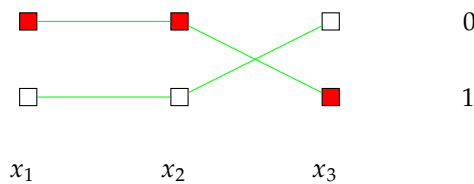
$$\mu_{f_2 \rightarrow x_3}(x_3) = \max_{x_2} [\log f_2(x_2, x_3) + \mu_{x_2 \rightarrow f_2}(x_2)] = \begin{bmatrix} \log 0.2 + \log 0.4 \\ \log 0.5 + \log 0.3 \end{bmatrix} \text{ where } x_3 = \begin{matrix} 0 \\ 1 \end{matrix}$$

as well as backtracking functions:

$$\Phi_{f_1 \rightarrow x_2}(x_2) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ where } x_2 = \begin{matrix} 0 \\ 1 \end{matrix}$$

$$\Phi_{f_2 \rightarrow x_3}(x_3) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ where } x_3 = \begin{matrix} 0 \\ 1 \end{matrix}$$

At this point, we can write down the lattice/trellis diagram:



This kind of diagram shows explicitly the  $K$  (2 in this case) possible states (one per row of the diagram) for each of the variables  $x_i$  in the model. The two paths through the lattice correspond to configurations that give the global maximum of the joint probability distribution.

If, for example, we get that:

$$\max_{x_3} \mu_{f_2 \rightarrow x_3}(x_3) = \log 0.5 + \log 0.3$$

$$\operatorname{argmax} x_3 = 1$$

then we can obtain the tuple that maximizes our joint density by following the path backward in the diagram above (we are guaranteed that this path is unique), starting from  $x_3 = 1$ , i.e.  $(0, 0, 1)$  (red path in the figure).

**Note:** in the context of Hidden Markov Models, the max-plus algorithm is called *Viterbi algorithm*.

## 4.5 Inference in general Probabilistic Graphical Models

In what follows, we want to extend what we have seen so far to general probabilistic graphical models, meaning Factor Graphs which contain loops.

In these cases, we cannot identify the root and the leaves, hence we don't have well defined forward and backward directions.

There are several possibilities:

- ▶ **Junction Tree algorithm:** it roughly builds a tree over the cliques of the Factor Graph, then exact inference is done in this tree. The worst case complexity of this algorithm is exponential in the size of the largest clique (so possibly very heavy computationally);
- ▶ **Loopy Belief Propagation:** forward and backward pass of the sum-product algorithm are iterated several times, until a fix point is reached. Unfortunately there is no guarantee that the algorithm will converge. When it does, it provides an approximate answer to the inference problem;
- ▶ **Monte Carlo Sampling:** a general strategy for approximate inference based on sampling;
- ▶ **Variational Inference:** it approximates the posterior distribution with a simpler distribution belonging to a pre-specified (parametric) class, which is the closer one to the true posterior, minimizing the KL-divergence.