

Inference and the sum-product algorithm

Guillaume Obozinski

Swiss Data Science Center



African Masters of Machine Intelligence, 2018-2019, AIMS, Kigali

Probabilistic inference

Given a discrete Gibbs model of the form:

$$p(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C),$$

where \mathcal{C} is the set of cliques of the graph, inference is the problem of computation of several related quantities of interest:

- Computation of the marginal $p(x_i)$ or more generally $p(x_C)$.
- Computation of the partition function Z
- Computation of the conditional marginal $p(x_i | X_j = x_j, X_k = x_k)$

Rest of this part of the lecture

We will focus in the rest of this part to inference for undirected graphical models that have cliques of size 1 and 2.

$$p(x) = \frac{1}{Z} \prod_{i \in V} \psi_i(x_i) \prod_{\{i,j\} \in E} \psi_{i,j}(x_i, x_j)$$

- The general case is not much more difficult but requires the concept of *factor graph*.
- To perform inference on a directed graph, we will simply moralize the graph and apply the inference algorithm for undirected graphs
- We will focus on the cases where inference can be done efficiently using *dynamic programming*, that is:
 - in the chain case
 - in the tree case

Inference on a chain

We define X_i a random variable, taking value in $\{1, \dots, K\}$, $i \in V = \{1, \dots, n\}$ with joint distribution

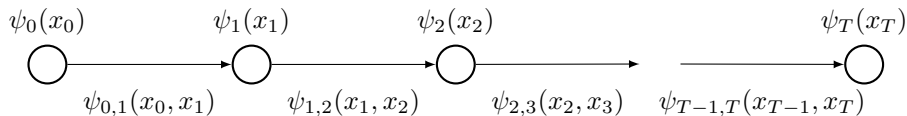
$$p(x) = \frac{1}{Z} \prod_{i=1}^n \psi_i(x_i) \prod_{i=2}^n \psi_{i-1,i}(x_{i-1}, x_i)$$

Computing

$$p(x_j) = \sum_{x_{V \setminus \{j\}}} p(x_1, \dots, x_n)$$

has a priori a complexity of $O(K^n)$.

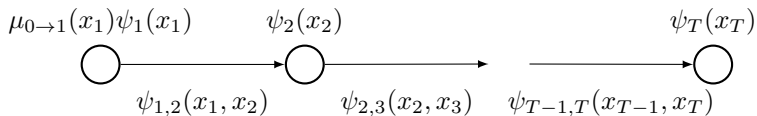
Elimination algorithm on a chain



→ Summing over $x_0 \dots$

- $$\mu_{0 \rightarrow 1}(x_1) = \sum_{x_0} \psi_0(x_0) \psi_{0,1}(x_0, x_1)$$

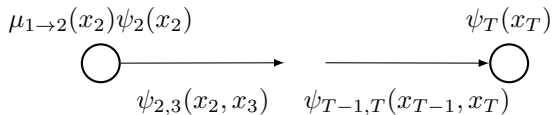
Elimination algorithm on a chain



→ Summing over $x_1 \dots$

- $$\mu_{1 \rightarrow 2}(x_2) = \sum_{x_1} \mu_{0 \rightarrow 1}(x_1) \psi_1(x_1) \psi_{1,2}(x_1, x_2)$$

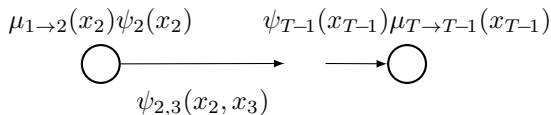
Elimination algorithm on a chain



→ Summing over x_T ...

- $$\mu_{T \rightarrow T-1}(x_{T-1}) = \sum_{x_T} \psi_{T-1,T}(x_{T-1}, x_T) \psi_T(x_T)$$

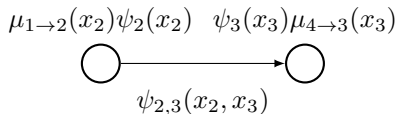
Elimination algorithm on a chain



→ Summing over $x_4, \dots, x_{T-1} \dots$

- $$\mu_{i \rightarrow i-1}(x_{i-1}) = \sum_{x_i} \psi_{i-1,i}(x_{i-1}, x_i) \psi_i(x_i) \mu_{i+1 \rightarrow i}(x_i)$$


Elimination algorithm on a chain



→ Summing over x_3 ...

- $$\mu_{3 \rightarrow 2}(x_2) = \sum_{x_3} \psi_{2,3}(x_2, x_3) \psi_3(x_3) \mu_{4 \rightarrow 3}(x_3)$$

Elimination algorithm on a chain

$$\mu_{1 \rightarrow 2}(x_2)\psi_2(x_2)\mu_{3 \rightarrow 2}(x_2)$$


→ Summing over x_2 ...

- $Z = \sum_{x_2} \mu_{1 \rightarrow 2}(x_2)\psi_2(x_2)\mu_{3 \rightarrow 2}(x_2)$

Elimination algorithm on a chain

Z

→ Done...

Ascending and descending messages

$$\mu_{i \rightarrow i-1}(x_{i-1}) = \sum_{x_i} \psi_{i-1,i}(x_{i-1}, x_i) \psi_i(x_i) \mu_{i+1 \rightarrow i}(x_i)$$

$$\mu_{i \rightarrow i+1}(x_{i+1}) = \sum_{x_i} \mu_{i-1 \rightarrow i}(x_i) \psi_i(x_i) \psi_{i,i+1}(x_i, x_{i+1})$$

Note that node i always receives a message which is a function of x_i .

After propagating all the messages, we have

$$p(x_j) = \frac{1}{Z} \mu_{j-1 \rightarrow j}(x_j) \psi_j(x_j) \mu_{j+1 \rightarrow j}(x_j).$$

How do we compute Z ?

$$1 = \sum_{x_j} p(x_j) \quad \Rightarrow \quad Z = \sum_{x_j} \mu_{j-1 \rightarrow j}(x_j) \psi_j(x_j) \mu_{j+1 \rightarrow j}(x_j)$$

Computing all marginals $p(x_1), p(x_2), \dots, p(x_n)$

Computing $p(x_j)$ requires to compute

$$\boxed{\mu_{1 \rightarrow 2}} \rightarrow \boxed{\mu_{2 \rightarrow 3}} \rightarrow \dots \rightarrow \boxed{\mu_{j-1 \rightarrow j}} \quad \text{and}$$

$$\boxed{\mu_{j+1 \rightarrow j}} \leftarrow \dots \leftarrow \boxed{\mu_{n-1 \rightarrow n-2}} \leftarrow \boxed{\mu_{n \rightarrow n-1}}$$

How do we compute $p(x_{j'})$ for $j' \neq j$? Can use the same messages !

If we compute all $n - 1$ ascending messages

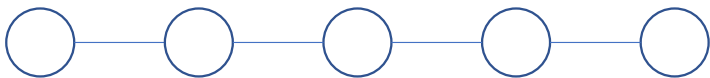
$$\boxed{\mu_{1 \rightarrow 2}} \rightarrow \boxed{\mu_{2 \rightarrow 3}} \rightarrow \dots \rightarrow \boxed{\mu_{j-1 \rightarrow j}} \rightarrow \dots \rightarrow \boxed{\mu_{n-2 \rightarrow n-1}} \rightarrow \boxed{\mu_{n-1 \rightarrow n}}$$


... and all $n - 1$ descending messages


$$\boxed{\mu_{2 \rightarrow 1}} \leftarrow \boxed{\mu_{3 \rightarrow 1}} \leftarrow \dots \leftarrow \boxed{\mu_{j+1 \rightarrow j}} \leftarrow \dots \leftarrow \boxed{\mu_{n-1 \rightarrow n-2}} \leftarrow \boxed{\mu_{n \rightarrow n-1}}$$


then all marginals are computed with an additional $O(n)$ computation.

Message passing on a chain: animation




 Marginal not computed

 Marginal computed






 Model edge

 Message

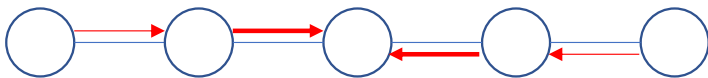
 Active message






Message passing on a chain: animation



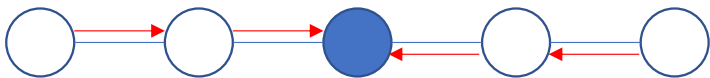
-  Marginal not computed
-  Marginal computed
-  Model edge
-  Message
-  Active message






Message passing on a chain: animation



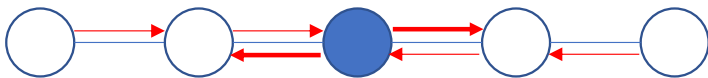
-  Marginal not computed
-  Marginal computed
-  Model edge
-  Message
-  Active message






Message passing on a chain: animation



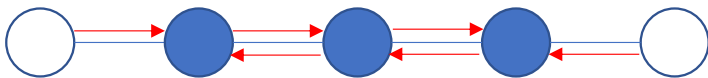
-  Marginal not computed
-  Marginal computed
-  Model edge
-  Message
-  Active message






Message passing on a chain: animation



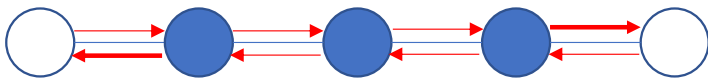
-  Marginal not computed
-  Marginal computed
-  Model edge
-  Message
-  Active message






Message passing on a chain: animation



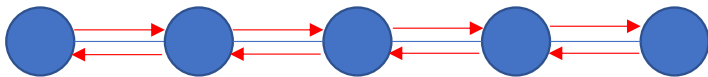
-  Marginal not computed
-  Marginal computed
-  Model edge
-  Message
-  Active message






Message passing on a chain: animation



-  Marginal not computed
-  Marginal computed
-  Model edge
-  Message
-  Active message

Message passing on a chain: animation



-  Marginal not computed
-  Marginal computed
-  Model edge
-  Message
-  Active message

Computing pairwise marginals $p(x_j, x_{j+1})$

Passing

- ascending messages until node j
- descending messages until node $j + 1$

one gets:

$$p(x_j, x_{j+1}) = \frac{1}{Z} \mu_{j-1 \rightarrow j}(x_j) \psi_j(x_j) \psi_{j,j+1}(x_j, x_{j+1}) \psi_{j+1}(x_{j+1}) \mu_{j+2 \rightarrow j+1}(x_{j+1})$$

Inference in undirected trees

Let (V, E) be a tree and assume a tree graphical model

$$p(x) = \frac{1}{Z} \prod_{i \in V} \psi_i(x_i) \prod_{(i,j) \in E} \psi_{i,j}(x_i, x_j)$$

Again we wish to compute $p(x_r)$ for some node r

- Orient the tree by setting node r to be the root
- Let \mathcal{C}_j be the set of children of node j
- Let \mathcal{D}_j be the set of descendants of node j
- Let π_j be the parent of node j

Let

$$F(x_i, x_j, x_{\mathcal{D}_j}) := \psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{D}_j} [\psi_k(x_k) \psi_{\pi_k, k}(x_{\pi_k}, x_k)]$$

Then we have

$$p(x) = \psi_r(x_r) \prod_{i \in \mathcal{C}(r)} F(x_r, x_i, x_{\mathcal{D}_i})$$

Exploiting a recursion on the tree

$$F(x_i, x_j, x_{\mathcal{D}_j}) := \psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{D}_j} [\psi_k(x_k) \psi_{\pi_k, k}(x_{\pi_k}, x_k)]$$

$$F(x_i, x_j, x_{\mathcal{D}_j}) = \psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{C}_j} F(x_j, x_k, x_{\mathcal{D}_k})$$

As a consequence

$$\begin{aligned} \sum_{x_j, x_{\mathcal{D}_j}} F(x_i, x_j, x_{\mathcal{D}_j}) &= \sum_{x_j} \left[\psi_{i,j}(x_i, x_j) \psi_j(x_j) \sum_{x_{\mathcal{D}_j}} \prod_{k \in \mathcal{C}_j} F(x_j, x_k, x_{\mathcal{D}_k}) \right] \\ &= \sum_{x_j} \left[\psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{C}_j} \sum_{x_k, x_{\mathcal{D}_k}} F(x_j, x_k, x_{\mathcal{D}_k}) \right] \end{aligned}$$

So if we define $\mu_{j \rightarrow i}(x_i) := \sum_{x_j, x_{\mathcal{D}_j}} F(x_i, x_j, x_{\mathcal{D}_j})$, then we have

$$\mu_{j \rightarrow i}(x_i) = \sum_{x_j} \psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{C}_j} \mu_{k \rightarrow j}(x_j)$$

Rooted sum-product algorithm: ascending scheme

The recursion

$$\mu_{j \rightarrow i}(x_i) = \sum_{x_j} \psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{C}_j} \mu_{k \rightarrow j}(x_j)$$

defines an algorithm starting at the leaves.

- 1 For any leaf j with parent $i = \pi_j$

$$\mu_{j \rightarrow i}(x_i) = \sum_{x_j} \psi_{i,j}(x_i, x_j) \psi_j(x_j)$$

- 2 Then for any node that only has leaves, we can compute the message to their parents
- 3 etc

At the end of the algorithm:

Each node i has sent a message $\mu_{i \rightarrow \pi_i}(x_{\pi_i})$ to its parent.

Finally computing $p(x_r)$

Remember

$$p(x) = \frac{1}{Z} \psi_r(x_r) \prod_{i \in \mathcal{C}(r)} F(x_r, x_i, x_{\mathcal{D}_i})$$

$$\begin{aligned} p(x_r) &= \sum_{x_{\mathcal{D}_r}} p(x) = \frac{1}{Z} \psi_r(x_r) \sum_{x_{\mathcal{D}_r}} \prod_{i \in \mathcal{C}(r)} F(x_r, x_i, x_{\mathcal{D}_i}) \\ &= \frac{1}{Z} \psi_r(x_r) \prod_{i \in \mathcal{C}(r)} \sum_{x_i, x_{\mathcal{D}_i}} F(x_r, x_i, x_{\mathcal{D}_i}) \\ &= \frac{1}{Z} \psi_r(x_r) \prod_{i \in \mathcal{C}(r)} \mu_{i \rightarrow r}(x_r) \end{aligned}$$

So

$$p(x_r) = \frac{1}{Z} \psi_r(x_r) \prod_{i \in \mathcal{C}(r)} \mu_{i \rightarrow r}(x_r)$$

Towards the sum-product without root

Notice

- if we change the root, the edges whose orientation does not change still transmit the same message !
- the edges whose orientation has changed now send a message in the other direction...

This leads to a general scheme in which each edge will at some point be traversed by a message in each direction.

Message passing without orientation of the tree

Let \mathcal{N}_i denote the set of neighbors of node i .

Consider the update:

$$\mu_{j \rightarrow i}(x_i) = \sum_{x_j} \psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{N}_j \setminus \{i\}} \mu_{k \rightarrow j}(x_j)$$

Key remark: a node can only send a message if it has received messages from **all of its neighbors except one**.

Computing the marginals from the messages

At each node $i \in V$

$$p(x_i) = \frac{1}{Z} \psi_i(x_i) \prod_{k \in \mathcal{N}_i} \mu_{k \rightarrow i}(x_i).$$

For each edge,

$$p(x_i, x_j) = \frac{1}{Z} \psi_i(x_i) \psi_j(x_j) \psi_{ij}(x_i, x_j) \left(\prod_{\substack{k \in \mathcal{N}_i \\ k \neq j}} \mu_{k \rightarrow i}(x_i) \right) \left(\prod_{\substack{\ell \in \mathcal{N}_j \\ \ell \neq i}} \mu_{\ell \rightarrow j}(x_j) \right)$$

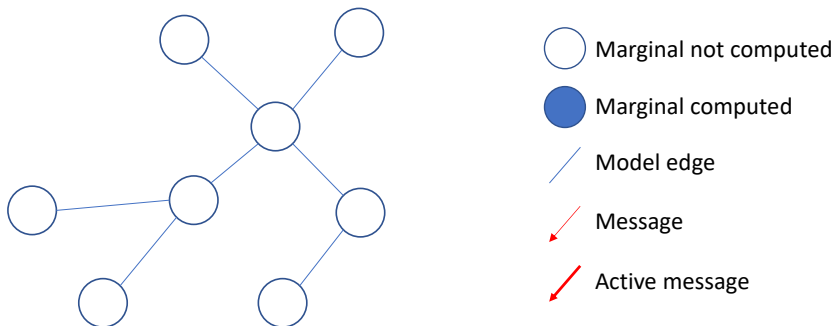
Sum-product algorithm: sequential version

At each iteration:

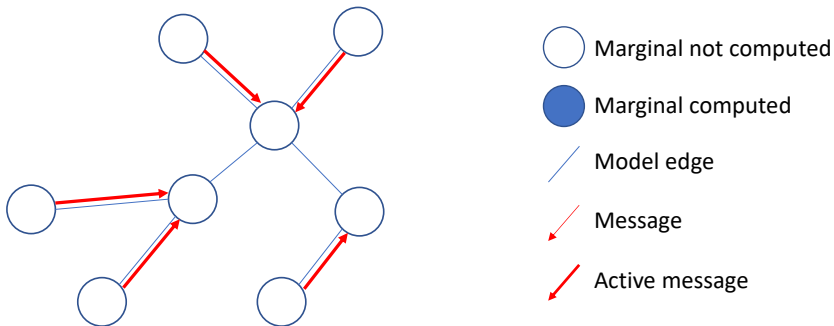
Any node j that has received messages from **all but one** – call it i – of its neighbors sends a message $\mu_{j \rightarrow i}(x_i)$ to i with

$$\mu_{j \rightarrow i}(x_i) = \sum_{x_j} \psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{N}_j \setminus \{i\}} \mu_{k \rightarrow j}(x_j)$$

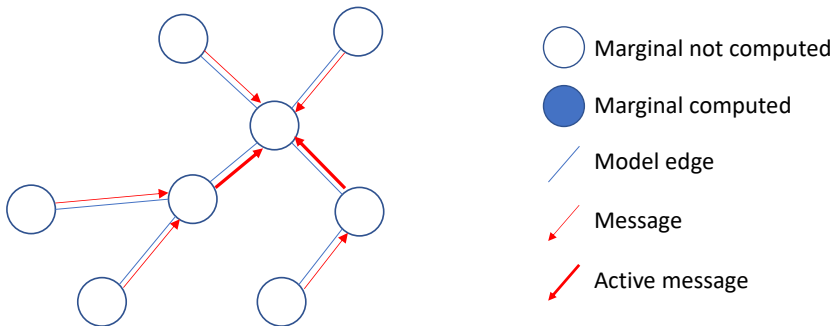
Sequential message passing on a tree: animation



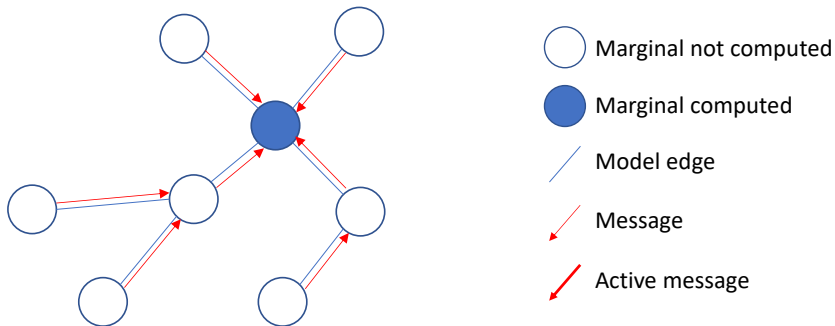
Sequential message passing on a tree: animation



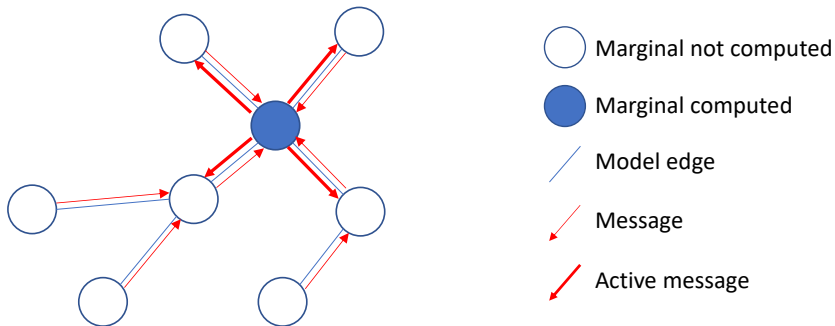
Sequential message passing on a tree: animation



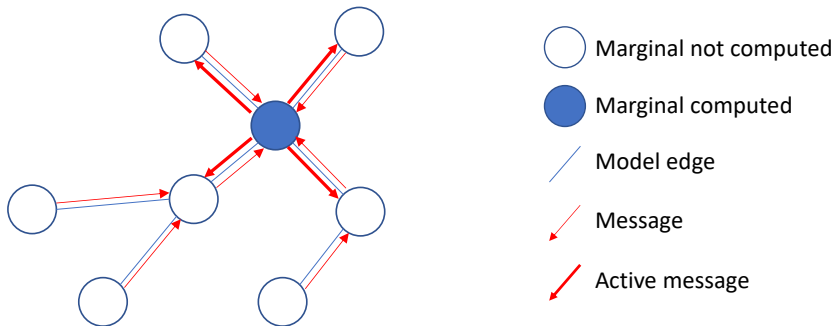
Sequential message passing on a tree: animation



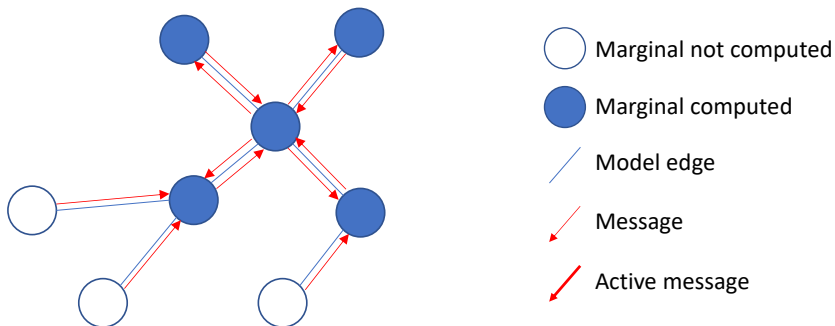
Sequential message passing on a tree: animation



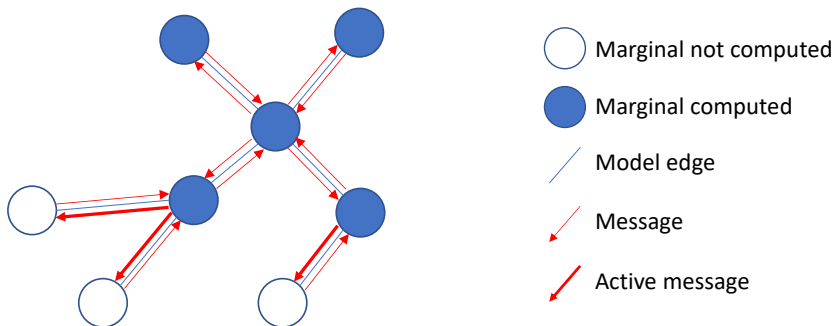
Sequential message passing on a tree: animation



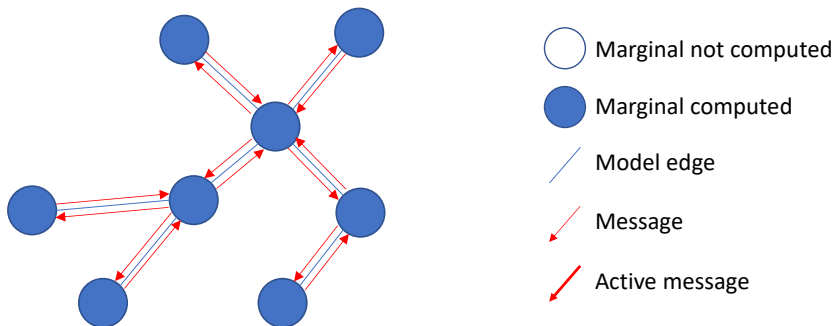
Sequential message passing on a tree: animation



Sequential message passing on a tree: animation



Sequential message passing on a tree: animation



Flooding sum-product algorithm (aka parallel SP)

Initialize all messages at random

At each iteration:

For each node j

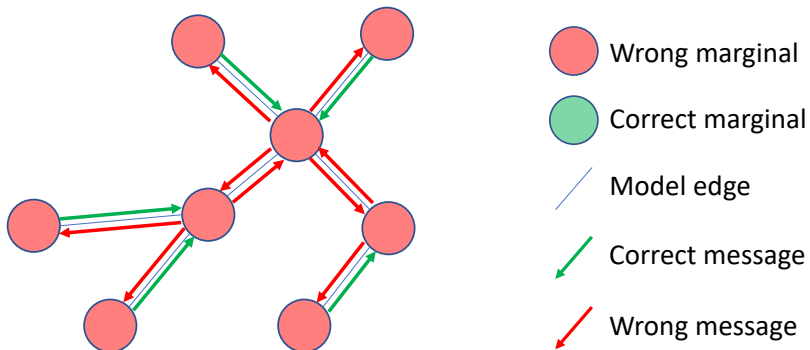
For each neighbor $i \in \mathcal{N}_j$

- j sends to i the message $\mu_{j \rightarrow i}(x_i)$ with

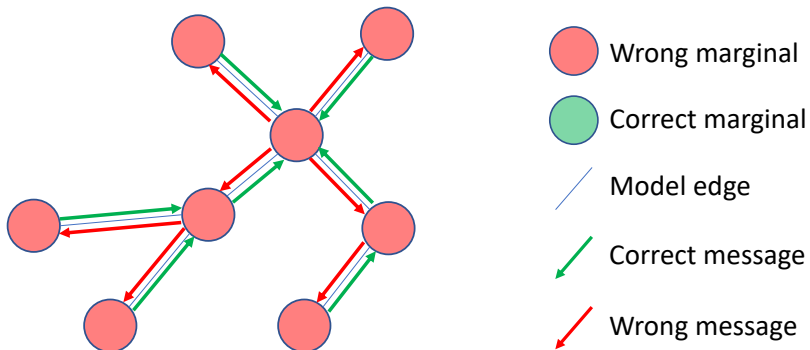
$$\mu_{j \rightarrow i}(x_i) = \sum_{x_j} \psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{N}_j \setminus \{i\}} \mu_{k \rightarrow j}(x_j)$$

The algorithm converges after a number of iterations equal to the diameter of the tree.

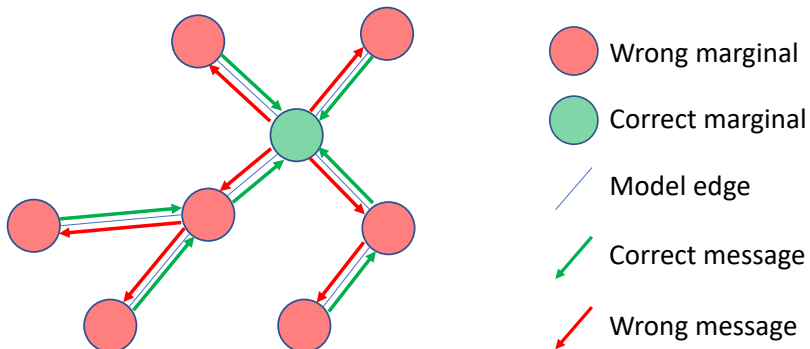
Flooding sum-product: animation



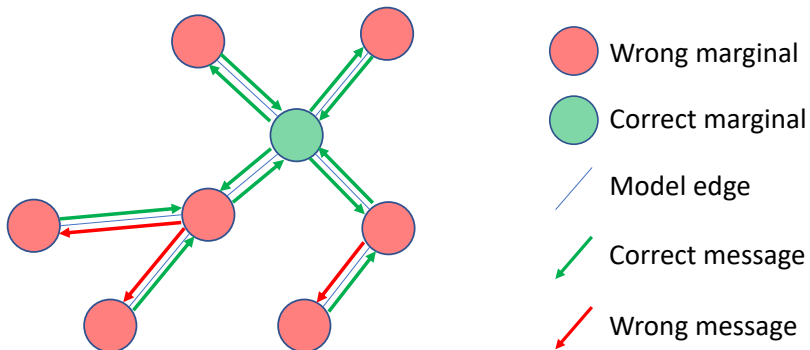
Flooding sum-product: animation



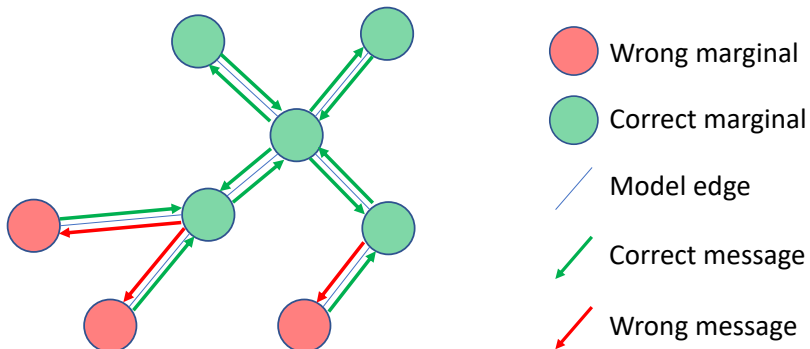
Flooding sum-product: animation



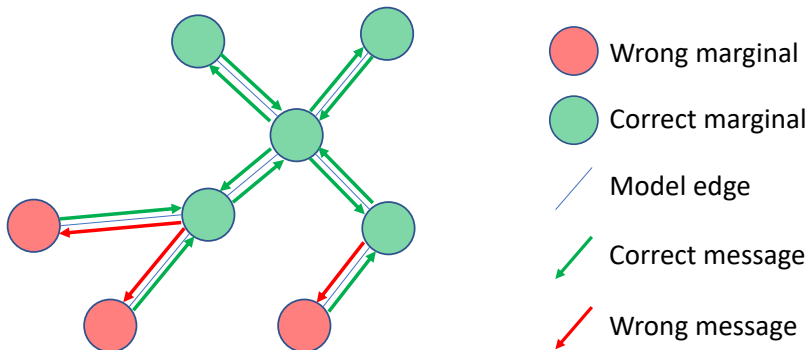
Flooding sum-product: animation



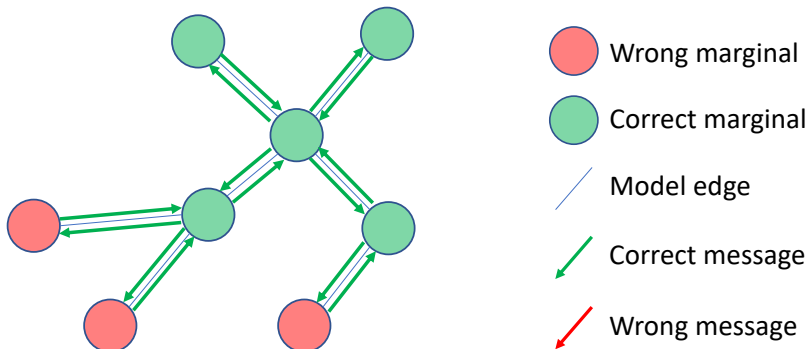
Flooding sum-product: animation



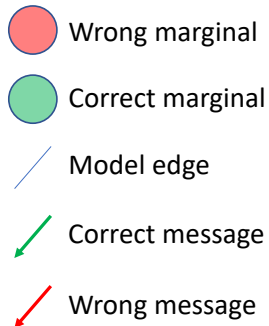
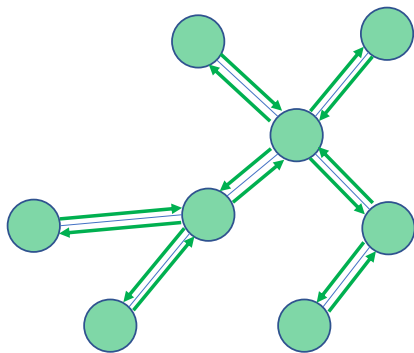
Flooding sum-product: animation



Flooding sum-product: animation



Flooding sum-product: animation



Computing *conditional marginals* $p(x_i \mid X_B = x_{B0})$

Example

$$p(x_i | x_5 = 3, x_{10} = 2) \propto p(x_i, x_5 = 3, x_{10} = 2)$$

Idea: redefine potentials

$$\tilde{\psi}_5(x_5) = \psi_5(x_5) \delta(x_5, 3)$$

Computing *conditional marginals* $p(x_i \mid X_B = x_{B0})$

Given observations $X_j = x_{j0}$ for $j \in B$, define modified potentials:

$$\forall j \in H, \tilde{\psi}_j(x_j) = \psi_j(x_j) \quad \forall j \in B, \tilde{\psi}_j(x_j) = \psi_j(x_j)\delta(x_j, x_{j0})$$

Proposition

$$\text{Let } p(x) = \frac{1}{Z} \prod_{i \in V} \psi_i(x_i) \prod_{(i,j) \in E} \psi_{i,j}(x_i, x_j)$$

$$\text{and } q(x) = \frac{1}{\tilde{Z}} \prod_{i \in V} \tilde{\psi}_i(x_i) \prod_{(i,j) \in E} \psi_{i,j}(x_i, x_j)$$

$$\text{then } q(x) = \mathbb{P}(X = x \mid X_B = x_{B0}) \quad \text{and} \quad \frac{\tilde{Z}}{Z} = \mathbb{P}(X_B = x_{B0}).$$

Proof.

By construction, $q(x) = c p(x) \delta(x_H, x_{B0})$ with $c = \frac{Z}{\tilde{Z}}$. But
 $1 = \sum_x q(x) = c \sum_{x_H} p(x_H, x_{B0}) = c p(x_{B0}).$

□

Inference beyond trees

- The SPA is also called *belief propagation* or *message passing*.
- On trees, it is an exact inference algorithm.
- If G is not a tree, the algorithm can be extended to *loopy belief propagation* but
 - It does not converge in general to the correct marginals
 - it sometimes gives reasonable approximations.

Junction trees

- The exact SPA can be extended to graphs that are close to trees, in the sense that they can be viewed as trees if nodes can be grouped together into cliques to form a tree.
- Elegant theory, but rarely used in practice.

Generalized distributive law

We needed to compute

$$p(x_{i_0}) = \sum_{x_{V \setminus \{i_0\}}} \left[\prod_{i \in V} \psi_i(x_i) \prod_{\{i,j\} \in E} \psi_{ij}(x_i, x_j) \right]$$

The only key property we used is that

The addition is *distributive* over the product

$$(a + b)(c + d) = ac + bc + ad + bd$$

Max product for decoding (MAP inference)

Decoding consists in computing the most probable configuration

$$p(x^*) = \max_{x_V} \left[\prod_{i \in V} \psi_i(x_i) \prod_{\{i,j\} \in E} \psi_{ij}(x_i, x_j) \right]$$

Remark: this is often the most probable configuration of a conditional model (given some input data).

Decoding (aka MAP inference) with max-product

Apply the same algorithms as for SPA, but replace the update with

$$\check{\mu}_{j \rightarrow i}(x_i) = \max_{x_j} \psi_{i,j}(x_i, x_j) \psi_j(x_j) \prod_{k \in \mathcal{N}_j \setminus \{i\}} \check{\mu}_{k \rightarrow j}(x_j)$$

- Known as the Viterbi algorithm in the case of chains.

Max-sum algorithm

Same on log-scale:

$$\log \check{\mu}_{j \rightarrow i}(x_i) = \max_{x_j} \left[\log \psi_{i,j}(x_i, x_j) + \log \psi_j(x_j) + \sum_{k \in \mathcal{N}_j \setminus \{i\}} \log \check{\mu}_{k \rightarrow j}(x_j) \right]$$

Other inference methods

- There are many (exact/approximate/inexact) inference algorithms
- For a general graph exact inference is NP-hard
- The sum-product algorithm (SPA) performs inference on trees, with a complexity linear in the number of nodes.
- SPA can be generalized to graph that are close to trees in some sense using the *junction tree theory*.
- In general, one needs to use approximate or inexact methods
 - Gibbs sampling, and other MCMC sampling methods
 - Variational methods
 - Mean field (/Structured mean field)
 - Loopy belief propagation
 - TRW-entropy based inference

References I