**Factor Graph Reminder**

\[ p(x) = p(x_1)p(x_2)p(x_3|x_1,x_2) \]

**Sum-product algorithm**

Generalizes what we did with chains to compute the marginal of a selected node, \( x_n \).

Generalizes and simplifies an algorithm introduced as “belief propagation”.

\[ \sum_A F(x,X_A) = \sum_A f(x,x_{A1},x_{A2})F_{A1}(x_{A1},A1)F_{A2}(x_{A2},A2) \]

Considering the first factor in the product on the previous slide,

\[ \sum_A F(x,X_A) = \sum_A f(x,x_{A1},x_{A2})F_{A1}(x_{A1},A1)F_{A2}(x_{A2},A2) \]
Sum-product algorithm

We can continue on recursively until we get to the leaf nodes.

However, a message passing implementation is simpler, and is better suited to computing all marginals at once.

We defined two kinds of messages:
1) From nodes to factors.
2) From factors to nodes.

In analogy with chains, factor-to-node messages provide marginal distributions for the node, for a subgraph. (In the chain case, we had the left side and the right side).

In the chain case we did not have factor nodes. This worked because then second kind of message is just “pass through” or “copy” in the case of only two links. So, we described it as simply passing messages from node to node.

Marginal distribution for a node $x$

$$p(x) = \sum_{k \in \text{vars}(x)} \prod_{i \in \text{vars}(x)} F(x, X_i)$$  
(marginalize)

$$= \prod_{i \in \text{vars}(x)} \left\{ \sum_{x_i} F(x, X_i) \right\}$$  
(interchange sums and products)

(recall our fancy formula)

$$\left( \sum a_i \right) \left( \sum b_j \right) = \sum \sum a_i b_j$$

Note that each sum is simpler than what we started with because the variable sets are disjoint except for $x$. 

This factor expands to

$$\sum F(x, X_s) = \sum f(x, x_{a1}, x_{a2}) F_{a1}(x_{a1}, A1) F_{a2}(x_{a2}, A2)$$

$$= \sum f(x, x_{a1}, x_{a2}) \sum F_{a1}(x_{a1}, A1) \sum F_{a2}(x_{a2}, A2)$$

This factor expands to

$$\sum_{A1} F_{a1}(x_{a1}, A1) = \prod_{m \in \text{connected to } a1} F_{a1}(x_{a1}, A1)$$
Factor → node messages

\[ p(x) = \sum_{x_i \in \text{Fun}(x)} \prod_{x_i} F(x_i, x_i) \]

\[ = \prod_{x_i \in \text{Fun}(x)} \left\{ \sum_{x_i} F(x_i, x_i) \right\} \]

\[ = \prod_{x_i \in \text{Fun}(x)} \mu_{f_i \rightarrow x}(x) \]

\[ \mu_{f_i \rightarrow x}(x) = \sum_{x_i} F(x_i, x_i) \]

Computing the factor → node messages

\[ \mu_{f_i \rightarrow x}(x) = \sum_{x_i} F_i(x_i, x) \] (sum removes all variables except x.)

Where \[ F_i(x_i, x) = f_i(x_i, x_1, x_2, \ldots, x_M)G_i(x_i, x_1)G_2(x_i, x_2) \ldots G_M(x_M, x_M) \]

\[ \sum_{x_i} F_i(x_i, x) = \sum_{x_i} \sum_{x_1} \cdots \sum_{x_M} f_i(x_i, x_1, x_2, \ldots, x_M) \sum_{x_1} G_i(x_i, x_1) \sum_{x_2} G_2(x_2, x_1) \cdots \sum_{x_M} G_M(x_M, x_M) \]

(We get the above by moving products outside sums where possible)

So, \[ \sum_{x_i} F_i(x_i, x) = \sum_{x_i} \cdots \sum_{x_M} f_i(x_i, x_1, x_2, \ldots, x_M) \prod_{\text{mon}(i,j)} \sum_{x_m} G_m(x_m, x_m) \]

Computing the factor → node messages

\[ \sum_{x_i} F_i(x_i, x) = \sum_{x_i} \sum_{x_1} \cdots \sum_{x_M} f_i(x_i, x_1, x_2, \ldots, x_M) \prod_{\text{mon}(i,j)} \sum_{x_m} G_m(x_m, x_m) \]

\[ = \sum_{x_i} \cdots \sum_{x_M} f_i(x_i, x_1, x_2, \ldots, x_M) \prod_{\text{mon}(i,j)} \mu_{x_n \rightarrow f}(x_n) \]

where we define:

\[ \mu_{x_n \rightarrow f}(x_n) = \sum_{x_m} G_m(x_m, x_m) \] (node → factor messages)
Summary of computation for factor → node message

\[
\mu_{\text{factor} \to \text{node}}(x) = \sum_{x_i} \cdots \sum_{x_M} f(x, x_1, \ldots, x_M) \prod_{m \in \text{n}(f_i)} \mu_{\text{node} \to \text{factor}}(x_m)
\]

The node → factor message

(We have defined)

\[
\mu_{\text{node} \to \text{factor}}(x_m) = \sum_{x_m} G_m(x_m, X_{s_m})
\]

(For a node \(x_m\) we send its distribution with the other variables in the subgraph marginalized out.)

Computing the node → factor message

\[
\mu_{\text{node} \to \text{factor}}(x_m) = ?
\]

This is just like the first case, but we exclude the node we are sending to.

\[
p(x) = \prod_{m \in \text{n}(x)} \mu_{\text{node} \to \text{factor}}(x_m)
\]

Recall we started with

Nodes that only have two links just pass the message through (i.e., in the chain we skipped this part).
Review in pictures

The sum-product algorithm (1)

We could implement what we have just described as recursion, but the local view of nodes getting and passing messages suggests:

Pass messages from leaves to root. If you just want more than one marginal or plan to do other computation, store the results.

Initialization: If leaf node is a variable node, then start with a unity message. If leaf node is factor, then start with the factor.

\[
\mu_{x \to f} = \frac{1}{Z} \prod_{i} f_{i}(x_{m}, X_{m})
\]

The root node can compute the needed marginal.

The sum-product algorithm (2)

We could implement what we have just described as recursion, but the local view of nodes getting and passing messages suggests:

Pass messages from leaves to root. If you just want more than one marginal or plan to do other computation, store the results.

Initialization: If leaf node is a variable node, then start with a unity message. If leaf node is factor, then start with the factor.

Note that all needed messages for computation will arrive at each node eventually.

The root node can compute the needed marginal.

The sum-product algorithm (3)

To prepare for other computations (e.g., all marginals), pass messages from the root to the leaves.

Now every node has incoming messages on all its links, and can thus be considered the root.

Hence we can compute all marginals for twice the cost of computing one of them.

(From before, also note that all messages that can be passed, have now been passed).
The sum-product algorithm (4)

Another easy computation is the marginal for the group of variables in a factor.

Intuitively (and easily shown---homework) this is given by:

\[ p(x) = f(x) \prod_{i \in \text{in}(f_i)} \mu_{x \rightarrow f_i}(x) \]

The sum-product algorithm example

Let \( \bar{p}(x) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_3, x_4) \)

First we pass messages from leaves to root.

Declare \( x_3 \) as root node.
\[ \mu_{x_1 \rightarrow f_a}(x_1) = 1 \]
\[ \mu_{f_a \rightarrow x_2}(x_2) = \sum_{x_1} f_a(x_1, x_2) \]
\[ \mu_{f_c \rightarrow x_2}(x_2) = \sum_{x_4} f_c(x_2, x_4) \]

\[ \mu_{f_{i \rightarrow x}}(x) = \sum_{x_1} \cdots \sum_{x_M} f(x, x_1, \ldots, x_M) \prod_{\text{node} \rightarrow \text{factor}} \mu_{x_n \rightarrow f_i}(x_n) \]

\[ \mu_{x_2 \rightarrow f_b}(x_2) = \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2) \]
\[ \mu_{f_b \rightarrow x_3}(x_3) = \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \rightarrow f_b}(x_2) \]

Summary of messages from leaves to root:
\[ \mu_{x_1 \rightarrow f_a}(x_1) = 1 \]
\[ \mu_{f_a \rightarrow x_2}(x_2) = \sum_{x_1} f_a(x_1, x_2) \]
\[ \mu_{x_4 \rightarrow f_c}(x_4) = 1 \]
\[ \mu_{f_c \rightarrow x_2}(x_2) = \sum_{x_4} f_c(x_2, x_4) \]
\[ \mu_{x_2 \rightarrow f_b}(x_2) = \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2) \]
\[ \mu_{f_b \rightarrow x_3}(x_3) = \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \rightarrow f_b}(x_2) \]
Next we pass messages from root to leaves.

\[ \mu_{x_1 \rightarrow f_b} (x_3) = 1 \]

\[ \mu_{f_b \rightarrow x_2} (x_2) = \sum_{x_3} f_b (x_2, x_3) \]

Candidate for third and fourth?

Lets go towards \( x_1 \) first.

\[ \mu_{x_2 \rightarrow f_a} (x_2) = \mu_{f_b \rightarrow x_2} (x_2) \mu_{f_a \rightarrow x_2} (x_2) \]

\[ \mu_{f_a \rightarrow x_1} (x_1) = \sum_{x_2} f_a (x_1, x_2) \mu_{x_2 \rightarrow f_a} (x_2) \]

Summary of messages from root to leaves.

\[ \mu_{x_3 \rightarrow f_b} (x_3) = 1 \]

\[ \mu_{f_b \rightarrow x_2} (x_2) = \sum_{x_3} f_b (x_2, x_3) \]

\[ \mu_{x_2 \rightarrow f_a} (x_2) = \mu_{f_b \rightarrow x_2} (x_2) \mu_{f_a \rightarrow x_2} (x_2) \]

\[ \mu_{f_a \rightarrow x_1} (x_1) = \sum_{x_2} f_a (x_1, x_2) \mu_{x_2 \rightarrow f_a} (x_2) \]

\[ \mu_{x_2 \rightarrow f_c} (x_2) = \mu_{f_a \rightarrow x_2} (x_2) \mu_{f_b \rightarrow x_2} (x_2) \]

\[ \mu_{f_c \rightarrow x_4} (x_4) = \sum_{x_2} f_c (x_2, x_4) \mu_{x_2 \rightarrow f_c} (x_2) \]
Handling observed variables

Usually we have observed variables (e.g., evidence). We simply clamp those variables to their observed values.

More formally, denote hidden variables by $h$, and observed ones by $v$. Denote the observed value as $\hat{v}$. For each observed variable, $v_i$, with value $\hat{v_i}$, we can introduce factors into the graph

$$I(v_i, \hat{v}_i) = \begin{cases} 1 & \text{if } v_i = \hat{v}_i \\ 0 & \text{otherwise} \end{cases}$$

Then, $p(h, v = \hat{v}) = p(h, v) \prod_i I(v_i, \hat{v}_i)$

(needs to be normalized to get $p(h|\hat{v})$, but this is easy since we are doing sum-product.)

Max-sum algorithm

Method to compute.

$$x^\text{max} = \arg \max_x p(x)$$

i.e., $p(x^\text{max}) = \max_x p(x)$

Recall inference on chains

$$p(x) = \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3) \ldots \psi_{N-2,N-1}(x_{N-2}, x_{N-1})\psi_{N-1,N}(x_{N-1}, x_N)$$

Naive computation of $\arg \max_x p(x)$ would evaluate the above for each value of $x$, and take the max.

Too expensive!!
Recall speeding up marginalization

\[ p(x_n) = \left( \sum_{x_{n-1}} \cdots \sum_{x_1} \prod_{i=1}^{n-1} \psi_{n-i,n-i}(x_{n-i}, x_{n-i+1}) \right) \left( \sum_{x_{n-1}} \cdots \sum_{x_1} \prod_{i=1}^{n-1} \psi_{i,i+1}(x_i, x_{i+1}) \right) \]

\[ \sum_{x_{n-1}} \cdots \sum_{x_1} \prod_{i=1}^{n-1} \psi_{n-i,n-i}(x_{n-i}, x_{n-i+1}) = \left[ \sum_{x_n} \psi_{n-1,n}(x_{n-1}, x_n) \right] \cdots \left[ \sum_{x_2} \psi_{1,2}(x_1, x_2) \right] \left[ \sum_{x_1} \psi_{0,1}(x_0, x_1) \right] \]

and

\[ \sum_{x_{n-1}} \cdots \sum_{x_1} \prod_{i=1}^{n-1} \psi_{i,i+1}(x_i, x_{i+1}) = \left[ \sum_{x_n} \psi_{n-1,n}(x_{n-1}, x_n) \right] \cdots \left[ \sum_{x_2} \psi_{1,2}(x_1, x_2) \right] \left[ \sum_{x_1} \psi_{0,1}(x_0, x_1) \right] \]

What if we could do with max() what we are doing with \( \Sigma \)?

Max on a chain

In analogy with marginals on a chain,

\[ \max_{x_n} p(x) = \frac{1}{Z} \max_{x_n} \left[ \max_{x_{n-1}} \cdots \max_{x_1} \prod_{i=1}^{n-1} \psi_{n-i,n-i}(x_{n-i}, x_{n-i+1}) \right] \left[ \max_{x_{n-1}} \cdots \max_{x_1} \prod_{i=1}^{n-1} \psi_{i,i+1}(x_i, x_{i+1}) \right] \]

where,

\[ \max_{x_n} \cdots \max_{x_1} \left( \prod_{i=1}^{n-1} \psi_{n-i,n-i}(x_{n-i}, x_{n-i+1}) \right) = \max_{x_n} \left\{ \max_{x_{n-1}} \left( \max_{x_{n-2}} \left( \cdots \left( \max_{x_1} \left( \psi_{0,1}(x_0, x_1) \right) \right) \right) \right) \right\} \]

and similarly for the part in red.

Max-sum algorithm

Two steps.

1) Compute the max while remembering certain computations
2) Compute a value of \( x \) that achieves the max

The message passing algorithm for step (1) is clear from the analog with the “sum-product” algorithm, except that it would then be called the “max-product” algorithm.

Computing long products loses precision (*), so we switch to log(), and call it the max-sum algorithm.

(*) Less of an issue with marginalization.