Inference on graphs

- Given a graph and its conditionals or potentials compute
  \[ p(\theta | e) \] (particular \( \theta \) and \( e \), marginalizing out other variables)
  \[ p(X) \] (particular event, marginalizing out other variables)
  \[ \text{argmax } p(\theta | e) \] (particular \( \theta \) and \( e \), marginalizing other variables)
  \[ \text{argmax } p(\theta, \theta_n, e_n | e) \] (all variables, will nuisance / unobserved)

Marginals on a chain

Recall
\[ p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2) \ldots p(x_{N-1}|x_{N-2})p(x_N|x_{N-1}) \]
Converted to
\[ p(x) = \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3) \ldots \psi_{N-2,N-1}(x_{N-2}, x_{N-1})\psi_{N-1,N}(x_{N-1}, x_N) \]
Assume \( N \) discrete variables, with \( K \) values each.

Compute the marginal of a node in the middle, \( p(x_n) \)

Inference on graphs

- Simplest example (Bayes’ rule)
  - (a) model
  - (b) illustrates observed
  - (c) inference reverses the arrow

- Computationally
  \[ p(x|y) = \frac{p(y|x)p(x)}{\sum_{x'} p(y|x')p(x')} \]
Marginals on a chain

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

Direct calculation of \( p(x_n) \)

\[ p(x_n) = \sum_{x_1} \sum_{x_2} \ldots \sum_{x_{n-1}} \sum_{x_{n+1}} \ldots \sum_{x_N} p(x) \]

Computational complexity is \( O(K^N) \). Way too slow!

Fancy formulas from algebra

\[ \sum_{x_1, x_2, \ldots, x_N} f(x_1, x_2, \ldots, x_N) = \sum_{x_1} \sum_{x_2} \ldots \sum_{x_N} f(x_1, x_2, \ldots, x_N) \]

(all values of each)

(essentially a definition)

\[ (\sum a_i)(\sum b_j) = \sum \sum a_i b_j \]

Back to marginals on a chain

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

\[ p(x_n) = \sum_{x_1} \sum_{x_2} \ldots \sum_{x_{n-1}} \sum_{x_{n+1}} \ldots \sum_{x_N} f_L(x_1, x_2, \ldots, x_N) \]

\[ = \left( \sum_{x_1} \sum_{x_2} \ldots \sum_{x_{n-1}} \psi_{i,i+1}(x_i, x_{i+1}) \right) \left( \sum_{x_{n+1}} \ldots \sum_{x_N} f_L(x_1, x_2, \ldots, x_N) \right) \]

\[ = \left( \sum_{x_1} \sum_{x_2} \ldots \sum_{x_{n-1}} \psi_{i,i+1}(x_i, x_{i+1}) \right) \left( \sum_{x_{n+1}} \ldots \sum_{x_N} \prod_{i=1}^{N-1} \psi_{i,i+1}(x_i, x_{i+1}) \right) \]

\[ = \left( \sum_{x_1} \sum_{x_2} \ldots \sum_{x_{n-1}} \prod_{i=1}^{N-1} \psi_{i,i+1}(x_i, x_{i+1}) \right) \left( \sum_{x_{n+1}} \ldots \sum_{x_N} \prod_{i=1}^{N-1} \psi_{i,i+1}(x_i, x_{i+1}) \right) \]
\[ p(x) = \left( \sum_{x_{n-1}} \prod_{i=0}^{n-2} \psi_{n-2-i}(x_{n-2}, x_{n-1}) \right) \left( \sum_{x_{n-1}} \prod_{i=0}^{n-2} \psi_{n-2-i}(x_{n-2}, x_{n-1}) \right) \]

\[ \prod_{i=0}^{n-2} \psi_{n-2-i}(x_{n-2}, x_{n-1}) = \psi_{n-1,0}(x_{n-2}, x_{n-1}) \psi_{n-2,0}(x_{n-2}, x_{n-1}) \ldots \psi_{2,0}(x_2, x_1) \psi_{1,0}(x_1, x_2) \]

\[ \sum_{x_{n-1}} \prod_{i=0}^{n-2} \psi_{n-2-i}(x_{n-2}, x_{n-1}) = \left( \sum_{x_{n-1}} \prod_{i=0}^{n-2} \psi_{n-2-i}(x_{n-2}, x_{n-1}) \right) \left( \sum_{x_{n-1}} \prod_{i=0}^{n-2} \psi_{n-2-i}(x_{n-2}, x_{n-1}) \right) \]

**Warmup example**

\[ \sum_{x_{n-1}} \psi(x_{n-1}, x_n) \psi(x_n, x_2) = \sum_{x_{n-1}} \psi(x_{n-1}, x_n) \sum_{x_{n-2}} \psi(x_n, x_2) \]

(Recall the distributive law: \( ba + ca = a(b + c) \))

For K=2, the first component of the sum:

\[ \sum_{x_{n-1}} \psi(x_{n-1}, x_2) \psi(x_2, x_1) = \psi(x_1, x_2) + \psi(x_2, x_1) + \psi(x_1, x_2) + \psi(x_2, x_1) \]

\[ \sum_{x_{n-1}} \psi(x_{n-1}, x_2) \psi(x_2, x_1) = \psi(x_2, x_1) + \psi(x_1, x_2) + \psi(x_2, x_1) + \psi(x_1, x_2) \]

\[ \sum_{x_{n-1}} \psi(x_{n-1}, x_2) \psi(x_2, x_1) = \left( \sum_{x_{n-1}} \psi(x_{n-1}, x_2) \right) + \left( \sum_{x_{n-1}} \psi(x_{n-1}, x_2) \right) \]

\[ \sum_{x_{n-1}} \psi(x_{n-1}, x_2) \psi(x_2, x_1) = \left( \sum_{x_{n-1}} \psi(x_{n-1}, x_2) \right) + \left( \sum_{x_{n-1}} \psi(x_{n-1}, x_2) \right) \]

(Deriving the right factor is similar.)
Matrix interpretation (for two variables)

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

\[\psi_{i,i+1}(x_i, x_{i+1}) \mapsto Q_{i+1,i} \quad \text{(note transposition!)}\]

\[\sum_i \psi_{i,i+1}(x_i, x_{i+1}) \mapsto \text{sums columns to get a vector } V_{i+1}\]

\[\sum_{x_i} \psi_{i+1,i+2}(x_{i+1}, x_{i+2}) \left\{ \sum_{x_i} \psi_{i,i+1}(x_i, x_{i+1}) \right\} \mapsto Q_{i+2,i+1} \cdot V_i\]

Computational Complexity

Suppose each variable has \(K\) values.

What is the cost of evaluating the first factor?

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

The cost of the left factor is \(O(N \cdot K^2)\).

The other factor is similar.

We see that the overall cost is \(O(N \cdot K^2)\).

Much better than the naive computation where we had \(O(K^N)\)!

Message passing interpretation

Define \(\mu_a(x_n)\) as a message passed from node \(x_{n-1}\) to node \(x_n\).

Define \(\mu_b(x_n)\) as a message passed from node \(x_{n+1}\) to node \(x_n\).

Passing messages will correspond to the computation of taking input messages, and computing output messages.
Message passing interpretation

\[ \sum_{x_{n-1}} \sum_{x_n} \prod_{i=1}^{n-1} \psi_{i,i+1}(x_i, x_{i+1}) = \left[ \begin{array}{c}
\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \\
\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \\
\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \\
\vdots \\
\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \\
\end{array} \right] \]

Algorithm

Send a message from \( x_1 \) to \( x_n \).
Send a message from \( x_N \) to \( x_n \).
Element wise multiply messages.
Normalize by sum (Z).

\[ p(x_n) = \frac{1}{Z} \mu_a(x_n) \mu_b(x_n) \]

Computing all marginals

To compute all marginals, send a message from left to right, and right to left, storing the result.

Now compute any marginal as before.

This way, computing all marginals is only twice as expensive as computing one of them.

The normalization constant is easily computed using any convenient node.
What if a node is observed?

If a node is observed, then we do the obvious. Specifically, we clamp the values of variables to the particular case.

EG, if $x_i$ is observed, then we do not need to marginalize over it.

Factor Graphs

Suppose $p(x)$ factorizes as:

$$p(x) = \prod_i f(x_i)$$

where $x_i$ are sets of of variables within $x$.

Make a node for each $x_i$ as usual.

Now, make a different kind of node for $f()$ (e.g., squares).

Draw edges between the factor nodes and the variables in the variable set, $s$.

Note that the factorization formula means that we can convert both directed and undirected graphs to factor graphs.

Factor Graph Example

Suppose $p(x)$ factorizes as:

$$p(x) = \prod_i f(x_i) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_3, x_4)$$

The graph is:

Factor Graph Example (continued)

Suppose $p(x)$ factorizes as:

$$p(x) = \prod_i f(x_i) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_3, x_4) f_d(x_4)$$

This layout emphasizes that factor graphs are bipartite.

Note two factors for the clique for 1 and 2, suggesting that factor graphs can preserve extra structure compared to undirected graphs.
Factor Graph Example (2)

\[ p(x) = \frac{p(x_1) p(x_2) p(x_3 | x_1, x_2)}{f_a f_b f_c} \]

Factor Graph Summary

\[ p(x) = \prod f(x_i) \quad \text{where } x_i \text{ are sets of variables within } x. \]

Denote variables by circles

Denote each factor by a square

Draw links between squares and variables in the sets \( x_i \).

Factor graphs are bipartite

Factor graph for a distribution is not necessarily unique.

Factor graphs conveniently represent the extended message passing needed for inference on trees/polytrees.
**Trees/Polytrees**

Directed graph. Root node has no parents, others have exactly one parent.

Undirected graph. Only one path between any pair of nodes.

A directed graph with only one path per pair is a polytree.

**Factor Graphs and Trees**

Factor graphs for directed trees, undirected trees, and directed polytrees are trees.

(Recall definition for undirected trees---there is only one path between any two nodes).

This means that (variable) node, $x_n$, with $K$ branches divides a tree into $K$ subtrees whose factors do not share variables except $x_n$.

**Observations about factor graphs for trees**

Any node can be root

Any node with $N$ links splits the graph into $N$ subgraphs which do not share nodes.

Having passed messages from:
1) the leaves to a chosen root;
2) the chosen root to the leaves,
all messages that can be passed have been passed.

Further, the number of messages in 1 and 2 are the same.

**Observations about factor graphs**
Sum-product algorithm

Generalizes what we did with chains.

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

As with chains, consider the problem of computing the marginal of a selected node, \( x_n \).

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

Factor --> node messages

The node \( x \) with \( N \) neighbors divides the graph into \( N \) subgraphs.

Define \( F(x,X_i) \) as the product of all factors involving \( x \) and nodes in the subgraph, \( X_i \).

\[
p(x) = \prod_{i \in \text{set}(x)} F(x,X_i)
\]

(joint distribution)

\[F(x,X_i)\]

Example, \( N=3 \)

\( x \) connects subgraphs with node sets \( A, B, C \).

\[
p(x) = F(x,X_a)F(x,X_b)F(x,X_c)
\]

where \( F(x,X_a) = \prod_y f(X_y) \)

and where \( X_i \subseteq \{x\} \cup A \)

(similarly for B and C)

Factor --> node messages

\[
p(x) = \sum \prod_{i \in \text{set}(x)} F(x,X_i)
\]

(now marginalize)

\[= \prod_{i \in \text{set}(x)} \left( \sum_{X_i} F(x,X_i) \right)\]

(interchange sums and products)

(recall our fancy formula)

\[
(\sum a_i)(\sum b_j) = \sum \sum a_i b_j
\]

Note that each sum is simpler than what we started with because the variable sets are disjoint except for \( x \).
Factor $\rightarrow$ node messages

\[
p(x) = \sum_{s} \prod_{x_j \in \text{dom}(x)} F(x, X_s) = \prod_{x_j \in \text{dom}(x)} \left\{ \sum_{x_i} F(x, X_s) \right\}
\]

(define)

\[
\mu_{f_x \rightarrow x}(x) = \sum_{X_s} F(x, X_s)
\]

Computing the factor $\rightarrow$ node message

The node $\rightarrow$ factor message

(define)

\[
\mu_{x_m \rightarrow f_j}(x_m) = \sum_{X_m} \prod_{x_i \in \text{dom}(f_j)} f(x, x_1, \ldots, x_M) \mu_{x_m \rightarrow f_j}(x_m)
\]

(for a node $x_m$ we send its distribution with the other variables in the subgraph marginalized out).
Computing the node→factor message

This is just essentially recursion (like base case except that we exclude the node we are sending to)

\[ \mu_{x_m \rightarrow f_s}(x_m) = \prod_{l \in \mathbb{N}(x_m) \setminus f_s} \mu_{f_l \rightarrow x_m}(x_m) \]

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

Sum-product algorithm (1)

Basic computation --- compute marginal for a node, \( x_n \).

The node \( x \) with \( N \) neighbors divides the graph into \( N \) subgraphs.

Define \( F(x, X_s) \) as the product of all factors involving \( x \) and nodes in the subgraph, \( X_s \).

\[ p(x) = \prod_{i \in [N]} F(x, X_s) \]  

(joint distribution)

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 leave node is factor, then start with the factor.
The sum-product algorithm (3)

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 leave 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 (4)

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.

The sum-product algorithm (5)

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

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

\[
p(x_s) = f(x_s) \prod_{i \in \pi(f)} \mu_{x_i \rightarrow f_i}(x_i)
\]

The sum-product algorithm (6)

If the factor graph came from a directed graph, then the expression for \(p(x)\) is already normalized.

Otherwise (as was the case of the chain), we can determine the normalization constant from one of the marginals (relatively inexpensive because only one variable is involved at that point).
Sum-product algorithm example

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

First we pass messages from leaves to root.

Declare \( x_3 \) as root node.

What are the messages we need to send?

\[
\begin{align*}
\mu_{x_1 \rightarrow f_a}(x_1) &= 1 \\
\mu_{f_a \rightarrow x_2}(x_2) &= \sum_{x_1} f_a(x_1, x_2) \\
\mu_{x_4 \rightarrow f_c}(x_4) &= 1 \\
\mu_{f_c \rightarrow x_2}(x_2) &= \sum_{x_4} f_c(x_2, x_4)
\end{align*}
\]
Next we pass messages from root to leaves.

Candidate for the first and second ones?

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

Summary of messages from leaves to root

\[
\mu_{x_1 \rightarrow f_a} (x_1) = 1
\]
\[
\mu_{f_a \rightarrow x_2} (x_2) = \sum_{x_1} f_a (x_1, x_2)
\]
\[
\mu_{x_1 \rightarrow f_c} (x_1) = 1
\]
\[
\mu_{f_c \rightarrow x_2} (x_2) = \sum_{x_4} f_c (x_2, x_4)
\]
\[
\mu_{x_2 \rightarrow f_b} (x_2) = \mu_{f_a \rightarrow x_2} (x_2) \mu_{f_c \rightarrow x_2} (x_2)
\]
\[
\mu_{f_b \rightarrow x_3} (x_3) = \sum_{x_2} f_b (x_2, x_3) \mu_{x_2 \rightarrow f_b} (x_2)
\]

Candidate for third and fourth?

\[
\mu_{x_3 \rightarrow f_b} (x_3) = 1
\]
\[
\mu_{f_b \rightarrow x_2} (x_2) = \sum_{x_3} f_b (x_2, x_3)
\]
\[
\mu_{x_2 \rightarrow f_a}(x_2) = \mu_{f_b \rightarrow x_2}(x_2) \mu_{f_a \rightarrow x_2}(x_2)
\]
\[
\mu_{f_b \rightarrow x_1}(x_1) = \sum_{x_2} f_a(x_1, x_2) \mu_{x_2 \rightarrow f_a}(x_2)
\]

(similar to previous one)

\[
\mu_{x_3 \rightarrow f_b}(x_3) = 1
\]
\[
\mu_{f_b \rightarrow x_2}(x_2) = \sum_{x_3} f_b(x_2, x_3)
\]
\[
\mu_{x_2 \rightarrow f_a}(x_2) = \mu_{f_b \rightarrow x_2}(x_2) \mu_{f_a \rightarrow x_2}(x_2)
\]
\[
\mu_{f_a \rightarrow x_1}(x_1) = \sum_{x_2} f_a(x_1, x_2) \mu_{x_2 \rightarrow f_a}(x_2)
\]
\[
\mu_{x_2 \rightarrow f_c}(x_2) = \mu_{f_b \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2)
\]
\[
\mu_{f_c \rightarrow x_4}(x_4) = \sum_{x_2} f_c(x_2, x_4) \mu_{x_2 \rightarrow f_c}(x_2)
\]

An illustrative check
\[
\tilde{p}(x) = \mu_{f_b \rightarrow x}(x_2) \mu_{f_a \rightarrow x}(x_2) \mu_{f_c \rightarrow x}(x_3)
\]

\[
= \left( \sum_{x_2} f_a(x_1, x_2) \mu_{x_2 \rightarrow f_a}(x_1) \right) \left( \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \rightarrow f_b}(x_1) \right) \left( \sum_{x_2} f_c(x_2, x_4) \mu_{x_2 \rightarrow f_c}(x_1) \right)
\]

\[
= \left( \sum_{x_2} f_a(x_1, x_2) \right) \left( \sum_{x_2} f_b(x_2, x_3) \right) \left( \sum_{x_2} f_c(x_2, x_4) \right)
\]

\[
= \sum_{x_2} \sum_{x_2} \tilde{p}(x_2, x_4)
\]

\[
= \sum_{x_2} \sum_{x_2} \tilde{p}(x)
\]
Sum-product algorithm example

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

Declare $x_3$ as root node.

$\mu_{x_3 \rightarrow f_c}(x_3) = 1$
$\mu_{f_c \rightarrow x_2}(x_2) = \sum_{x_3} f_b(x_2, x_3)$
$\mu_{x_2 \rightarrow f_a}(x_2) = \mu_{f_c \rightarrow x_2}(x_2)\mu_{f_b \rightarrow x_2}(x_2)$
$\mu_{f_b \rightarrow x_1}(x_1) = \sum_{x_2} f_a(x_1, x_2)\mu_{x_3 \rightarrow f_a}(x_2)$
$\mu_{x_2 \rightarrow f_c}(x_2) = \mu_{f_b \rightarrow x_2}(x_2)\mu_{f_a \rightarrow x_2}(x_2)$
$\mu_{f_a \rightarrow x_4}(x_4) = \sum_{x_2} f_c(x_2, x_4)\mu_{x_2 \rightarrow f_c}(x_2)$

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 a factor

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

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

(can be normalized to get $p(h|\hat{v})$)
Max-sum algorithm

Method to compute.

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

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

Recall speeding up marginalization

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

$$\sum_{x_{n-1}} \cdots \sum_{x_1} \prod_{i=1}^{n-1} \psi_{i,i+1}(x_{i},x_{i+1}) = \left[ \sum_{x_1} \psi_{1,2}(x_1,x_2) \right] \left[ \sum_{x_2} \psi_{2,3}(x_2,x_3) \right]$$

and

$$\sum_{x_{n-1}} \cdots \sum_{x_1} \prod_{i=1}^{n-1} \psi_{i,i+1}(x_{i},x_{i+1}) = \left[ \sum_{x_1} \psi_{1,2}(x_1,x_2) \right] \left[ \sum_{x_2} \psi_{2,3}(x_2,x_3) \right]$$

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

Recall inference on chains

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

Naive computation of \(\arg \max_x p(\mathbf{x})\)

would evaluate the above for each value of \(\mathbf{x}\), and take the max.

Too expensive!!

Helpful facts

First note that.

$$\max_x p(\mathbf{x}) = \max_{x_1} \max_{x_2} \cdots \max_{x_n} p(\mathbf{x})$$

Second note that.

$$\max(ab,ac) = a \max(b,c) \quad (\text{for } a \geq 0)$$
Max on a chain

\[
\max_x p(x) = \frac{1}{Z} \max_{x_1} \left[ \max_{x_2} \left[ \ldots \max_{x_{N-1}} \left[ \prod_{i=1}^{N} (x_i) \right] \right] \right]
\]

The second line shows that we can do this using message passing.
(Compute max distribution, then multiply, rinse and repeat.)

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 should be called the “max-product” algorithm.
Computing long products loses precision, so we switch to log().

Max-sum algorithm

Note that
\[
\ln \left( \max_x (p(x)) \right) = \max_x \left( \ln (p(x)) \right)
\]
And we have
\[
\max (a + b, a + c) = a + \max (b, c)
\]

Max-sum algorithm

Working now in analogy with the sum-product algorithm
\[
\mu_{f \rightarrow x} (x) = \max_{x_1, x_2, \ldots, x_M} \left[ \ln \left( f(x_1, x_2, \ldots, x_M) \right) + \sum_{m \in n(x)} \mu_{x_m \rightarrow f} (x_m) \right]
\]
and
\[
\mu_{x \rightarrow f} (x) = \sum_{x \in n(x)} \mu_{f \rightarrow x} (x)
\]
Max-sum algorithm

Working now in analogy with the sum-product algorithm

For initialization at leaf nodes

\[ \mu_{f \rightarrow x}(x) = 0 \]

and

\[ \mu_{x \rightarrow f}(x) = \ln(f(x)) \]

Max-sum algorithm

Now we need to find an \( x \) where \( p \) reaches the max.

This does not have a direct analogy in the sum-product algorithm.

**Why we do not know \( x \) yet:**

The factor-to-node messages takes a distribution for the maxima over the downstream variables, and multiplies it by the factor (sum using logs), and reports a new distribution.

We do not know which value in the new distribution will be part of the maximum.

Max-sum algorithm

To compute the max using the chosen root node,

\[ p^{\text{max}} = \max_x \left[ \sum_{\mu_{x \rightarrow y}(x)} \mu_{x \rightarrow y}(x) \right] \]

Can passing messages backwards find the arg max?

At the root node, which is a product (sum in logs), the contributions of the maximum are factored, so messages backwards can find the configuration.

But at the factor nodes, if there are two configurations that give the same maximum, we might get pieces of each of them.

This is because the variable for the node collecting the factor-to-node messages is shared.

(We made an arbitrary choice, but did not write it down!)
Max-sum algorithm

Adjustment to forward message passing so backtracking works:

The factor-to-node operations store the dependencies for the various choices of $x_i$.

Then, once the node-to-factor backtracking expresses a choice, a consistent set of values for $x_i$ for the max can be found.

Max-sum algorithm (back-tracking)

In more detail, when we compute

$$
\mu_{f \rightarrow x}(x) = \max_{x_1, x_2, \ldots, x_M} \left[ \ln \left( f(x, x_1, x_2, \ldots, x_M) + \sum_{\text{mum}(f) \mid x} \mu_{x_m \rightarrow f}(x_m) \right) \right]
$$

store

$$
\phi(x) = \arg \max_{x_1, x_2, \ldots, x_M} \left[ \ln \left( f(x, x_1, x_2, \ldots, x_M) + \sum_{\text{mum}(f) \mid x} \mu_{x_m \rightarrow f}(x_m) \right) \right]
$$

(This records the downstream choices for any upstream choice of $x$)

Then, once we know the overall max, we can recover a set of $x_i$ that leads to it by backtracking.