Exact Inference

Chris Williams
(based on slides by Michael U. Gutmann)

Probabilistic Modelling and Reasoning (INFR11134)
School of Informatics, The University of Edinburgh

Spring Semester 2024
Recap

\[ p(x|y_o) = \frac{\sum_z p(x,y_o,z)}{\sum_{x,z} p(x,y_o,z)} \]

Assume that \(x, y, z\) each are \(d = 500\) dimensional, and that each element of the vectors can take \(K = 10\) values.

**Issue 1:** To specify \(p(x, y, z)\), we need to specify \(K^{3d} - 1 = 10^{1500} - 1\) non-negative numbers, which is impossible.

**Topic 1: Representation** What reasonably weak assumptions can we make to efficiently represent \(p(x, y, z)\)?

- Directed and undirected graphical models, factor graphs
- Factorisation and independencies
Recap

\[ p(x|y_o) = \frac{\sum_z p(x,y_o,z)}{\sum_{x,z} p(x,y_o,z)} \]

- **Issue 2**: The sum in the numerator goes over the order of \(K^d = 10^{500}\) non-negative numbers and the sum in the denominator over the order of \(K^{2d} = 10^{1000}\), which is impossible to compute.

- **Topic 2: Exact inference** Can we further exploit the assumptions on \(p(x,y,z)\) to efficiently compute the posterior probability or derived quantities?

- **Note**: we do not want to introduce new assumptions but exploit those that we made to deal with issue 1.

- **Quantities of interest**:
  - \(p(x|y_o)\) (marginal inference)
  - \(\arg\max_x p(x|y_o)\) (inference of most probable states)
  - \(\mathbb{E}[g(x) \mid y_o]\) for some function \(g\) (posterior expectations)
Assumptions

Unless otherwise mentioned, we here assume discrete valued random variables whose joint pmf factorises as

\[ p(x_1, \ldots, x_d) \propto \prod_{i=1}^{m} \phi_i(X_i), \]

with \( X_i \subseteq \{x_1, \ldots, x_d\} \) and \( x_i \in \{1, \ldots, K\} \).

Note:

- Includes case where (some of) the \( \phi_i \) are conditionals
- The \( x_i \) could be categorical taking on maximally \( K \) different values.
1. Marginal inference by variable elimination

2. Marginal inference for factor trees (sum-product algorithm)

3. Inference of most probable states for factor trees
1. Marginal inference by variable elimination
   - Exploiting the factorisation by using the distributive law 
     \( ab + ac = a(b + c) \) and by caching computations
   - Variable elimination for general factor graphs
   - The principles of variable elimination also apply to continuous random variables

2. Marginal inference for factor trees (sum-product algorithm)

3. Inference of most probable states for factor trees
Basic ideas of variable elimination

1. Use the distributive law $ab + ac = a(b + c)$ to exploit the factorisation ($\sum \prod \rightarrow \prod \sum$):
   reduces the overall dimensionality of the domain of the factors in the sum and thereby the computational cost.

2. Recycle/cache results
Consider discrete-valued random variables $x_1, x_2, x_3 \in \{1, \ldots, K\}$

Assume pmf factorises $p(x_1, x_2, x_3) \propto \phi_1(x_1)\phi_2(x_2)\phi_3(x_3)$

Task: compute $p(x_1 = k)$ for $k \in \{1, \ldots, K\}$

We can use the sum-rule

$$p(x_1 = k) = \sum_{x_2, x_3} p(x_1 = k, x_2, x_3)$$

Sum over $K^2$ terms for each $k$ (value of $x_1$).

Pre-computing $p(x_1, x_2, x_3)$ for all $K^3$ configurations and then computing the sum is neither necessary nor a good idea

Exploit factorisation when computing $p(x_1 = k)$. 
Example: full factorisation

\begin{align*}
\text{(sum rule)} & \quad p(x_1 = k) = \sum_{x_2, x_3} p(x_1 = k, x_2, x_3) \quad (1) \\
\text{(factorisation)} & \quad \propto \sum_{x_2} \sum_{x_3} \phi_1(k) \phi_2(x_2) \phi_3(x_3) \quad (2) \\
\text{(distr. law)} & \quad \propto \phi_1(k) \sum_{x_2} \sum_{x_3} \phi_2(x_2) \phi_3(x_3) \quad (3) \\
\text{(distr. law)} & \quad \propto \phi_1(k) \left[ \sum_{x_2} \phi_2(x_2) \right] \left[ \sum_{x_3} \phi_3(x_3) \right] \quad (4)
\end{align*}

Distributive law changes $\sum \prod$ in (2) to $\prod \sum$ in (4).
Example: full factorisation

\[ p(x_1 = k) \propto \phi_1(k) \left[ \sum_{x_2} \phi_2(x_2) \right] \left[ \sum_{x_3} \phi_3(x_3) \right] \]  

(5)

What’s the point?

▶ Because of the factorisation (independencies) we do not need to evaluate and store the values of \( p(x_1, x_2, x_3) \) for all \( K^3 \) configurations of the random variables.

▶ 2 sums over \( K \) numbers vs. 1 sum over \( K^2 \) numbers

▶ Recycling/caching of already computed quantities: we only need to compute

\[ \left[ \sum_{x_2} \phi_2(x_2) \right] \left[ \sum_{x_3} \phi_3(x_3) \right] \]

once; the value can be re-used when computing \( p(x_1 = k) \) for different \( k \).
Example: general factor graph

Example:

\[ p(x_1, \ldots, x_6) \propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\phi_D(x_3, x_6) \]

Task: Compute \( p(x_1, x_3) \)

Note the structural changes in the graph during variable elimination
Task: Compute $p(x_1, x_3)$

First eliminate $x_6$

$$p(x_1, \ldots, x_5) = \sum_{x_6} p(x_1, \ldots, x_6)$$

(factorisation) $\propto \sum_{x_6} \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\phi_D(x_3, x_6)$

(distr. law) $\propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\sum_{x_6} \phi_D(x_3, x_6)$

$$\propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\tilde{\phi}_6(x_3)$$
Example: general factor graph (cont)

Task: Compute $p(x_1, x_3)$

Eliminate $x_5$

$$p(x_1, \ldots, x_4) \propto \sum_{x_5} \phi_A(x_1, x_2, x_4) \phi_B(x_2, x_3, x_4) \phi_C(x_3, x_5) \tilde{\phi}_6(x_3)$$

$$\propto \phi_A(x_1, x_2, x_4) \phi_B(x_2, x_3, x_4) \tilde{\phi}_6(x_3) \sum_{x_5} \phi_C(x_3, x_5)$$

$$\propto \phi_A(x_1, x_2, x_4) \phi_B(x_2, x_3, x_4) \tilde{\phi}_6(x_3) \tilde{\phi}_5(x_3)$$
Define $\tilde{\phi}_{56}(x_3) = \tilde{\phi}_6(x_3)\tilde{\phi}_5(x_3)$

$$p(x_1, \ldots, x_4) \propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\tilde{\phi}_6(x_3)\tilde{\phi}_5(x_3)$$

$$\propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\tilde{\phi}_{56}(x_3)$$
Example: general factor graph (cont)

Task: Compute \( p(x_1, x_3) \)

Eliminate \( x_2 \)

\[
p(x_1, x_3, x_4) \propto \sum_{x_2} \phi_A(x_1, x_2, x_4) \phi_B(x_2, x_3, x_4) \tilde{\phi}_{56}(x_3)
\]

\[
\propto \tilde{\phi}_{56}(x_3) \sum_{x_2} \phi_A(x_1, x_2, x_4) \phi_B(x_2, x_3, x_4)
\]

\[
\propto \tilde{\phi}_{56}(x_3) \tilde{\phi}_2(x_1, x_3, x_4)
\]

Other justification for the cost: \( \phi_A(x_1, x_2, x_4) \phi_B(x_2, x_3, x_4) \) equals a compound factor \( \tilde{\phi}^*(x_1, x_2, x_3, x_4) \) that requires \( K^4 \) space when represented as a table. Summing out \( x_2 \) for all combinations of \((x_1, x_3, x_4)\) touches each table-entry once \( \Rightarrow O(K^4) \) cost.
Example: general factor graph (cont)

Task: Compute $p(x_1, x_3)$

Eliminate $x_4$

$$p(x_1, x_3) \propto \sum_{x_4} \tilde{\phi}_{56}(x_3) \tilde{\phi}_2(x_1, x_3, x_4)$$

$$\propto \tilde{\phi}_{56}(x_3) \sum_{x_4} \tilde{\phi}_2(x_1, x_3, x_4)$$

$$\propto \tilde{\phi}_{56}(x_3) \tilde{\phi}_{24}(x_1, x_3)$$

Normalisation to obtain $p(x_1 = k, x_3 = k')$ for any $k, k'$:

$$p(x_1 = k, x_3 = k') = \frac{\tilde{\phi}_{56}(x_3 = k') \tilde{\phi}_{24}(x_1 = k, x_3 = k')}{\sum_{x_1, x_3} \tilde{\phi}_{56}(x_3) \tilde{\phi}_{24}(x_1, x_3)}$$
Remarks

- Compared to precomputing $K^6$ numbers and then marginalising out variables, using the factorisation reduces the cost to $O(K^4)$.

- Caching: Most of the intermediate quantities can be re-used when computing $p(x_1 = k, x_3 = k')$ for different $k, k'$

- Structural changes in the graph during variable elimination:
  - Eliminated leaf-variable and factor node → factor node
  - Factor nodes that depend on the same variables → single factor node
  - Factor nodes between neighbours of the eliminated variable → single factor node connecting all neighbours
Without loss of generality: Given \( p(x_1, \ldots, x_d) \propto \prod_i^m \phi_i(x_i) \)
compute the marginal \( p(x_{\text{target}}) \) for some \( x_{\text{target}} \subseteq \{x_1, \ldots, x_d\} \).

- Assume that at iteration \( k \), you have the pmf over \( d^k = d - k \) variables \( X^k = (x_{i_1}, \ldots, x_{i_d}) \) that factorises as

\[
p(X^k) \propto \prod_{i=1}^{m^k} \phi_i^k(x_i^k)
\]

- Decide which variable to eliminate. Call it \( x^* \).
  \((x^* \in X^k, x^* \notin X_{\text{target}})\)

- Let \( X^{k+1} \) be equal to \( X^k \) with \( x^* \) removed. We have

\[
\text{(sum rule)} \quad p(X^{k+1}) = \sum_{x^*} p(X^k) \quad (6)
\]

\[
\text{(factorisation)} \quad \propto \sum_{x^*} \prod_{i=1}^{m^k} \phi_i^k(x_i^k) \quad (7)
\]
Variable (bucket) elimination (cont.)

\[
p(X^{k+1}) \propto \sum_{x^*} \prod_{i: x^* \notin \mathcal{X}^k_i} \phi^k_i(\mathcal{X}^k_i) \prod_{i: x^* \in \mathcal{X}^k_i} \phi^k_i(\mathcal{X}^k_i) \tag{8}
\]

(distr. law) \[\propto \prod_{i: x^* \notin \mathcal{X}^k_i} \phi^k_i(\mathcal{X}^k_i) \sum_{x^*} \prod_{i: x^* \in \mathcal{X}^k_i} \phi^k_i(\mathcal{X}^k_i) \tag{9}\]

\[
\propto \left[ \prod_{i: x^* \notin \mathcal{X}^k_i} \phi^k_i(\mathcal{X}^k_i) \right] \sum_{x^*} \phi^k_*(\mathcal{X}_*^k) \tag{10}\]

\[\mathcal{X}_*^k\] is the union of all \(\mathcal{X}_i^k\) that contain \(x^*\), and \(\tilde{\mathcal{X}}_*^k\) is \(\mathcal{X}_*^k\) with \(x^*\) removed,

\[
\mathcal{X}_*^k = \bigcup_{i: x^* \in \mathcal{X}^k_i} \mathcal{X}^k_i \quad \tilde{\mathcal{X}}_*^k = \mathcal{X}_*^k \setminus x^* \tag{11}\]
By re-labelling the factors and variables, we obtain

\[ p(X^{k+1}) \propto \prod_{i : x^* \notin X_i^k} \phi_i^k(X_i^k) \tilde{\phi}_*^k(\tilde{X}_*) \]  
\[ \propto m^{k+1} \prod_{i=1}^{m^{k+1}} \phi_i^{k+1}(X_i^{k+1}), \]  
(12)
(13)

which has the same form as \( p(X^k) \).

Set \( k = k + 1 \) and decide which variable \( x^* \) to eliminate next.

To compute \( p(X_{\text{target}}) \) stop when \( X^k = X_{\text{target}} \), followed by
normalisation.
How to choose the elimination variable $x^*$?

- When we marginalise over $x^*$ in iteration $k$, we generate the temporary compound factor $\phi^k_*$ that depends on

$$\mathcal{X}^k_* = \bigcup_{i: x^* \in \mathcal{X}^k_i} \mathcal{X}^k_i$$ (14)

Contains $x^*$ and the variables with which $x^*$ shares a factor node in the factor graph ("neighbours").

- Ex.: $p(x_1, \ldots, x_6) \propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\phi_D(x_3, x_6)$

If we eliminated $x^* = x_3$: $\mathcal{X}^*_* = \{x_2, x_3, x_4, x_5, x_6\}$
How to choose the elimination variable $x^*$?

- When we marginalise over $x^*$ in iteration $k$, we generate the temporary compound factor $\phi^k_*$ that depends on

\[
\mathcal{X}^k_* = \bigcup_{i:x^* \in \mathcal{X}^k_i} \mathcal{X}^k_i
\]  

(15)

Contains $x^*$ and the variables with which $x^*$ shares a factor node in the factor graph (“neighbours”).

- Eliminating $x^*$ costs $K^M_k$ where $M_k$ is the number of variables in $\mathcal{X}^k_*$. 

- Optimal choice of elimination order is difficult since the size of the factors can change when we eliminate variables (for details, see e.g. Koller, Section 9.4, not examinable).

- Heuristic: in each iteration, choose $x^*$ in a greedy way so that $\mathcal{X}^k_*$ is small, i.e. the variable with the least number of neighbours in the factor graph (e.g. $x_5$ or $x_6$ in the example).
Computing conditionals

- The same approach can be used to compute conditionals.
- Example: Given

\[ p(x_1, \ldots, x_6) \propto \phi_A(x_1, x_2, x_4)\phi_B(x_2, x_3, x_4)\phi_C(x_3, x_5)\phi_D(x_3, x_6) \]

assume you want to compute \( p(x_1 | x_3 = \alpha) \)
- We can write

\[ p(x_1, x_2, x_4, x_5, x_6 | x_3 = \alpha) \propto p(x_1, x_2, x_3 = \alpha, x_4, x_5, x_6) \]
\[ \propto \phi_A(x_1, x_2, x_4)\phi_B^{\alpha}(x_2, x_4)\phi_C^{\alpha}(x_5)\phi_D^{\alpha}(x_6) \]

and consider \( p(x_1, x_2, x_4, x_5, x_6 | x_3 = \alpha) \) to be a pdf/pmf \( \tilde{p}(x_1, x_2, x_4, x_5, x_6) \) defined up to the proportionality factor.
- We can compute \( p(x_1 | x_3 = \alpha) = \tilde{p}(x_1) \) by applying variable elimination to \( \tilde{p}(x_1, x_2, x_4, x_5, x_6) \).
Conceptually, all stays the same but we replace sums with integrals
  ▶ Simplifications due to distributive law remain valid
  ▶ Caching of results remains valid

In special cases, integral can be computed in closed form (e.g. Gaussian family)

If not: need for approximations (see later)

Approximations are also needed for discrete random variables when $K$ is large.
1. Marginal inference by variable elimination
   - Exploiting the factorisation by using the distributive law $ab + ac = a(b + c)$ and by caching computations
   - Variable elimination for general factor graphs
   - The principles of variable elimination also apply to continuous random variables

2. Marginal inference for factor trees (sum-product algorithm)

3. Inference of most probable states for factor trees
1. Marginal inference by variable elimination

2. Marginal inference for factor trees (sum-product algorithm)
   - Factor trees
   - Message passing for factor trees (sum-product algorithm)
   - The rules for sum-product message passing
   - Illustrating message passing on an example factor tree

3. Inference of most probable states for factor trees
Factor trees

- We next consider the class of models (pmfs/pdfs) for which the factor graph is a tree.
- Tree: graph where there is only one path connecting any two nodes (no loops!)
- Chain is an example of a factor tree. (see later: inference for HMMs)
- Useful property: the factor tree obtained after summing out a leaf variable is still a factor tree.
Motivating message passing on trees

Let

\[ p(x_1, \ldots, x_d) = \frac{1}{Z} \prod_{j=1}^{m} \phi_j(x_j) \]

So

\[ p(x) = \frac{1}{Z} \sum_{\mathcal{X}\setminus x} \prod_{j=1}^{m} \phi_j(x_j) \]

As the graph is a tree, \( \mathcal{X}\setminus x \) can be broken up into disjoint subsets \( \{\mathcal{X}_i\} \), with \( \cup_i \mathcal{X}_i = \mathcal{X}\setminus x \).

The product of potentials can also be factored as

\[ \prod_{j=1}^{m} \phi_j(x_j) = \prod_{i \in ne(x)} F_i(x, \mathcal{X}_i) \]

Each \( F_i(x, \mathcal{X}_i) \) will contain one or more of the \( m \) potentials
The sums over the \( \{\mathcal{X}_i\} \) can be pushed through the product to give

\[
p(x) = \frac{1}{Z} \prod_{i \in \text{ne}(x)} \left[ \sum_{\mathcal{X}_i} F_i(x, \mathcal{X}_i) \right]
\]

\[
def \frac{1}{Z} \prod_{i \in \text{ne}(x)} \mu_{\phi_i \rightarrow x}(x)
\]

Fragment of the factor graph

\[
F_i(x, \mathcal{X}_i) = \phi_i(x, x_1, \ldots, x_j) G_1(x_1, \mathcal{X}_{i1}) \ldots G_j(x_j, \mathcal{X}_{ij})
\]
Summing out $\mathcal{X}_i$ over $F_i(x, \mathcal{X}_i)$

$$\sum_{\mathcal{X}_i} F_i(x, \mathcal{X}_i) = \sum_{\mathcal{X}_i} \phi_i(x, x_1, \ldots, x_j) G_1(x_1, \mathcal{X}_{i1}) \cdots G_j(x_j, \mathcal{X}_{ij})$$

$$\text{def} = \sum_{\mathcal{X}_i} \phi_i(x, x_1, \ldots, x_j) \prod_{k=1}^{j} \mu_{x_k \rightarrow \phi_i(x_k)}$$

So

$$\mu_{\phi_i \rightarrow x}(x) = \sum_{\mathcal{X}_i} \phi_i(x, x_1, \ldots, x_j) \prod_{k=1}^{j} \mu_{x_k \rightarrow \phi_i(x_k)}$$
Message passing for factor trees (sum-product algorithm)

- Computation can be organized to pass messages from the leaves of the tree to a root node
- Root can be chosen as \( x \), the variable for which we wish to compute the marginal
- Messages are passed:
  - From a factor to a variable \( \mu_{\phi \rightarrow x} (x) \)
  - From a variable to a factor \( \mu_{x \rightarrow \phi} (x) \)
- A factor or variable can update its message once it has received all incoming messages
Note: The rules come from the fact that messages correspond to effective factors obtained after marginalisation.

- From a leaf variable node $x$ to a factor node $\phi$, the message $\mu_{x\rightarrow\phi}(x) = 1$.
- From a leaf factor node $\phi$ to a variable node $x$, the message $\mu_{\phi\rightarrow x}(x) = \phi(x)$. 
Rules of message passing: factor to variable messages

Note: The rules come from the fact that messages correspond to effective factors obtained after marginalisation.

Let \( x_1, \ldots, x_j \) be the neighbours of factor node \( \phi \), without variable \( x \).

\[
\mu_{\phi \to x}(x) = \sum_{x_1, \ldots, x_j} \phi(x_1, \ldots, x_j, x) \prod_{i=1}^{j} \mu_{x_i \to \phi}(x_i)
\]

Rule corresponds to eliminating variables \( x_1, \ldots, x_j \)

This is the sum-product operation
Note: The rules come from the fact that messages correspond to effective factors obtained after marginalisation.

Let $\phi_1, \ldots, \phi_j$ be the neighbours of variable node $x$, without factor $\phi$.

$$
\mu_{x \rightarrow \phi}(x) = \prod_{i=1}^{j} \mu_{\phi_i \rightarrow x}(x)
$$

Rule corresponds to simplifying the factorisation by multiplying effective factors defined on the same domain.
Note: The rules come from the fact that messages correspond to effective factors obtained after marginalisation.

Let $\phi_1, \ldots, \phi_j$ be all neighbours of variable node $x$.

$$p(x) = \frac{1}{Z} \prod_{i=1}^{j} \mu_{\phi_i \rightarrow x}(x) \quad Z = \sum_{x} \prod_{i} \mu_{\phi_i \rightarrow x}(x)$$

Note: The normalising constant $Z$ can be computed for any of the marginals. Same as the normaliser for $p(x_1, \ldots, x_d) \propto \prod_{i} \phi_i(x_i)$. 
Illustrating message passing on an example factor tree

Task: Compute $p(x_1)$ for

$$p(x_1, \ldots, x_5) \propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4)\phi_E(x_3, x_5)\phi_F(x_5)$$
Sum out leaf-variable $x_5$

Task: Compute $p(x_1)$

$$p(x_1, \ldots, x_4) = \sum_{x_5} p(x_1, \ldots, x_5)$$

$$\propto \sum_{x_5} \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4)\phi_E(x_3, x_5)\phi_F(x_5)$$

$$\propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4) \sum_{x_5} \phi_E(x_3, x_5)\phi_F(x_5)$$

$$\propto \phi_A(x_1)\phi_B(x_2)\phi_C(x_1, x_2, x_3)\phi_D(x_3, x_4)\tilde{\phi}_5(x_3)$$
Visualising the computation

Graph with transformed factors:

Graph with “messages”:
Visualising the computation

Graph with “messages”:

Messages:

\[
\mu_{\phi_F \rightarrow x_5}(x_5) = \phi_F(x_5) \quad \text{(initialisation)}
\]
\[
\mu_{x_5 \rightarrow \phi_E}(x_5) = \mu_{\phi_F \rightarrow x_5}(x_5) = \phi_F(x_5)
\]
\[
\mu_{\phi_E \rightarrow x_3}(x_3) = \sum_{x_5} \phi_E(x_3, x_5) \mu_{x_5 \rightarrow \phi_E}(x_5) = \sum_{x_5} \phi_E(x_3, x_5) \phi_F(x_5) = \tilde{\phi}_5(x_3)
\]
Sum out leaf-variable $x_4$

Task: Compute $p(x_1)$

\[
p(x_1, \ldots, x_3) = \sum_{x_4} p(x_1, \ldots, x_4)
\]

\[
\propto \sum_{x_4} \phi_A(x_1) \phi_B(x_2) \phi_C(x_1, x_2, x_3) \phi_D(x_3, x_4) \tilde{\phi}_5(x_3)
\]

\[
\propto \phi_A(x_1) \phi_B(x_2) \phi_C(x_1, x_2, x_3) \tilde{\phi}_5(x_3) \sum_{x_4} \phi_D(x_3, x_4)
\]

\[
\propto \phi_A(x_1) \phi_B(x_2) \phi_C(x_1, x_2, x_3) \tilde{\phi}_5(x_3) \tilde{\phi}_4(x_3)
\]
Visualising the computation

Graph with transformed factors:

Graph with messages:
Visualising the computation

Graph with messages:

Messages:

\[
\mu_{x_4 \rightarrow \phi_D(x_4)} = 1 \quad \text{(initialisation)}
\]

\[
\mu_{\phi_D \rightarrow x_3} = \sum_{x_4} \phi_D(x_3, x_4) \mu_{x_4 \rightarrow \phi_D(x_4)} = \sum_{x_4} \phi_D(x_3, x_4) = \tilde{\phi}_4(x_3)
\]
Sum out both $x_2$ and $x_3$

\[
p(x_1, \ldots, x_3) \propto \phi_A(x_1) \phi_B(x_2) \phi_C(x_1, x_2, x_3) \tilde{\phi}_5(x_3) \tilde{\phi}_4(x_3)
\]

\[
p(x_1) \propto \phi_A(x_1) \sum_{x_2, x_3} \phi_C(x_1, x_2, x_3) \phi_B(x_2) \tilde{\phi}_4(x_3) \tilde{\phi}_5(x_3)
\]

\[
\propto \phi_A(x_1) \tilde{\phi}_{5432}(x_1)
\]

Hence

\[
p(x_1) = \frac{\phi_A(x_1) \tilde{\phi}_{5432}(x_1)}{\sum_{x'_1} \phi_A(x'_1) \tilde{\phi}_{5432}(x'_1)}
\]
Visualising the computation

Graph with transformed factors:

Graph with messages:
Visualising the computation

Graph with messages:

Messages:

\[
\mu_{\phi_B \rightarrow x_2}(x_2) = \phi_B(x_2) \quad \text{(initialisation)}
\]

\[
\mu_{x_2 \rightarrow \phi_C}(x_2) = \mu_{\phi_B \rightarrow x_2}(x_2) = \phi_B(x_2)
\]

\[
\mu_{x_3 \rightarrow \phi_C}(x_3) = \mu_{\phi_D \rightarrow x_3}(x_3)\mu_{\phi_E \rightarrow x_3}(x_3) = \tilde{\phi}_4(x_3)\tilde{\phi}_5(x_3)
\]

\[
\mu_{\phi_C \rightarrow x_1}(x_1) = \tilde{\phi}_{5432}(x_1) = \sum_{x_2, x_3} \phi_C(x_1, x_2, x_3)\mu_{x_2 \rightarrow \phi_C}(x_2)\mu_{x_3 \rightarrow \phi_C}(x_3)
\]

\[
= \sum_{x_2, x_3} \phi_C(x_1, x_2, x_3)\phi_B(x_2)\tilde{\phi}_4(x_3)\tilde{\phi}_5(x_3)
\]

\[
\mu_{\phi_A \rightarrow x_1}(x_1) = \phi_A(x_1) \quad \text{(initialisation)}
\]
Single marginal from messages

We have seen that

\[ p(x_1) \propto \phi_A(x_1) \tilde{\phi}_{5432}(x_1) \]
\[ \propto \mu_{\phi_{A\rightarrow x_1}}(x_1) \mu_{\phi_{C\rightarrow x_1}}(x_1) \]

Marginal is proportional to the product of the incoming messages.
Single marginal from messages

Cost (due to properties of variable elimination):

- Linear in number of variables $d$, exponential in maximal number of variables attached to a factor node.
  
  (cost known upfront since no new factors are created unlike in the general case considered before)

- Recycling: most messages do not depend on $x_1$ and can be re-used for computing $p(x_1)$ for any value of $x_1$ (as well as for computing the marginal distribution of other variables, see next slides)
Further marginals from messages

- We have seen that

\[ p(x_1) \propto \phi_A(x_1) \tilde{\phi}_{5432}(x_1) \]

\[ \propto \mu_{\phi_A \rightarrow x_1}(x_1) \mu_{\phi_C \rightarrow x_1}(x_1) \]

- **Remember**: Messages are effective factors

- This correspondence allows us to write down the marginal for other variables too. The incoming messages are all we need.
Further marginals from messages

- Example: For $p(x_2)$ we need $\mu_{\phi_B \to x_2}$ and $\mu_{\phi_C \to x_2}$
- $\mu_{\phi_B \to x_2}$ is known but $\mu_{\phi_C \to x_2}$ needs to be computed
- $\mu_{\phi_C \to x_2}$ is the effective factor for $x_2$ if all variables of the subtrees attached to $\phi_c$ are eliminated.
- Can be computed from previously computed factors:

$$\mu_{\phi_A \to x_1} \quad \text{and} \quad \mu_{x_3 \to \phi_C}$$
Further marginals from messages

By definition of the messages, and their correspondence to effective factors, we have

\[ p(x_1, x_2, x_3) \propto \phi_C(x_1, x_2, x_3)\mu_{\phi_A \rightarrow x_1}(x_1)\mu_{\phi_B \rightarrow x_2}(x_2)\mu_{x_3 \rightarrow \phi_C}(x_3) \]

Eliminating \( x_1 \) and \( x_3 \) gives

\[ p(x_2) \propto \mu_{\phi_B \rightarrow x_2}(x_2) \sum_{x_1, x_3} \phi_C(x_1, x_2, x_3)\mu_{x_3 \rightarrow \phi_C}(x_3)\mu_{\phi_A \rightarrow x_1}(x_1) \]

\[ \propto \mu_{\phi_B \rightarrow x_2}(x_2)\mu_{\phi_c \rightarrow x_2}(x_2) \]
Further marginals from messages

We had

$$\mu_{\phi_{c \rightarrow x_2}}(x_2) = \sum_{x_1, x_3} \phi_c(x_1, x_2, x_3) \mu_{x_3 \rightarrow \phi_{c}}(x_3) \mu_{\phi_{A \rightarrow x_1}}(x_1)$$

Introducing variable to factor message $\mu_{x_1 \rightarrow \phi_c} = \mu_{\phi_{A \rightarrow x_1}} = \phi_A$

$$\mu_{\phi_{c \rightarrow x_2}}(x_2) = \sum_{x_1, x_3} \phi_c(x_1, x_2, x_3) \mu_{x_3 \rightarrow \phi_{c}}(x_3) \mu_{x_1 \rightarrow \phi_c}(x_1)$$
All (univariate) marginals from messages

- We can use the messages to compute the marginals of all variables in the graph.
- For the marginal of a variable \( x \) we need to know the incoming messages \( \mu_{\phi_i \rightarrow x} \) from all factor nodes \( \phi_i \) connected to \( x \).
- Achieve by passing messages to root, and then from root.
- This means that if each edge has a message in both directions, we can compute the marginals of all variables in the graph.
Joint distributions from messages

- The correspondence between messages and effective factors allows us to find the joint distribution for variables connected to the same factor node (neighbours).
- For example, we can compute $p(x_3, x_5)$ from messages.
- The messages $\mu_{x_3 \rightarrow \phi_E}$ and $\mu_{x_5 \rightarrow \phi_E}$ correspond to effective factors attached to $x_3$ and $x_5$, respectively.

\[
\mu_{x_3 \rightarrow \phi_E} \quad \mu_{x_5 \rightarrow \phi_E}
\]

- Factor graph corresponds to

\[
p(x_3, x_5) \propto \phi_E(x_3, x_5)\mu_{x_3 \rightarrow \phi_E}(x_3)\mu_{x_5 \rightarrow \phi_E}(x_5)
\]
Rules of message passing: joint marginals

**Note:** The rules come from the fact that messages correspond to effective factors obtained after marginalisation.

Let $x_1, \ldots, x_j$ be all neighbours of factor node $\phi$.

$$p(x_1, \ldots, x_j) = \frac{1}{Z} \phi(x_1, \ldots, x_j) \prod_{i=1}^{j} \mu_{x_i \rightarrow \phi}(x_i)$$
Computing conditionals with message passing

- As in the variable elimination section, we redefine the potentials to take into account evidential variables $x_e$
- This can be viewed as defining a new factor graph on the non-evidential variables
- Or, one can keep the original factor graph, but for factor-to-variable messages, the sum is only taken over non-evidential variables; any evidential variables in the potential are set to their evidential states
A word about numerics

- In practice, it is better to work in the log-domain.
- Log of products of messages $\rightarrow$ sums of log-messages.
- For factor-to-variable messages, we need the log-sum-exp trick:

$$\log \mu_{\phi \rightarrow x}(x) = \log \left( \sum_{x_1, \ldots, x_j} \phi(x_1, \ldots, x_j, x) \prod_{i=1}^{j} \mu_{x_i \rightarrow \phi}(x_i) \right)$$

With $\lambda_i(x_i) = \log \mu_{x_i \rightarrow \phi}(x_i)$, introduce $\omega(x_1, \ldots, x_j, x)$,

$$\omega(x_1, \ldots, x_j, x) = \log \phi(x_1, \ldots, x_j, x) + \log \prod_{i=1}^{j} \mu_{x_i \rightarrow \phi}(x_i)$$

$$= \log \phi(x_1, \ldots, x_j, x) + \sum_{i=1}^{j} \lambda_i(x_i).$$

Depends on $x_1, \ldots, x_j$ and $x$ (assumed fixed here). This gives

$$\log \mu_{\phi \rightarrow x}(x) = \log \left( \sum \exp(\omega(x_1, \ldots, x_j, x)) \right)$$
A word about numerics

- We had

\[
\log \mu_{\phi \to x}(x) = \log \left( \sum_{x_1, \ldots, x_j} \exp(\omega(x_1, \ldots, x_j, x)) \right)
\]

- Sum goes over all possible values of \(x_1, \ldots, x_j\). If the \(\omega(x_1, \ldots, x_j, x)\) are very large or small, we have numerical overflow/underflow problems.

- Introduce \(\omega^*(x) = \max_{x_1, \ldots, x_j} \omega(x_1, \ldots, x_j, x)\) so that

\[
\log \mu_{\phi \to x}(x) = \log \sum_{x_1, \ldots, x_j} \exp(\omega^*(x)) \exp(\omega(x_1, \ldots, x_j, x) - \omega^*(x))
\]

\[
= \log \left( \exp(\omega^*(x)) \sum_{x_1, \ldots, x_j} \exp(\omega(x_1, \ldots, x_j, x) - \omega^*(x)) \right)
\]

\[
= \omega^*(x) + \log \left( \sum_{x_1, \ldots, x_j} \exp(\omega(x_1, \ldots, x_j, x) - \omega^*(x)) \right)
\]

- Numerically stable because \(\exp(\omega(x_1, \ldots, x_j, x) - \omega^*(x)) \leq 1\).
Other names for the sum-product algorithm

- Other names for the sum-product algorithm include:
  - sum-product message passing
  - message passing
  - belief propagation
- Whatever the name: it is variable elimination applied to factor trees
Key advantages of the sum-product algorithm

Assume $p(x_1, \ldots, x_d) \propto \prod_{i=1}^{m} \phi_i(\mathcal{X}_i)$, with $\mathcal{X}_i \subseteq \{x_1, \ldots, x_d\}$, can be represented as a factor tree.

- The sum-product algorithm allows us to compute
  - all univariate marginals $p(x_i)$.
  - all joint distributions $p(\mathcal{X}_i)$ for the variables $\mathcal{X}_i$ that are part of the same factor $\phi_i$.

- Cost: If variables can take maximally $K$ values and there are maximally $M$ elements in the $\mathcal{X}_i$: $O(2dK^M) = O(dK^M)$
Applicability of the sum-product algorithm

- Factor graph must be a tree
- Can be used to compute conditionals (same argument as for variable elimination)
- May be used for continuous random variables (same caveats as for variable elimination)
If the factor graph is not a tree

- Use variable elimination
- Group variables together so that the factor graph becomes a tree (for details, see e.g. Chapter 6 in Barber; not examinable)
- Pretend the factor graph is a tree and use message passing (loopy belief propagation; not examinable)
- Can you condition on some variables so that the conditional is a tree? Message passing can then be used to solve part of the inference problem.

Example: \( p(x_1, x_2, x_3, x_4) \) is not a tree but \( p(x_1, x_2, x_3|x_4) \) is.

Use law of total probability

\[
p(x_1) = \sum_{x_4} \sum_{x_2, x_3} p(x_1, x_2, x_3|x_4) p(x_4)
\]

(see Barber Section 5.3.2, “Loop-cut conditioning”; not examinable)
Summary

1. Marginal inference by variable elimination
   - Exploiting the factorisation by using the distributive law
     \( ab + ac = a(b + c) \) and by caching computations
   - Variable elimination for general factor graphs
   - The principles of variable elimination also apply to continuous random variables

2. Marginal inference for factor trees (sum-product algorithm)
   - Factor trees
   - Message passing for factor trees (sum-product algorithm)
   - The rules for sum-product message passing
   - Illustrating message passing on an example factor tree
Summary

1. Marginal inference by variable elimination
   - Exploiting the factorisation by using the distributive law $ab + ac = a(b + c)$ and by caching computations
   - Variable elimination for general factor graphs
   - The principles of variable elimination also apply to continuous random variables

2. Marginal inference for factor trees (sum-product algorithm)
   - Factor trees
   - Message passing for factor trees (sum-product algorithm)
   - The rules for sum-product message passing
   - Illustrating message passing on an example factor tree
1. Marginal inference by variable elimination

2. Marginal inference for factor trees (sum-product algorithm)

3. Inference of most probable states for factor trees
   - Maximisers of the marginals \( \neq \) maximiser of joint
   - We can exploit the factorisation (in the log-domain) using the distributive law \( \max(u + v, u + w) = u + \max(v, w) \)
   - Max-sum message passing
Other inference tasks

- So far: given a joint distribution \( p(\mathbf{x}) \), find marginals or conditionals over variables.

- Other common inference tasks:
  - Find a setting of the variables that maximises \( p(\mathbf{x}) \), i.e.
    \[
    \hat{\mathbf{x}} = \arg\max_{\mathbf{x}} p(\mathbf{x}) = \arg\max_{\mathbf{x}} \log p(\mathbf{x})
    \]
  - Find the corresponding value maximal value of \( p(\mathbf{x}) \), i.e.
    \[
    p_{\max} = p(\hat{\mathbf{x}}) = \max_{\mathbf{x}} p(\mathbf{x}) \quad \text{or} \quad \\
    \log p_{\max} = \log p(\hat{\mathbf{x}}) = \max_{\mathbf{x}} \log p(\mathbf{x})
    \]

- Note: the task includes \( \arg\max_{\mathbf{x}} \tilde{p}(\mathbf{x}|\mathbf{y}_o) \), which is known as maximum a-posteriori (MAP) estimation or inference.
Maximisers of the marginals $\neq$ maximiser of joint

- The sum-product algorithm gives us the univariate marginals $p(x_i)$ for all variables $x_1, \ldots, x_d$.
- But the vector with the $\text{argmax}_{x_i} p(x_i)$, $x_1, \ldots, x_d$, is not the same as $\text{argmax}_{x} p(x)$.
- Example (Bishop Table 8.1):

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$p(x_1, x_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$p(x_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_2$</th>
<th>$p(x_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.7</td>
</tr>
<tr>
<td>1</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Distributive law to exploit the factorisation

- With the sum-product algorithm, we could compute the marginal $p(x)$ for any $x$ by summing out all other variables and exploiting the factorisation.

- Let us consider the case where $x_d$ is the target variable

$$p(x_d) = \sum_{x_1, \ldots, x_{d-1}} p(x)$$

$$= \frac{1}{Z} \sum_{x_1, \ldots, x_{d-1}} \prod_{i=1}^{m} \phi_i(x_i)$$

- For the max problem, we have $p_{\text{max}} = \max_{x_d} \eta^*(x_d)$

$$\eta^*(x_d) = \max_{x_1, \ldots, x_{d-1}} p(x)$$

$$= \frac{1}{Z} \max_{x_1, \ldots, x_{d-1}} \prod_{i=1}^{m} \phi_i(x_i)$$
Max-product algorithm

- The problem has the same structure with the correspondence

\[
\sum_{x_1, \ldots, x_{d-1}} \rightarrow \max_{x_1, \ldots, x_{d-1}}
\]

- To compute \( p(x_d) \), we relied on the distributive law

\[
ab + ac = a(b + c) \\
\text{sum}(ab, ac) = a \text{sum}(b, c)
\]

- To compute \( \eta^*(x_d) \), we can use the distributive law

\[
\max(ab, ac) = a \max(b, c)
\]

Work in the log-domain

Let us work in the log-domain for numerical stability.

Consider again

\[
p(x_d) = \sum_{x_1, \ldots, x_{d-1}} p(x)
\]

\[
= \frac{1}{Z} \sum_{x_1, \ldots, x_{d-1}} \prod_{i=1}^{m} \phi_i(x_i)
\]

Max problem in the log-domain: \( \log p_{\text{max}} = \max_{x_d} \gamma^*(x_d) \)

\[
\gamma^*(x_d) = \max_{x_1, \ldots, x_{d-1}} \log p(x)
\]

\[
= -\log Z + \max_{x_1, \ldots, x_{d-1}} \sum_{i=1}^{m} \log \phi_i(x_i)
\]
Work in the log-domain

- The problem has the same structure with the correspondence

\[
\sum_{x_1, \ldots, x_{d-1}} \rightarrow \max_{x_1, \ldots, x_{d-1}}, \quad \prod_{i=1}^{m} \rightarrow \sum_{i=1}^{m}, \quad \phi_i(x_i) \rightarrow \log \phi_i(x_i)
\]

- To compute \(p(x_d)\), we relied on the distributive law

\[
ab + ac = a(b + c) \\
\text{sum}(ab, ac) = a \text{ sum}(b, c)
\]

- To compute \(\gamma^*(x_d)\), we can use the distributive law

\[
\max(\log a + \log b, \log a + \log c) = \log a + \max(\log b, \log c)
\]

- Message passing algorithm by replacing sum with max, products with sums, and factors with log-factors.
Sum-product algorithm with $x_d$ as root (recap)

**Factor to variable**

$$\mu_{\phi \rightarrow x}(x) = \sum_{x_1, \ldots, x_j} \phi(x_1, \ldots, x_j, x) \prod_{i=1}^{j} \mu_{x_i \rightarrow \phi}(x_i)$$

where $\{x_1, \ldots, x_j\} = \text{ne}(\phi) \setminus \{x\}$

**Variable to factor**

$$\mu_{x \rightarrow \phi}(x) = \prod_{i=1}^{j} \mu_{\phi_i \rightarrow x}(x)$$

where $\{\phi_1, \ldots, \phi_j\} = \text{ne}(x) \setminus \{\phi\}$

**Univariate marginal**

$$p(x_d) = \frac{1}{Z} \prod_{i=1}^{j} \mu_{\phi_i \rightarrow x_d}(x_d)$$

$$Z = \sum_{x_d} \prod_{i=1}^{j} \mu_{\phi_i \rightarrow x_d}(x_d)$$

where $\{\phi_1, \ldots, \phi_j\} = \text{ne}(x_d)$

**Initialisation**

At leaf variable nodes: $\mu_{x \rightarrow \phi}(x) = 1$

At leaf factor nodes: $\mu_{\phi \rightarrow x}(x) = \phi(x)$
Max-sum algorithm with $x_d$ as root

**Factor to variable**

$$\gamma_{\phi \rightarrow x}(x) = \max_{x_1, \ldots, x_j} \log \phi(x_1, \ldots, x_j, x) + \sum_{i=1}^j \gamma_{x_i \rightarrow \phi}(x_i)$$

where $\{x_1, \ldots, x_j\} = \text{ne}(\phi) \setminus \{x\}$

**Variable to factor**

$$\gamma_{x \rightarrow \phi}(x) = \sum_{i=1}^j \gamma_{\phi_i \rightarrow x}(x)$$

where $\{\phi_1, \ldots, \phi_j\} = \text{ne}(x) \setminus \{\phi\}$

**Maximum probability**

$$\gamma^*(x_d) = -\log Z + \sum_{i=1}^j \gamma_{\phi_i \rightarrow x_d}(x_d)$$

$$\log p_{\text{max}} = \max_{x_d} \gamma^*(x_d)$$

where $\{\phi_1, \ldots, \phi_j\} = \text{ne}(x_d)$

**Initialisation**

At leaf variable nodes: $\gamma_{x \rightarrow \phi}(x) = 0$

At leaf factor nodes: $\gamma_{\phi \rightarrow x}(x) = \log \phi(x)$
Max-sum algorithm

- After computation of $\gamma^*(x_d)$, we obtain
  \[
  \log p_{\text{max}} = \max_{x_d} \gamma^*(x_d)
  \]
  Result does not depend on choice of $x_d$.
- Compute $\hat{x} = \arg\max_x p(x)$ recursively via “backtracking”.
- When solving the optimisation problem
  \[
  \gamma_{\phi \rightarrow x}(x) = \max_{x_1, \ldots, x_j} \log \phi(x_1, \ldots, x_j, x) + \sum_{i=1}^{j} \gamma_{x_i \rightarrow \phi}(x_i)
  \]
  we also build the function (look-up table)
  \[
  \gamma^*_{\phi \rightarrow x}(x) = \arg\max_{x_1, \ldots, x_j} \log \phi(x_1, \ldots, x_j, x) + \sum_{i=1}^{j} \gamma_{x_i \rightarrow \phi}(x_i)
  \]
  which returns the maximiser $(\hat{x}_1, \ldots, \hat{x}_j)$ for each value of $x$.
- Start the recursion with $\hat{x}_d = \arg\max_{x_d} \gamma^*(x_d)$, backtrack to the leaf variables to compute $\hat{x}$. 
Example

Model (pmf):

\[ p(x_1, x_2, x_3, x_4) \propto \phi_A(x_1) \phi_B(x_2) \phi_C(x_1, x_2, x_3) \phi_D(x_3, x_4) \]

Factor graph (tree):

Goal: \( (\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4) = \arg\max_{x_1, \ldots, x_4} p(x_1, x_2, x_3, x_4) \)
Example

- Select root towards which we send messages. Here: $x_4$.
- Messages that we need to send:

\[ \phi_B \downarrow \]

\[ \phi_A \rightarrow \]

\[ \phi_C \downarrow \]

\[ \phi_D \rightarrow \]

\[ \phi_A \rightarrow x_1(x_1) = \log \phi_A(x_1) \]

\[ \phi_B \rightarrow x_2(x_2) = \log \phi_B(x_2) \]

- Initialise:
Example

$x_1$ and $x_2$ copy the messages:

\[
\gamma_{x_1 \rightarrow \phi_C} (x_1) = \gamma_{\phi_A \rightarrow x_1} (x_1)
\]

\[
\gamma_{x_2 \rightarrow \phi_C} (x_2) = \gamma_{\phi_B \rightarrow x_2} (x_2)
\]

For $\gamma_{\phi_C \rightarrow x_3} (x_3)$ solve optimisation problem

\[
\gamma_{\phi_C \rightarrow x_3} (x_3) = \max_{x_1, x_2} \left[ \log \phi_C (x_1, x_2, x_3) + \gamma_{x_1 \rightarrow \phi_C} (x_1) + \gamma_{x_2 \rightarrow \phi_C} (x_2) \right]
\]

\[
\gamma_{\phi_C \rightarrow x_3}^* (x_3) = \arg\max_{x_1, x_2} \left[ \log \phi_C (x_1, x_2, x_3) + \gamma_{x_1 \rightarrow \phi_C} (x_1) + \gamma_{x_2 \rightarrow \phi_C} (x_2) \right]
\]

for all values of $x_3$. 

Example

- $x_3$ copies the message: $\gamma_{x_3 \rightarrow \phi_D}(x_3) = \gamma_{\phi_C \rightarrow x_3}(x_3)$
- For $\gamma_{\phi_D \rightarrow x_4}(x_4)$ solve optimisation problem

\[
\gamma_{\phi_D \rightarrow x_4}(x_4) = \max_{x_3} \left[ \log \phi_D(x_3, x_4) + \gamma_{x_3 \rightarrow \phi_D}(x_3) \right]
\]

\[
\gamma_{\phi_D \rightarrow x_4}^*(x_4) = \arg\max_{x_3} \left[ \log \phi_D(x_3, x_4) + \gamma_{x_3 \rightarrow \phi_D}(x_3) \right]
\]

for all values of $x_4$. 
After computation of $\gamma_{\phi_D \to x_4}(x_4)$, we obtain $\log p_{\max}$ as

$$\log p_{\max} = \max_{x_d} \gamma^*(x_d)$$

$$\gamma^*(x_4) = -\log Z + \gamma_{\phi_D \to x_4}(x_4)$$

This requires knowledge of $Z$. We can compute $Z$ via the sum-product algorithm.

$Z$ not needed if we are only interested in $\arg\max p(x_1, \ldots, x_4)$
Example

Backtracking:

- Compute $\hat{x}_4 = \arg\max_{x_4} \gamma^*(x_4) = \arg\max_{x_4} \gamma_{\phi_D \rightarrow x_4}(x_4)$
- Plug $\hat{x}_4$ into look-up table $\gamma^*_{\phi_D \rightarrow x_4}(x_4)$ to look up best value of $x_3$:

  $$\hat{x}_3 = \gamma^*_{\phi_D \rightarrow x_4}(\hat{x}_4)$$

- Plug $\hat{x}_3$ into look-up table $\gamma^*_{\phi_C \rightarrow x_3}(x_3)$ to look up best values of $(x_1, x_2)$:

  $$(\hat{x}_1, \hat{x}_2) = \gamma^*_{\phi_C \rightarrow x_3}(\hat{x}_3)$$

- This gives $(\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4) = \arg\max_{x_1, \ldots, x_4} p(x_1, x_2, x_3, x_4)$
1. Marginal inference by variable elimination
   - Exploiting the factorisation by using the distributive law
   \[ab + ac = a(b + c)\] and by caching computations
   - Variable elimination for general factor graphs
   - The principles of variable elimination also apply to continuous random variables

2. Marginal inference for factor trees (sum-product algorithm)
   - Factor trees
   - Message passing for factor trees (sum-product algorithm)
   - The rules for sum-product message passing
   - Illustrating message passing on an example factor tree

3. Inference of most probable states for factor trees
   - Maximisers of the marginals \(\neq\) maximiser of joint
   - We can exploit the factorisation (in the log-domain) using the distributive law
   \[
   \max (u + v, u + w) = u + \max (v, w)
   \]
   - Max-sum message passing
Further Reading

- Bishop secs. 8.4.3-8.4.5 covers factor trees, sum-product and max-product inference
- Also Barber secs. 5.1, 5.2.1
- The topics are also covered in many other sources
These slides are modified from ones produced by Michael Gutmann, made available under Creative Commons licence CC BY 4.0.

©Michael Gutmann and Chris Williams, The University of Edinburgh 2018-2024 CC BY 4.0 🅿️.