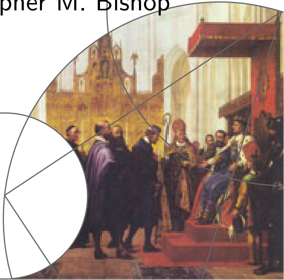
Graphical models II

Elements of Machine Learning

Jens Petersen

with content and images from
Pattern Recognition and Machine Learning, Christopher M. Bishop



About this lecture

Methods for inference in graphical models

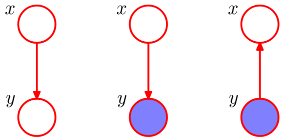
- Factor graphs
- Sum-Product algorithm
- Max-Sum algorithm
- Sampling methods

Inference in graphical models

- Graphical model structure allow efficient algorithms for inference
- By for instance passing local messages around the graph

A basic example

Graphical representation of Bayes' theorem



Suppose we are interested in the posterior distribution of x given y (right figure).

We can write the joint distribution as $p(x, y) = p(x)p(y|x)$ and using the sum and product rules of probability

$$p(y) = \sum_{x'} p(y|x')p(x')$$

which we can combine with Bayes' to give us

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

Inference on a chain

Joint distribution

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

Marginal probability of x_n by definition, sum over all variables except x_n

$$p(x_n) = \sum_{x_1}\cdots\sum_{x_{n-1}}\sum_{x_{n+1}}\cdots\sum_{x_N} p(\mathbf{x})$$

which is exponential in N.

With chain we can do better, because each potential $\psi_{N-1,N}(x_{N-1}, x_N)$ is the only one that depends on x_N. So we only need this term in the summation.

$$\sum_{x_N} = \psi_{N-1,N}(x_{N-1}, x_N)$$

Inference on a chain

We can proceed recursively from both ends of the chain towards x_n to give

$$p(x_n) = \frac{1}{Z}$$

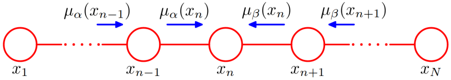
$$\underbrace{\left(\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \dots \left(\sum_{x_2} \psi_{2,3}(x_2, x_3) \left(\sum_{x_1} \psi_{1,2}(x_1, x_2) \right) \right) \dots \right)}_{\mu_\alpha(x_n)}$$

$$\underbrace{\left(\sum_{x_{n+1}} \psi_{n,n+1} \dots \left(\sum_{x_n} \psi_{N-1,N}(x_{N_1}, x_N) \right) \dots \right)}_{\mu_\beta(x_n)}$$

Message passing

- $\mu_{alpha}(x_n)$ forward through the chain
- $\mu_{beta}(x_n)$ backward through the chain

Inference on a chain



Forward messages

$$\mu_\alpha(x_n) = \begin{cases} \sum_{x_1} \psi_{1,2}(x_1, x_2) & \text{for } n = 2 \\ \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n)\mu_\alpha(x_{n-1}) & \text{else.} \end{cases}$$

Backward messages

$$\mu_\beta(x_n) = \begin{cases} \sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) & \text{for } n = N \\ \sum_{x_{n+1}} \psi_{n+1,n}(x_{n+1}, x_n)\mu_\beta(x_{n+1}) & \text{else.} \end{cases}$$

Normalization constant (choose any node n and sum messages)

$$Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$$

Inference on a chain

What if want to compute marginals $p(x_n)$ for all n?

- wasteful as many messages are repeated

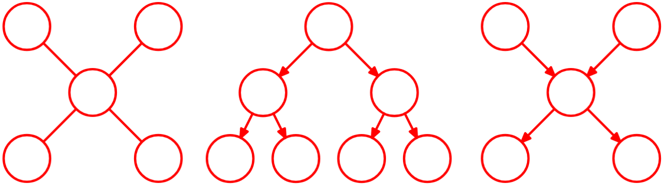
Solution:

1. compute and store all forward messages, $\mu_\alpha(x_n)$
2. compute and store all backward messages, $\mu_\beta(x_n)$
3. compute Z at any node x_m
4. compute all $p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$

Trees

Efficient inference possible in tree-structured graphs as well...

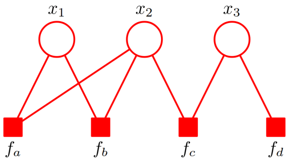
- Undirected tree - one and only one path between any pair of nodes (no loops)
- Directed tree - a single node (**root**) with no parents, and all other nodes have one parent
  - Moralization leads to an undirected tree
- Directed polytree - nodes with more than one parent exist but only one path (ignoring edge direction) between any pair of nodes



Undirected tree, directed tree and a directed polytree

Factor graphs[1][2]

A factor graph is a bipartite graph representing the factorization of a function



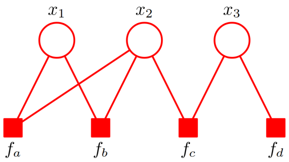
- one variable node for each variable
- one factor node for each function
- one edge between x_i and f_j if x_i is a variable in f_j

[1] wikipedia.org

[2] `https://www.doc.ic.ac.uk/~mpd37/teaching/ml_tutorials/2016-11-09-Svensson-BP.pdf`

Factor graphs[1][2]

A factor graph is a bipartite graph representing the factorization of a function



Write distribution as product of factors

$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$$

where x_s denotes subset of the variables. Example:

$$p(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

[1] wikipedia.org

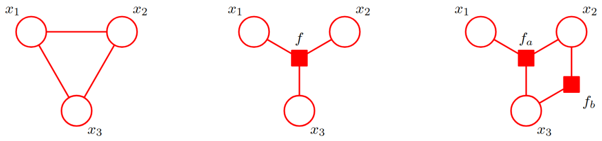
[2] `https://www.doc.ic.ac.uk/~mpd37/teaching/ml_tutorials/2016-11-09-Svensson-BP.pdf`

Why factor graphs?

Allows one to

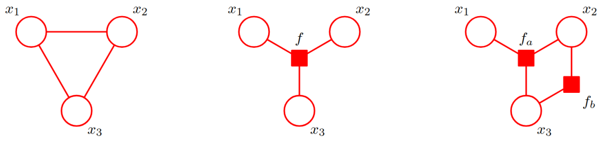
- visualize the problem structure
  - clarify dependencies as well as how a complicated problem can be split into smaller functions of fewer variables
- efficiently compute marginals through algorithms such as the Sum-Product algorithm

Factor graphs from undirected graphs



- left: undirected graph with single clique potential $\psi(x_1, x_2, x_3)$
- middle: factor graph representing the same distribution with $f = \psi(x_1, x_2, x_3)$
- right: another factor graph representing the same distribution with $f_a(x_1, x_2, x_3)f_b(x_1, x_2) = \psi(x_1, x_2, x_3)$.

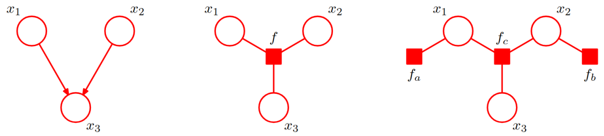
Factor graphs from undirected graphs



- left: undirected graph with single clique potential $\psi(x_1, x_2, x_3)$
- middle: factor graph representing the same distribution with $f = \psi(x_1, x_2, x_3)$
- right: another factor graph representing the same distribution with $f_a(x_1, x_2, x_3) f_b(x_1, x_2) = \psi(x_1, x_2, x_3)$.

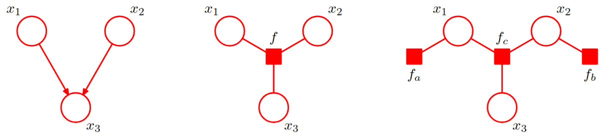
Convertion: create variable nodes corresponding to nodes in the original graph, and then create additional factor nodes corresponding to the maximal cliques $\mathbf{x}_s$, with the factors $f_s(\mathbf{x}_s)$ equal to the clique potentials.

Factor graphs from directed graphs



- left: directed graph with factorization $p(x_1)p(x_2)p(x_3|x_1, x_2)$
- middle: factor graph of the same distribution with $f(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3|x_1, x_2)$
- right: another factor graph of the same distribution with $f_a(x_1) = p(x_1)$, $f_b(x_2) = p(x_2)$ and $f_c(x_1, x_2, x_3) = p(x_3|x_1, x_2)$.

Factor graphs from directed graphs



- left: directed graph with factorization $p(x_1)p(x_2)p(x_3|x_1, x_2)$
- middle: factor graph of the same distribution with $f(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3|x_1, x_2)$
- right: another factor graph of the same distribution with $f_a(x_1) = p(x_1)$, $f_b(x_2) = p(x_2)$ and $f_c(x_1, x_2, x_3) = p(x_3|x_1, x_2)$.

Convertion: create variable nodes corresponding to nodes in the original graph, and then create factor nodes corresponding to the conditional distributions, and then finally add the appropriate links.

The Sum-Product algorithm

The algorithm known as Belief Propagation is a special case of the Sum-Product algorithm

- Goal: to obtain an efficient, exact inference algorithm for finding (multiple) marginals
  - Messages from variable to factor nodes
  - Messages from factor to variable nodes

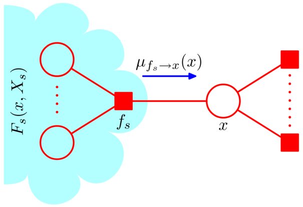
The Sum-Product algorithm

The algorithm known as Belief Propagation is a special case of the Sum-Product algorithm

- Goal: to obtain an efficient, exact inference algorithm for finding (multiple) marginals
  - Messages from variable to factor nodes
  - Messages from factor to variable nodes

Why is it called belief propagation? A message to a variable node can be thought of as the message senders 'beliefs' about the state of that variable node

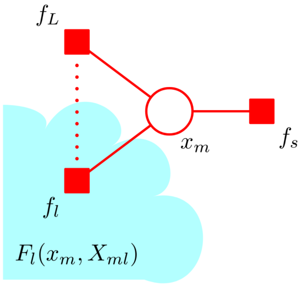
The Sum-Product algorithm



Each message from a factor node (f_s) to variable node (x) is the product of the factor with messages from all neighboring nodes marginalized over all variables except the one associated with x.

$$\mu_{f_s \to x}(x) = \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in ne(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

The Sum-Product algorithm

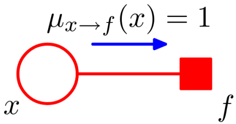


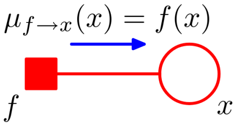
Each message from a variable node (x) to a factor node (f_s) is the product of all messages arriving from neighboring factor nodes except the recipient.

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in ne(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

The Sum-Product algorithm

Initialization





Initialize messages at the leaves of the graph

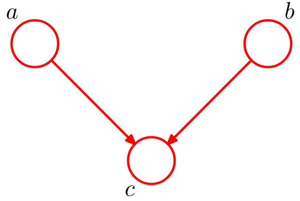
The Sum-Product algorithm

To compute local marginals

1. pick an arbitrary node as the root
2. compute and propagate messages from the leaf nodes towards the root, storing received messages at every node
    - a node can send a messages towards the root once it has received messages from all of its other neighbours
3. compute and propagate messages from the root towards the leaf nodes, storing received messages at every node
4. compute the product of received messages at each node for which the marginal is required, and normalize if needed (how do we normalize? Remember the procedure for the chain graph.)

The Sum-Product algorithm - example I

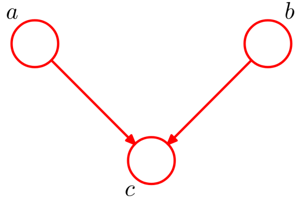
Suppose we are given the following graph and corresponding joint distribution



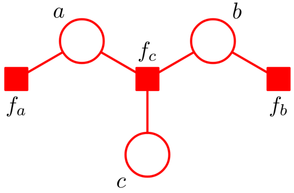
$$p(a, b, c) = p(c|a, b)p(a)p(b),$$

The Sum-Product algorithm - example I

We can convert this into the following factor graph



$\longrightarrow$



$$p(a, b, c) = p(c|a, b)p(a)p(b),$$

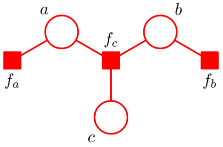
The Sum-Product algorithm - example I

Additionally, the distributions for a and b and the conditional distribution of c given a and b are

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
2	0.3

a	b	$p(c=0 \mid a,b)$	$p(c=1 \mid a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9

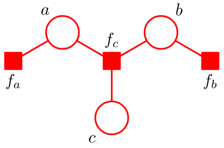


The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
2	0.3

a	b	$p(c=0 \mid a,b)$	$p(c=1 \mid a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



We start by selecting a node as root, lets choose c and begin passing messages from the leafs towards the root.

$\mu_{f_a \to a}(a) =$

a	f_a
0	0.7
1	0.3

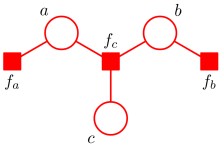
The message from factor f_a to variable a

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
2	0.3

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



$\mu_{f_a \to a}(a) =$

a	f_a
0	0.7
1	0.3

$\mu_{f_b \to b}(b) =$

b	f_b
0	0.5
1	0.2
3	0.3

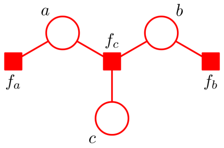
The message from factor f_b to variable b

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
2	0.3

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



$\mu_{f_a \to a}(a) =$

a	f_a
0	0.7
1	0.3

$\mu_{f_b \to b}(b) =$

b	f_b
0	0.5
1	0.2
3	0.3

$\mu_{a \to f_c}(a) = \mu_{f_a \to a}(a) =$

a	f_a
0	0.7
1	0.3

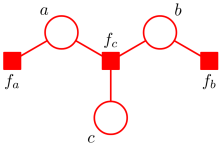
The message from variable a to factor f_c

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
2	0.3

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



$\mu_{f_b \to b}(b) =$

b	f_b
0	0.5
1	0.2
3	0.3

$\mu_{a \to f_c}(a) = \mu_{f_a \to a}(a) =$

a	f_a
0	0.7
1	0.3

$\mu_{b \to f_c}(b) = \mu_{f_b \to b}(b) =$

b	f_b
0	0.5
1	0.2
3	0.3

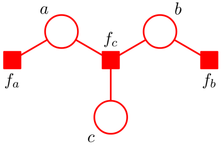
The message from variable b to factor f_c

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
2	0.3

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



$\mu_{a\to f_c}(a) = \mu_{f_a\to a}(a) =$

a	f_a
0	0.7
1	0.3

$\mu_{b\to f_c}(b) = \mu_{f_b\to b}(b) =$

b	f_b
0	0.5
1	0.2
3	0.3

$$\mu_{f_c\to c}(c) = \sum_a \sum_b f_c(a,b,c)\mu_{a\to f_c}(a)\mu_{b\to f_c}(b)$$

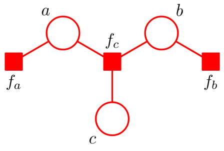
The message from factor f_c to variable c

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
3	0.3

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



When multiplying factors, think combining probabilities of events!

$$\mu_{a\to f_c}(a) = \mu_{f_a\to a}(a) =$$

a	f_a
0	0.7
1	0.3

$$\mu_{b\to f_c}(b) = \mu_{f_b\to b}(b) =$$

b	f_b
0	0.5
1	0.2
3	0.3

$$\mu_{f_c\to c}(c) = \sum_a \sum_b f_c(a,b,c)\mu_{a\to f_c}(a)\mu_{b\to f_c}(b) = \sum_a \sum_b f_c(a,b,c)$$

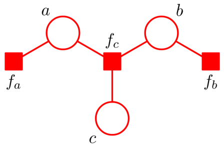
a	b	$f_a \cdot f_b$
0	0	$0.7 \cdot 0.5$
0	1	$0.7 \cdot 0.2$
0	2	$0.7 \cdot 0.3$
1	0	$0.3 \cdot 0.5$
1	1	$0.3 \cdot 0.2$
1	2	$0.3 \cdot 0.3$

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
3	0.3

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



Leading to...

$$\mu_{a\to f_c}(a) = \mu_{f_a\to a}(a) =$$

a	f_a
0	0.7
1	0.3

$$\mu_{b\to f_c}(b) = \mu_{f_b\to b}(b) =$$

b	f_b
0	0.5
1	0.2
3	0.3

$$\mu_{f_c\to c}(c) = \sum_a \sum_b f_c(a,b,c)\mu_{a\to f_c}(a)\mu_{b\to f_c}(b) = \sum_a \sum_b f_c(a,b,c)$$

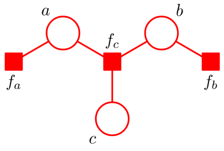
a	b	$f_a \cdot f_b$
0	0	0.35
0	1	0.14
0	2	0.21
1	0	0.15
1	1	0.06
1	2	0.09

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
3	0.3

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



Writing out the factor f_c

$$\mu_{f_c \to c}(c) = \sum_a \sum_b$$

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9

$\cdot$

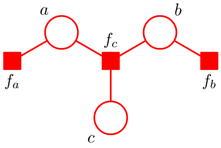
a	b	$f_a \cdot f_b$
0	0	0.35
0	1	0.14
0	2	0.21
1	0	0.15
1	1	0.06
1	2	0.09

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
3	0.3

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



Product of f_c, f_a and f_b

$$\mu_{f_c \to c}(c) = \sum_a \sum_b$$

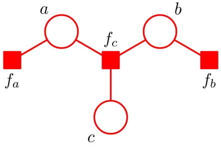
a	b	$p(c=0,a,b)$	$p(c=1,a,b)$
0	0	$0.1 \cdot 0.35$	$0.9 \cdot 0.35$
0	1	$0.2 \cdot 0.14$	$0.8 \cdot 0.14$
0	2	$0.3 \cdot 0.21$	$0.7 \cdot 0.21$
1	0	$0.3 \cdot 0.15$	$0.7 \cdot 0.15$
1	1	$0.2 \cdot 0.06$	$0.8 \cdot 0.06$
1	2	$0.1 \cdot 0.09$	$0.9 \cdot 0.09$

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
3	0.3

a	b	$p(c=0\|a,b)$	$p(c=1\|a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



Giving... Now all that is needed is to marginalize over a and b

$$\mu_{f_c \to c}(c) = \sum_a \sum_b$$

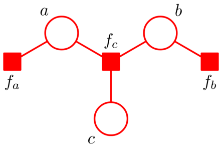
a	b	$p(c=0,a,b)$	$p(c=1,a,b)$
0	0	0.035	0.315
0	1	0.028	0.112
0	2	0.063	0.147
1	0	0.045	0.105
1	1	0.012	0.048
1	2	0.009	0.081

The Sum-Product algorithm - example I

a	$p(a)$
0	0.7
1	0.3

b	$p(b)$
0	0.5
1	0.2
3	0.3

a	b	$p(c=0 \mid a,b)$	$p(c=1 \mid a,b)$
0	0	0.1	0.9
0	1	0.2	0.8
0	2	0.3	0.7
1	0	0.3	0.7
1	1	0.2	0.8
1	2	0.1	0.9



Leading to...

$$\mu_{f_c \to c}(c) = \sum_a \sum_b$$

a	b	$p(c=0,a,b)$	$p(c=1,a,b)$
0	0	0.035	0.315
0	1	0.028	0.112
0	2	0.063	0.147
1	0	0.045	0.105
1	1	0.012	0.048
1	2	0.009	0.081

$=$

c	$p(c)$
0	0.192
1	0.808

The Sum-Product algorithm - example I

At this point we could start sending the messages backwards (root to leafs) to compute the marginal probabilities of the remaining variable nodes, but these are already given in this case.

As an exercise you can convince yourself that doing this indeed does not change the marginal probabilities of a and b.

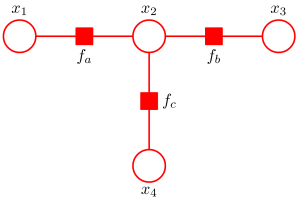
The Sum-Product algorithm - example I

At this point we could start sending the messages backwards (root to leafs) to compute the marginal probabilities of the remaining variable nodes, but these are already given in this case.

As an exercise you can convince yourself that doing this indeed does not change the marginal probabilities of a and b.

Note what a message is somewhat unclear from this writeup. However, they may for instance be implemented as vectors. One just needs to carefully handle the products and the input and output dimensions.

The Sum-Product algorithm - example II



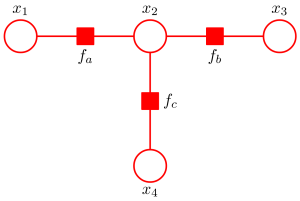
Unnormalized joint distribution

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

The Sum-Product algorithm - example II

Unnormalized joint distribution

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



Pick x_3 as root and compute messages towards it

$$\mu_{x_1 \to f_a}(x_1) = 1$$

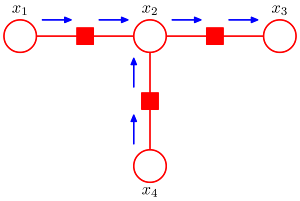
$$\mu_{f_a \to x_2}(x_2) = \sum_{x_1} f_a(x_1, x_2)$$

$$\mu_{x_4 \to f_c}(x_4) = 1$$

$$\mu_{f_c \to x_2}(x_2) = \sum_{x_4} f_c(x_2, x_4)$$

$$\mu_{x_2 \to f_b}(x_2) = \mu_{f_a \to x_2}(x_2) \mu_{f_c \to x_2}(x_2)$$

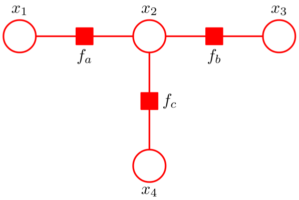
$$\mu_{f_b \to x_3}(x_3) = \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \to f_b}(x_2)$$



The Sum-Product algorithm - example II

Unnormalized joint distribution

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



Compute messages from x_3 towards the leafs

$$\mu_{x_3 \to f_b}(x_3) = 1$$

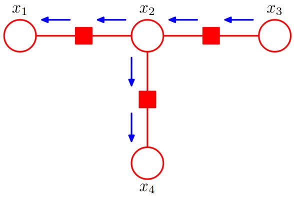
$$\mu_{f_b \to x_2}(x_2) = \sum_{x_3} f_b(x_2, x_3)$$

$$\mu_{x_2 \to f_a}(x_2) = \mu_{f_b \to x_2}(x_2) \mu_{f_c \to x_2}(x_2)$$

$$\mu_{f_a \to x_1}(x_1) = \sum_{x_2} f_a(x_1, x_2) \mu_{x_2 \to f_a}(x_2)$$

$$\mu_{x_2 \to f_c}(x_2) = \mu_{f_a \to x_2}(x_2) \mu_{f_b \to x_2}(x_2)$$

$$\mu_{f_c \to x_4}(x_4) = \sum_{x_2} f_c(x_2, x_4) \mu_{x_2 \to f_c}(x_2)$$



The Sum-Product algorithm - example II - check

$$\begin{aligned}
\widetilde{p}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
&= \left[\sum_{x_1} f_a(x_1, x_2)\right]\left[\sum_{x_3} f_b(x_2, x_3)\right]\left[\sum_{x_4} f_c(x_2, x_4)\right] \\
&= \sum_{x_1}\sum_{x_2}\sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) \\
&= \sum_{x_1}\sum_{x_3}\sum_{x_4} \widetilde{p}(\mathbf{x})
\end{aligned}$$

The Max-Sum algorithm

The maximum of the joint distribution is not necessarily at same values as the maxima of each marginal.

Example of a joint distribution over two binary variables for which the maximum of the joint distribution occurs for different variable values compared to the maxima of the two marginals.

	$x = 0$	$x = 1$
$y = 0$	0.3	0.4
$y = 1$	0.3	0.0

The Max-Sum algorithm

Goal: we want to find the most likely variable configuration and how likely it is.

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

Approach: write out the max in terms of components of **x**

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_1} \ldots \max_{x_M} p(\mathbf{x})$$

and exploit

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

where M is the number of variables and $a \geq 0$, to allow the exchange of products with maximizations across each factor.

The Max-Sum algorithm - chain

$$\begin{aligned}
\max_{\mathbf{x}} p(\mathbf{x}) &= \frac{1}{Z} \max_{x_1} \dots \max_{x_N} \left(\psi_{x_1,x_2}(x_1, x_2) \dots \psi_{N-1,N}(x_{N-1}, x_N) \right) \\
&= \frac{1}{Z} \max_{x_1} \left(\max_{x_2} \left(\psi_{1,2}(x_1, x_2) \left(\dots \max_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right) \right) \right)
\end{aligned}$$

The Max-Sum algorithm

Generalizes to tree-structured factor graph

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_n} \prod_{f_s \in ne(x_n)} \max_{X_s} f_s(x_n, X_s)$$

maximizing as close to the leaf nodes as possible

This could be called the Max-Product algorithm, but computing the product of a lot of small numbers may lead to numerical issues, so use

$$\ln\left(\max_{\mathbf{x}} p(\mathbf{x})\right) = \max_{\mathbf{x}} \ln p(\mathbf{x})$$

and again, use distributive law

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

to replace all the products with sums.

The Max-Sum algorithm

As with Sum-Product, we can now choose a root node and...
initialization (leaf nodes)

- in Sum-Product we have

$$\mu_{x\to f}(x) = 1$$
$$\mu_{f\to x}(x) = f(x)$$

- in Max-Sum we have

$$\mu_{x\to f}(x) = 0$$
$$\mu_{f\to x}(x) = \ln f(x)$$

The Max-Sum algorithm

and recursion

- in Sum-Product we have

$$\mu_{f\to x}(x) = \sum_{x_1}\cdots\sum_{x_M} f(x, x_1, \ldots, x_M) \prod_{m\in ne(f)\setminus x} \mu_{x_m\to f}(x_m)$$

$$\mu_{x\to f}(x) = \prod_{l\in ne(x_m)\setminus f} \mu_{f_l\to x}(x)$$

- in Max-Sum we have

$$\mu_{f\to x}(x) = \max_{x_1,\ldots x_M} \big(\ln f(x, x_1, \ldots, x_M) + \sum_{m\in ne(f_s)\setminus x} \mu_{x_m\to f}(x_m)\big)$$

$$\mu_{x\to f}(x) = \sum_{l\in ne(x)\setminus f} \mu_{f_l\to x}(x)$$

The Max-Sum algorithm

Termination at root node

- in Sum-Product we have

$$p(x) = \prod_{s \in ne(x)} \mu_{f_s \to x}(x)$$

- in Max-Sum we have

$$p^{\max} = \max_x \Big(\sum_{s \in ne(x)} \mu_{f_s \to x}(x) \Big)$$

$$x^{\max} = \arg\max_x \Big(\sum_{s \in ne(x)} \mu_{f_s \to x}(x) \Big)$$

The Max-Sum algorithm

How to find the remaining variables?

The Max-Sum algorithm

How to find the remaining variables?

- Because multiple configurations may give rise to the same p^{max} we cannot simply send messages back towards the leafs
- We can store the variables giving rise to maximum in the forward pass for each variable

$$\phi(x) = \arg \max_{x_1,\dots x_M} \left(\ln f(x, x_1, \dots, x_M) + \sum_{m \in ne(f_s) \setminus x} \mu_{x_m \to f}(x_m) \right)$$

- and follow them backwards

Inference in general graphs

Exact

- Junction tree algorithm
  - Often not feasible

Approximate

- Variational methods (covered later in course)
  - Deterministic
- Sampling methods
  - Stochastic
- Loopy belief propagation
  - Handling loops by repeatedly passing messages around until convergence

Sampling methods

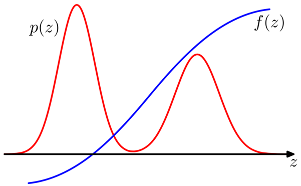
Sampling has been mentioned multiple times

- Ancestral sampling
- Generative models

Sampling distributions can be used to approximate

- the posterior distribution $p(\mathbf{z}|\mathbf{x})$.
- the expectation step in EM.
- exemplify joint distributions

Sampling methods



Goal: find the expectation of some function $f(\mathbf{x})$ with respect to a probability distribution

$$E[f] = \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

Given samples $\mathbf{z}^{(l)}, l \in \{1, \dots L\}$ drawn independently from $p(\mathbf{z})$, we can approximate by

$$\hat{f} = \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^{(l)})$$

Sampling methods

Given samples $\mathbf{z}^{(l)}, l \in \{1, \dots L\}$ drawn independently from $p(\mathbf{z})$, we can approximate by

$$\hat{f} = \frac{1}{L} \sum_{l=1}^{L} f(\mathbf{z}^{(l)})$$

We have

$$E[\hat{\mathbf{z}}] = E[\mathbf{z}]$$

Sampling methods

Given samples $\mathbf{z}^{(l)}, l \in \{1, \dots L\}$ drawn independently from $p(\mathbf{z})$, we can approximate by

$$\hat{f} = \frac{1}{L} \sum_{l=1}^{L} f(\mathbf{z}^{(l)})$$

We have

$$E[\hat{\mathbf{z}}] = E[\mathbf{z}]$$

and

$$\mathrm{var}[\hat{f}] = \frac{1}{L}[(f - E[f])^2]$$

Note the variance does not depend on the dimension of $\mathbf{x}$, but we need independent samples from $p(\mathbf{z})$!

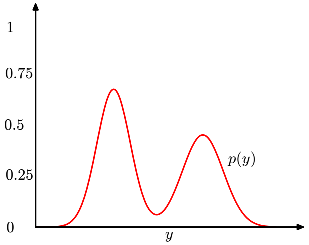
Sampling methods - inverse transform method

Given: a uniformly sampled distribution of z on the interval $(0, 1)$, and a distribution of y which is related to z through $y = f(z)$

Goal: sample y

Sampling methods - inverse transform method

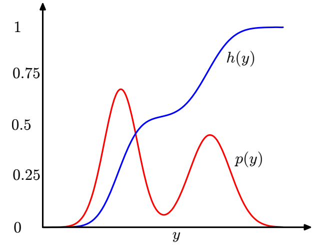
Using the cumulative distribution



Here $p(y)$ is the distribution we want to sample, where $y = f(z)$.

Sampling methods - inverse transform method

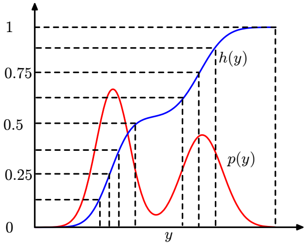
Using the cumulative distribution



Here $p(y)$ is the distribution we want to sample, where $y = f(z)$. The function $h(y)$ is the cumulative distribution of $p(y)$, that is $h(y) = \int_{-\infty}^{y} p(y)$.

Sampling methods - inverse transform method

Using the cumulative distribution



Here $p(y)$ is the distribution we want to sample, where $y = f(z)$. The function $h(y)$ is the cumulative distribution of $p(y)$, that is $h(y) = \int_{-\infty}^{y} p(y)$. So if we can compute $h^{-1}(z)$, then we can sample $p(y)$ as $y = h^{-1}(z)$.

Sampling methods - inverse transform example

Exponential distribution

$$p(y) = \lambda \exp(-\lambda y)$$

where $0 \leq y < \infty$.

Sampling methods - inverse transform example

Exponential distribution

$$p(y) = \lambda \exp(-\lambda y)$$

where $0 \leq y < \infty$. Integrating gives

$$h(y) = \int_0^\infty p(y) = 1 - \exp(-\lambda y)$$

Sampling methods - inverse transform example

Exponential distribution

$$p(y) = \lambda \exp(-\lambda y)$$

where $0 \leq y < \infty$. Which can be inverted to give

$$h^{-1}(z) = -\lambda^{-1} \ln(1 - z)$$

So if we sample $\hat{z}$ uniformly in $(0, 1)$ and compute $\hat{y} = h^{-1}(\hat{z})$ then $\hat{y}$ will be sampled according to an exponential distribution.

Rejection sampling

Assumptions

- Sampling directly from $p(\mathbf{z})$ is difficult but we are able to evaluate $p(\mathbf{z})$ for any value of $\mathbf{z}$ up to an unknown normalization constant Z_p

$$p(z) = \frac{1}{Z_p}\tilde{p}(z)$$

- We can sample from a simpler **proposal distribution** $q(z)$ for which there exists a constant k such that

$$kq(z) \geq \tilde{p}(z)$$

Rejection sampling

Assumptions

- Sampling directly from $p(\mathbf{z})$ is difficult but we are able to evaluate $p(\mathbf{z})$ for any value of $\mathbf{z}$ up to an unknown normalization constant Z_p

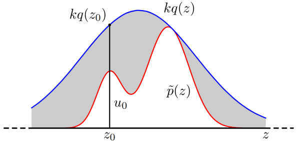
$$p(z) = \frac{1}{Z_p} \tilde{p}(z)$$

- We can sample from a simpler **proposal distribution** $q(z)$ for which there exists a constant k such that

$$kq(z) \geq \tilde{p}(z)$$

Then we can sample from $p(\mathbf{z})$ using rejection sampling

Rejection sampling



Samples are rejected if they fall in the grey area between the unnormalized distribution $\tilde{p}(z)$ and the scaled distribution $kq(z)$.

1. generate random number z_0 from distribution $q(z)$
2. generate random number u_0 from uniform distribution over $[0, kq(z_0)]$
3. if $u_0 > \tilde{p}(z_0)$ then the sample is rejected, otherwise it is retained.

The retained z are distributed according $p(z)$.

Rejection sampling

$$\begin{aligned} p(\mathrm{accept}) &= \int \frac{\tilde{p}(z)}{kq(z)} q(z) dz \\ &= \frac{1}{k} \int \tilde{p}(z) dz \end{aligned}$$

So k should be as small as possible subject to the limitation that $kq(z) > \tilde{p}(z)$ for all $z \rightarrow$ the proposal distribution q should be close to p

Rejection sampling

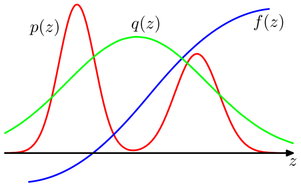
$$\begin{aligned} p(\text{accept}) &= \int \frac{\tilde{p}(z)}{kq(z)} q(z) dz \\ &= \frac{1}{k} \int \tilde{p}(z) dz \end{aligned}$$

So k should be as small as possible subject to the limitation that $kq(z) > \tilde{p}(z)$ for all $z \rightarrow$ the proposal distribution q should be close to p

Rejection sampling does not work well with high dimensional spaces, as the number of rejected samples increases exponentially with dimensions.

Importance sampling

Goal: Approximate expectations directly



Reuse the idea of the proposal distribution $q(\mathbf{z})$ and express the expectation in the form of a finite sum over samples $\{\mathbf{z}^{(l)}\}$ drawn from $q(\mathbf{z})$

- No rejection of samples
- Correct bias introduced by sampling from the wrong distribution by weighting samples

Importance sampling

$$\begin{aligned} E[f] &= \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} \\ &= \int f(\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})d\mathbf{z} \end{aligned}$$

Importance sampling

$$
\begin{aligned}
E[f] &= \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} \\
&= \int f(\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})d\mathbf{z} \\
&\simeq \frac{1}{L}\sum_{l=1}^{L}\frac{p(\mathbf{z}^{(l)})}{q(\mathbf{z}^{(l)})}f(\mathbf{z}^{(l)})
\end{aligned}
$$

Note we can see this as sampling $f(\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}$ from the $q(\mathbf{z})$ distribution.

Importance sampling

$$
\begin{aligned}
E[f] &= \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} \\
&= \int f(\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})d\mathbf{z} \\
&\simeq \frac{1}{L}\sum_{l=1}^{L}\frac{p(\mathbf{z}^{(l)})}{q(\mathbf{z}^{(l)})}f(\mathbf{z}^{(l)})
\end{aligned}
$$

$w_l = p(\mathbf{z}^{(l)})/q(\mathbf{z}^{(l)})$ are known as **importance weights**

Importance sampling

- Success depends on how well the proposal distribution $q(\mathbf{x})$ matches the desired distribution $p(\mathbf{x})$.
- Must have $p(z) > 0 \Rightarrow q(z) > 0$.

Sampling and the EM algorithm

If the expectation step of the EM algorithm cannot be performed analytically, we may instead approximate it with sampling.

$$Q(\theta, \theta^{\text{old}}) = \int p(\mathbf{Z}|\mathbf{X}, \theta^{\text{old}}) \ln p(\mathbf{Z}, \mathbf{X}|\theta) d\mathbf{Z}$$

Estimate integral with sum over samples drawn from the current estimate of the poster distribution $p(\mathbf{Z}^{(l)}, \mathbf{X}|\theta)$

$$Q(\theta, \theta^{\text{old}}) \simeq \frac{1}{L} \sum_{l=1}^{L} \ln p(\mathbf{Z}^{(l)}, \mathbf{X}|\theta)$$

Python libraries

- pgmpy - Python library for Probabilistic Graphical Models
  - Directed, undirected, factor graphs, etc...
  - Sampling methods, Belief Propagation, Max-Product, etc...
  - And much more.
- Pyro - Python/PyTorch based library for Deep Universal Probabilistic Programming
  - Unifying deep learning and graphical models
  - Variational inference, sampling

Questions?