Winter School: Mathematics for Data Modelling

Lecture 2: Inference and Propagation Algorithms

Zoubin Ghahramani
zoubin@eng.cam.ac.uk
http://learning.eng.cam.ac.uk/zoubin/

Department of Engineering
University of Cambridge, UK

Machine Learning Department
Carnegie Mellon University, USA

Sheffield, January 2008
Inference in a graphical model

Consider the following graph:

\[
p(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|B, C)p(E|C, D)
\]

**Inference**: evaluate the probability distribution over some set of variables, given the values of another set of variables.

For example, how can we compute \( P(A|C = c) \)? Assume each variable is binary.

**Naive method:**

\[
p(A, C = c) = \sum_{B,D,E} p(A, B, C = c, D, E) \quad [16 \text{ operations}]
\]

\[
p(C = c) = \sum_A p(A, C = c) \quad [2 \text{ operations}]
\]

\[
p(A|C = c) = \frac{p(A, C = c)}{p(C = c)} \quad [2 \text{ operations}]
\]

Total: \( 16 + 2 + 2 = 20 \) operations
Inference in a graphical model

Consider the following graph: which represents:

\[ p(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|B, C)p(E|C, D) \]

Computing \( p(A|C = c) \).

**More efficient method:**

\[
p(A, C = c) = \sum_{B,D,E} p(A)p(B)p(C = c|A, B)p(D|B, C = c)p(E|C = c, D)
\]

\[
= \sum_{B} p(A)p(B)p(C = c|A, B) \sum_{D} p(D|B, C = c) \sum_{E} p(E|C = c, D)
\]

\[
= \sum_{B} p(A)p(B)p(C = c|A, B) \quad \text{[4 operations]}
\]

**Total: 4+2+2 = 8 operations**

Belief propagation methods use the conditional independence relationships in a graph to do efficient inference (for singly connected graphs, exponential gains in efficiency!).
Belief Propagation (in singly connected DAGs)

**Definition:** A DAG is *singly connected* if its underlying undirected graph is a tree, *i.e.* there is only one undirected path between any two nodes.

![Singly Connected DAG](image)

**Goal:** For some node $X$ we want to compute $p(X|e)$ given evidence $e$.

Since we are considering singly connected graphs:

- every node $X$ divides the evidence into **upstream** $e^+_X$ and **downstream** $e^-_X$
- every edge $X \rightarrow Y$ divides the evidence into **upstream** $e^+_{XY}$ and **downstream** $e^-_{XY}$. 
Three key ideas behind Belief Propagation

Idea 1: The probability of a variable $X$ can be found by combining upstream and downstream evidence:

$$p(X|e) = \frac{p(X,e)}{p(e)} = \frac{p(X,e^+_X,e^-_X)}{p(e^+_X,e^-_X)} \propto p(X|e^+_X) \times \underbrace{p(e^-_X|X,e^+_X)}_{X \text{ d-separates } e^-_X \text{ from } e^+_X}$$

$$= p(X|e^+_X)p(e^-_X|X) = \pi(X)\lambda(X)$$

Idea 2: The upstream and downstream evidence can be computed via a local message passing algorithm between the nodes in the graph.

Idea 3: “Don’t send back to a node (any part of) the message it sent to you!”
Belief Propagation

\[ \text{top-down upstream evidence: (message } U_i \text{ sends to } X) \]
\[ \pi_X(U_i) = p(U_i|e_{U_iX}^+) \]

\[ \text{bottom-up downstream evidence: (message } Y_j \text{ sends to } X) \]
\[ \lambda_{Y_j}(X) = p(e_{XY_j}^-|X) \]

To update the probability of \( X \) given the evidence:

\[ \text{BEL}(X) = p(X|e) = \frac{1}{Z} \lambda(X) \pi(X) \]
\[ \lambda(X) = \prod_j \lambda_{Y_j}(X) \]
\[ \pi(X) = \sum_{U_1 \cdots U_n} p(X|U_1, \ldots, U_n) \prod_i \pi_X(U_i) \]
Belief Propagation (cont.)

top-down upstream evidence: (message $U_i$ sends to $X$)

$$\pi_X(U_i) = p(U_i|e^+_{U_iX})$$

bottom-up downstream evidence: (message $Y_j$ sends to $X$)

$$\lambda_{Y_j}(X) = p(e^-_{XY_j}|X)$$

Bottom-up propagation, message $X$ sends to $U_i$:

$$\lambda_X(U_i) = \sum_X \lambda(X) \sum_{U_k:k\neq i} p(X|U_1,\ldots,U_n) \prod_{k\neq i} \pi_X(U_k)$$

Top-down propagation, message $X$ sends to $Y_j$:

$$\pi_{Y_j}(X) = \frac{1}{Z} \left[ \prod_{k\neq j} \lambda_{Y_k}(X) \right] \sum_{U_1\ldots U_n} p(X|U_1,\ldots,U_n) \prod_i \pi_X(U_i) = \frac{1}{Z} \frac{\text{BEL}(X)}{\lambda_{Y_j}(X)}$$

$Z$ is the normaliser ensuring $\sum_X \pi_{Y_j}(X) = 1$
Factor graph propagation
In a factor graph, the joint probability distribution is written as a product of factors. Consider a vector of variables $\mathbf{x} = (x_1, \ldots, x_n)$

$$p(\mathbf{x}) = p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_j f_j(\mathbf{x}_{S_j})$$

where $Z$ is the normalisation constant, $S_j$ denotes the subset of $\{1, \ldots, n\}$ which participate in factor $f_j$ and $\mathbf{x}_{S_j} = \{x_i : i \in S_j\}$.

**variables nodes**: we draw open circles for each variable $x_i$ in the distribution.

**factor nodes**: we draw filled dots for each factor $f_j$ in the distribution.
Propagation in Factor Graphs

Let $n(x)$ denote the set of factor nodes that are neighbors of $x$. Let $n(f)$ denote the set of variable nodes that are neighbors of $f$.

We can compute probabilities in a factor graph by propagating messages from variable nodes to factor nodes and vice versa.

**message from variable $x$ to factor $f$:**

$$
\mu_{x \rightarrow f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \rightarrow x}(x)
$$

**message from factor $f$ to variable $x$:**

$$
\mu_{f \rightarrow x}(x) = \sum_{x \setminus x} \left( f(x) \prod_{y \in n(f) \setminus \{x\}} \mu_{y \rightarrow f}(y) \right)
$$
Propagation in Factor Graphs

\( \mathbb{n}(x) \) denotes the set of factor nodes that are neighbors of \( x \).
\( \mathbb{n}(f) \) denotes the set of variable nodes that are neighbors of \( f \).

**message from variable** \( x \) **to factor** \( f \):

\[
\mu_{x \rightarrow f}(x) = \prod_{h \in \mathbb{n}(x) \setminus \{ f \}} \mu_{h \rightarrow x}(x)
\]

**message from factor** \( f \) **to variable** \( x \):

\[
\mu_{f \rightarrow x}(x) = \sum_{x \setminus x} \left( f(x) \prod_{y \in \mathbb{n}(f) \setminus \{ x \}} \mu_{y \rightarrow f}(y) \right)
\]

If a variable has only one factor as a neighbor, it can initiate message propagation.

Once a variable has received all messages from its neighboring factor nodes we can compute the probability of that variable by multiplying all the messages and renormalising:

\[
p(x) \propto \prod_{h \in \mathbb{n}(x)} \mu_{h \rightarrow x}(x)
\]
initialise all messages to be 1

an example schedule of messages resulting in computing $p(x_4)$:

<table>
<thead>
<tr>
<th>message direction</th>
<th>message value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \to f_1$</td>
<td>$1(x_1)$</td>
</tr>
<tr>
<td>$x_3 \to f_2$</td>
<td>$1(x_3)$</td>
</tr>
<tr>
<td>$f_1 \to x_2$</td>
<td>$\sum x_1 f_1(x_1, x_2)1(x_1)$</td>
</tr>
<tr>
<td>$f_2 \to x_2$</td>
<td>$\sum x_3 f_2(x_3, x_2)1(x_3)$</td>
</tr>
<tr>
<td>$x_2 \to f_3$</td>
<td>$\left( \sum x_1 f_1(x_1, x_2) \right) \left( \sum x_3 f_2(x_3, x_2) \right)$</td>
</tr>
<tr>
<td>$f_3 \to x_4$</td>
<td>$\sum x_2 f_3(x_2, x_4) \left( \sum x_1 f_1(x_1, x_2) \right) \left( \sum x_3 f_2(x_3, x_2) \right)$</td>
</tr>
</tbody>
</table>
initialise all messages to be 1

an example schedule of messages resulting in computing \( p(x_4|x_1 = a) \):

<table>
<thead>
<tr>
<th>message direction</th>
<th>message value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 \rightarrow f_1 )</td>
<td>( \delta(x_1 = a) )</td>
</tr>
<tr>
<td>( x_3 \rightarrow f_2 )</td>
<td>( 1(x_3) )</td>
</tr>
<tr>
<td>( f_1 \rightarrow x_2 )</td>
<td>( \sum x_1 f_1(x_1, x_2)\delta(x_1 = a) = f_1(x_1 = a, x_2) )</td>
</tr>
<tr>
<td>( f_2 \rightarrow x_2 )</td>
<td>( \sum x_3 f_2(x_3, x_2)1(x_3) )</td>
</tr>
<tr>
<td>( x_2 \rightarrow f_3 )</td>
<td>( f_1(x_1 = a, x_2) \left( \sum x_3 f_2(x_3, x_2) \right) )</td>
</tr>
<tr>
<td>( f_3 \rightarrow x_4 )</td>
<td>( \sum x_2 f_3(x_2, x_4)f_1(x_1 = a, x_2) \left( \sum x_3 f_2(x_3, x_2) \right) )</td>
</tr>
</tbody>
</table>
Inference in Hidden markov models and Linear Gaussian state-space models

\[ p(X_1,\ldots,T, Y_1,\ldots,T) = p(X_1)p(Y_1|X_1) \prod_{t=2}^{T} [p(X_t|X_{t-1})p(Y_t|X_t)] \]

- In HMMs, the states \( X_t \) are discrete.
- In linear Gaussian SSMs, the states are real Gaussian vectors.
- Both HMMs and SSMs can be represented as singly connected DAGs.
- The forward–backward algorithm in hidden Markov models (HMMs), and the Kalman smoothing algorithm in SSMs are both instances of belief propagation / factor graph propagation.
Inference in multiply connected DAGs

The Junction Tree algorithm: Form an undirected graph from your directed graph such that no additional conditional independence relationships have been created (this step is called “moralization”). Lump variables in cliques together and form a tree of cliques—this may require a nasty step called “triangulation”. Do inference in this tree of cliques.

Cutset Conditioning: or “reasoning by assumptions”. Find a small set of variables which, if they were given (i.e. known) would render the remaining graph singly connected. For each value of these variables run belief propagation on the singly connected network. Average the resulting beliefs with the appropriate weights (given by normalizing constants).

Loopy Belief Propagation: just use BP although there are loops. In this case the terms “upstream” and “downstream” are not clearly defined. No guarantee of convergence, except for certain special graphs, but often works well in practice (c.f. “turbo-decoding” for error-correcting codes).
Summary

- inference consists of the problem of computing $p(\text{variables of interest}|\text{observed variables})$

- for singly connected DAGs, belief propagation solves this problem exactly.

- for factor graphs, the analogous algorithm is factor graph propagation.

- well-known algorithms such as Kalman smoothing and forward-backward are special cases these general propagation algorithms.

- for multiply connected graphs, the junction tree algorithm solves the exact inference problem, but can be very slow (exponential in the cardinality of the largest clique).

- one approximate inference algorithm is “loopy belief propagation”—we will see other approximate inference algorithms in a later lecture.
Appendix: The Junction Tree Algorithm
The Junction Tree Algorithm 1

starting with a DAG...
moralize by marrying the parents of each node
remove edge directions
this results in an undirected graph with no additional C.I. relations
The Junction Tree Algorithm 3

triangulate so that there is no loop of length $> 3$ without a chord.
this is necessary so that the final junction tree satisfies the running intersection property.
The Junction Tree Algorithm 4

find cliques of the moralized, triangulated graph
The Junction Tree Algorithm

- **Form junction tree**: tree of (overlapping) sets of variables

- **Running intersection property** means that if a variable appears in more than one clique (e.g. $C$), it appears in all intermediate cliques in the tree.

- The junction tree propagation algorithm ensures that neighboring cliques have consistent probability distribution.

- Local consistency $\rightarrow$ global consistency
Appendix:

Fluffy and Moby: A Belief Propagation Demo
1. Model Structure

Fluffy = pet cat
Moby = pet fish

Fluffy ate Moby
Moby is dead

Fluffy is hungry
Fluffy has flu
Fluffy's food bowl is full
Fluffy has high temperature
2. Model Parameters

Fluffy = pet cat
Moby = pet fish

P(x1=1)=0.01
Fluffy ate Moby
P(x1=1)=0.01
Fluffy has flu

P(x3=1|x1=0)=0.1
P(x3=1|x1=1)=1.0
Moby is dead

P(x4=1|x1,x2) x1, x2→x4=1
0 0 0.9
0 1 0.1
1 0 0.1
1 1 0.01
Fluffy is hungry

P(x6=1|x4=0)=0.9
P(x6=1|x4=1)=0.1
Fluffy's food bowl is full

P(x5=1|x2=0)=0.1
P(x5=1|x2=1)=0.9
Fluffy has high temperature
3. Propagating Evidence

1. Observe “Moby is dead”, i.e. $x_3 = 1$

2. Send $\lambda_{x_3}(x_1) \equiv p(e_{x_1 \rightarrow x_3}^\sim | x_1) = \begin{bmatrix} 0.1 \\ 1.0 \end{bmatrix}$ message $x_3 \rightarrow x_1$

3. $BEL(x_1 | x_3 = 1) = \frac{1}{Z} \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix} \odot \begin{bmatrix} 0.1 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 0.91 \\ 0.09 \end{bmatrix}$
4. Propagating Evidence

4. Send $\pi_{x_4}(x_1) \equiv p(x_1|e_{x_1 \rightarrow x_4}^+) = \begin{bmatrix} 0.91 \\ 0.09 \end{bmatrix}$

5. Send $\pi_{x_4}(x_2) \equiv p(x_2|e_{x_2 \rightarrow x_4}^+) = p(x_2) = \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix}$ from $x_2 \rightarrow x_4$.

6. Compute $\pi(x_4) \equiv p(x_4|e_{x_4}^+) = \sum_{x_1,x_2} p(x_4|x_1,x_2)\pi_{x_4}(x_1)\pi_{x_4}(x_2) = \begin{bmatrix} 0.18 \\ 0.82 \end{bmatrix}$

7. $BEL(x_4|x_3 = 1) = \begin{bmatrix} 0.18 \\ 0.82 \end{bmatrix}$, whereas before observing $x_3 = 1$, $BEL(x_4) = \begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix}$. 
5. Propagating Evidence

8. Observe “Fluffy’s Food Bowl is Full” \( x_6 = 1 \)!

9. Send \( \lambda_{x_6}(x_4) = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} \) message \( x_6 \rightarrow x_4 \)

10. \( BE\!L(x_4|x_3 = 1, x_6 = 1) = \frac{1}{Z} \begin{bmatrix} 0.18 \\ 0.82 \end{bmatrix} \odot \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.66 \\ 0.34 \end{bmatrix} \)

11. Send \( \lambda_{x_4}(x_1) = \sum_{x_4} \lambda_{x_6}(x_4) \sum_{x_2} p(x_4|x_1, x_2) \pi_{x_4}(x_2) = \begin{bmatrix} 0.19 \\ 0.82 \end{bmatrix} \)

12. \( BE\!L(x_1|x_3 = 1, x_6 = 1) = \frac{1}{Z} \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix} \odot \begin{bmatrix} 0.1 \\ 1.0 \end{bmatrix} \odot \begin{bmatrix} 0.19 \\ 0.82 \end{bmatrix} = \begin{bmatrix} 0.70 \\ 0.30 \end{bmatrix} \Rightarrow \text{Fluffy still innocent!} \)
Appendix: Understanding BP equations

\[ p(X|e) = \text{BEL}(X) = \frac{1}{Z} \lambda(X) \pi(X) = p(e_X^-|X)p(X|e_X^+) \]  
(1)

\[ p(e_X^-|X) = \lambda(X) = \prod_j \lambda_{Y_j}(X) = \prod_j p(e_{XY_j}^-|X) \]  
(2)

\[ p(X|e_X^+) = \pi(X) = \sum_{U_1 \ldots U_n} p(X|U_1, \ldots, U_n) \prod_i \pi_X(U_i) \]
\[ = \sum_{U_1 \ldots U_n} p(X|U_1, \ldots, U_n) \prod_i p(U_i|e_{U_iX}^+) \]  
(3)

\[ Z \text{ is a normalization constant.} \]

All equations follow from the conditional independencies in the graph.
• eliminating observed variables

If a variable $x_i$ is observed, i.e. its value is given, then it is a constant in all factor that include $x_i$.

We can eliminate $x_i$ from the graph by removing the corresponding node and modifying all neighboring factors to treat it as a constant.
Elimination Rules for Factor Graphs

• eliminating hidden variables

If a variable $x_i$ is hidden and we are not interested in it we can eliminate it from the graph by summing over all its values.

$$
\sum_{x_i} p(x) = \frac{1}{Z} \sum_{x_i} \prod_j f_j(x_{S_j})
$$

$$
= \frac{1}{Z} \prod_{j \notin n(x_i)} f_j(x_{S_j}) \left( \sum_{x_i} \prod_{k \in n(x_i)} f_k(x_{S_k}) \right)
$$

$$
= \frac{1}{Z} \prod_{j \notin n(x_i)} f_j(x_{S_j}) f_{\text{new}}(x_{S_{\text{new}}})
$$

where $f_{\text{new}}(x_{S_{\text{new}}}) = \sum_{x_i} \prod_{k \in n(x_i)} f_k(x_{S_k})$ and $S_{\text{new}} = \bigcup_{k \in n(x_i)} S_k \setminus \{i\}$.

This causes all its neighboring factor nodes to merge into one new factor node.