Machine Learning Summer School

Lecture 1: Introduction to Graphical Models

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

August 2009
Three main kinds of graphical models

- Nodes correspond to random variables
- Edges represent statistical dependencies between the variables
Why do we need graphical models?

- Graphs are an **intuitive** way of representing and visualising the relationships between many variables. (Examples: family trees, electric circuit diagrams, neural networks)

- A graph allows us to abstract out the **conditional independence** relationships between the variables from the details of their parametric forms. Thus we can answer questions like: “Is $A$ dependent on $B$ given that we know the value of $C$?” just by looking at the graph.

- Graphical models allow us to define general **message-passing algorithms** that implement probabilistic inference efficiently. Thus we can answer queries like “What is $p(A|C = c)$?” without enumerating all settings of all variables in the model.

Graphical models = statistics × graph theory × computer science.
Conditional Independence

**Conditional Independence:**

$$X \perp Y | V \iff p(X|Y, V) = p(X|V)$$

when $p(Y, V) > 0$. Also

$$X \perp Y | V \iff p(X, Y|V) = p(X|V) p(Y|V)$$

In general we can think of conditional independence between sets of variables:

$$\mathcal{X} \perp \mathcal{Y} | V \iff p(\mathcal{X}, \mathcal{Y}|V) = p(\mathcal{X}|V) p(\mathcal{Y}|V)$$

**Marginal Independence:**

$$X \perp Y \iff X \perp Y | \emptyset \iff p(X, Y) = p(X) p(Y)$$
Conditional and Marginal Independence (Examples)

- Amount of Speeding Fine $\perp$ Type of Car $|\,$ Speed
- Lung Cancer $\perp$ Yellow Teeth $|\,$ Smoking
- $(\text{Position}, \text{Velocity})_{t+1} \perp (\text{Position}, \text{Velocity})_{t-1} | (\text{Position}, \text{Velocity})_t, \text{Acceleration}_t$
- Child’s Genes $\perp$ Grandparents’ Genes $|\,$ Parents’ Genes
- Ability of Team A $\perp$ Ability of Team B
- not ( Ability of Team A $\perp$ Ability of Team B $|\,$ Outcome of A vs B Game )
Factor Graphs

Two types of nodes:

- The circles in a factor graph represent random variables (e.g. $A$).
- The filled dots represent factors in the joint distribution (e.g. $g_1(\cdot)$).

(a) $p(A, B, C, D, E) = \frac{1}{Z} g_1(A, C) g_2(B, C, D) g_3(C, D, E)$

(b) $p(A, B, C, D, E) = \frac{1}{Z} g_1(A, C) g_2(B, C) g_3(C, D) g_4(B, D) g_5(C, E) g_6(D, E)$

The $g_i$ are non-negative functions of their arguments, and $Z$ is a normalization constant.

E.g. in (a), if all variables are discrete and take values in $A \times B \times C \times D \times E$:

$$Z = \sum_{a \in A} \sum_{b \in B} \sum_{c \in C} \sum_{d \in D} \sum_{e \in E} g_1(A = a, C = c) g_2(B = b, C = c, D = d) g_3(C = c, D = d, E = e)$$

Two nodes are neighbors if they share a common factor.
The circles in a factor graph represent random variables. The filled dots represent factors in the joint distribution.

(a) \[ p(A, B, C, D, E) = \frac{1}{Z} g_1(A, C) g_2(B, C, D) g_3(C, D, E) \]

(b) \[ p(A, B, C, D, E) = \frac{1}{Z} g_1(A, C) g_2(B, C) g_3(C, D) g_4(B, D) g_5(C, E) g_6(D, E) \]

Two nodes are neighbors if they share a common factor.

**Definition:** A *path* is a sequence of neighboring nodes.

**Fact:** \( X \perp \perp Y \mid \mathcal{V} \) if every path between \( X \) and \( Y \) contains some node \( V \in \mathcal{V} \)

**Corollary:** Given the neighbors of \( X \), the variable \( X \) is conditionally independent of all other variables: \( X \perp \perp Y \mid \text{ne}(X), \ \forall Y \notin \{X\} \cup \text{ne}(X) \)
Proving Conditional Independence

Assume:
\[ p(X, Y, V) = \frac{1}{Z} g_1(X, V) g_2(Y, V) \]  \hspace{1cm} (1)

We want to show conditional independence:
\[ X \perp \!\!\!\perp Y \mid V \iff p(X \mid Y, V) = p(X \mid V) \]  \hspace{1cm} (2)

Summing (1) over \( X \) we get:
\[ p(Y, V) = \frac{1}{Z} \left[ \sum_X g_1(X, V) \right] g_2(Y, V) \]  \hspace{1cm} (3)

Dividing (1) by (3) we get:
\[ p(X \mid Y, V) = \frac{g_1(X, V)}{\sum_X g_1(X, V)} \]  \hspace{1cm} (4)

Since the rhs. of (4) doesn't depend on \( Y \), it follows that \( X \) is independent of \( Y \) given \( V \). Therefore factorization (1) implies conditional independence (2).
Undirected Graphical Models

In an Undirected Graphical Model, the joint probability over all variables can be written in a factored form:

\[ p(x) = \frac{1}{Z} \prod_j g_j(x_{C_j}) \]

where \( x = (x_1, \ldots, x_K) \), and

\[ C_j \subseteq \{1, \ldots, K\} \]

are subsets of the set of all variables, and \( x_S \equiv (x_k : k \in S) \).

**Graph Specification:** Create a node for each variable. Connect nodes \( i \) and \( k \) if there exists a set \( C_j \) such that both \( i \in C_j \) and \( k \in C_j \). These sets form the *cliques* of the graph (fully connected subgraphs).

**Note:** Undirected Graphical Models are also called *Markov Networks*.

Very similar to factor graphs.
Undirected Graphical Models

\[ p(A, B, C, D, E) = \frac{1}{Z} g_1(A, C)g_2(B, C, D)g_3(C, D, E) \]

**Fact:** \( X \perp Y | \mathcal{V} \) if every path between \( X \) and \( Y \) contains some node \( V \in \mathcal{V} \)

**Corollary:** Given the neighbors of \( X \), the variable \( X \) is conditionally independent of all other variables: \( X \perp Y | \text{ne}(X), \ \forall Y \notin \{X\} \cup \text{ne}(X) \)

**Markov Blanket:** \( \mathcal{V} \) is a Markov Blanket for \( X \) iff \( X \perp Y | \mathcal{V} \) for all \( Y \notin \{X \cup \mathcal{V}\} \).

**Markov Boundary:** minimal Markov Blanket \( \equiv \text{ne}(X) \) for undirected and factor graphs
Comparing Undirected Graphs and Factor Graphs

All nodes in (a), (b), and (c) have exactly the same neighbors and therefore these three graphs represent exactly the same conditional independence relationships.

(c) also represents the fact that the probability factors into a product of pairwise functions.

Consider the case where each variable is discrete and can take on $K$ possible values. Then the functions in (a) and (b) are tables with $\mathcal{O}(K^3)$ cells, whereas in (c) they are $\mathcal{O}(K^2)$. 
Problems with Undirected Graphs and Factor Graphs

In UGs and FGs, many useful independencies are not represented—two variables are connected merely because some other variable depends on them:

This highlights the difference between marginal independence and conditional independence.

\( R \) and \( S \) are marginally independent (i.e. given nothing), but they are conditionally dependent given \( G \).

“Explaining Away”: Observing that the sprinkler is on would explain away the observation that the ground was wet, making it less probable that it rained.
A DAG Model / Bayesian network corresponds to a factorization of the joint probability distribution:

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

In general:

\[ p(X_1, \ldots, X_n) = \prod_{i=1}^{n} p(X_i|X_{pa(i)}) \]

where pa(i) are the parents of node i.

---

1“Bayesian networks” can and often are learned using non-Bayesian (i.e. frequentist) methods; Bayesian networks (i.e. DAGs) do not require parameter or structure learning using Bayesian methods. Also called “belief networks”.
Directed Acyclic Graphical Models (Bayesian Networks)

**Semantics:** $X \perp Y \mid \mathcal{V}$ if $\mathcal{V}$ d-separates $X$ from $Y$.\(^2\)

**Definition:** $\mathcal{V}$ d-separates $X$ from $Y$ if every undirected path\(^3\) between $X$ and $Y$ is blocked by $\mathcal{V}$. A path is blocked by $\mathcal{V}$ if there is a node $W$ on the path such that either:

1. $W$ has converging arrows along the path $(\rightarrow W \leftarrow)$\(^4\) and neither $W$ nor its descendants are observed (in $\mathcal{V}$), or
2. $W$ does not have converging arrows along the path $(\rightarrow W \rightarrow$ or $\leftarrow W \rightarrow)$ and $W$ is observed ($W \in \mathcal{V}$).

**Corollary:** Markov Boundary for $X$: $\{\text{parents}(X) \cup \text{children}(X) \cup \text{parents-of-children}(X)\}$.\(^2\)

---

\(^2\)See also the “Bayes Ball” algorithm in the Appendix

\(^3\)An undirected path ignores the direction of the edges.

\(^4\)Note that converging arrows along the path only refers to what happens on that path. Also called a collider.
Examples of D-Separation in DAGs

Examples:

- $A \perp\!\!\!\!\!\!\perp B$ since $A \rightarrow C \leftarrow B$ is blocked \(^1\) by $C$, $A \rightarrow C \rightarrow D \leftarrow B$ is blocked \(^1\) by $D$, etc.

- not $(A \perp\!\!\!\!\!\!\perp B | C)$ since $A \rightarrow C \leftarrow B$ is not blocked.

- $A \perp\!\!\!\!\!\!\perp D | \{B, C\}$ since $A \rightarrow C \rightarrow D$ is blocked \(^2\) by $C$, $A \rightarrow C \leftarrow B \rightarrow D$ is blocked \(^2\) by $B$, and $A \rightarrow C \rightarrow E \leftarrow D$ is blocked \(^2\) by $C$.

- not $(A \perp\!\!\!\!\!\!\perp B | E)$ since $A \rightarrow C \leftarrow B$ is not blocked.

Note that it is the *absence of edges* that conveys conditional independence.
From Directed Trees to Undirected Trees

$$p(x_1, x_2, \ldots, x_7) = p(x_3)p(x_1|x_3)p(x_2|x_3)p(x_4|x_3)p(x_5|x_4)p(x_6|x_4)p(x_7|x_4)$$

$$= \frac{p(x_1, x_3)p(x_2, x_3)p(x_3, x_4)p(x_4, x_5)p(x_4, x_6)p(x_4, x_7)}{p(x_3)p(x_3)p(x_4)p(x_4)p(x_4)}$$

$$= \frac{\text{product of cliques}}{\text{product of clique intersections}}$$

$$= g_1(x_1, x_3)g_2(x_2, x_3)g_3(x_3, x_4)g_4(x_4, x_5)g_5(x_4, x_6)g_6(x_4, x_7)$$

$$= \prod_{i} g_i(C_i)$$

Any directed tree can be converted into an undirected tree representing the same conditional independence relationships, and vice versa.
Consider the following simple model. A data set of $N$ points is generated i.i.d. from a Gaussian with mean $\mu$ and standard deviation $\sigma$:

$$p(x_1, \ldots, x_N, \mu, \sigma) = p(\mu)p(\sigma) \prod_{n=1}^{N} p(x_n | \mu, \sigma)$$

This can be represented graphically as follows:
Expressive Power of Directed and Undirected Graphs

No Directed Graph (Bayesian network) can represent these and only these independencies.

No matter how we direct the arrows there will always be two non-adjacent parents sharing a common child $\equiv$ dependence in Directed Graph but independence in Undirected Graph.

No Undirected Graph or Factor Graph can represent these and only these independencies.

Directed graphs are better at expressing causal generative models, undirected graphs are better at representing soft constraints between variables.
Summary

• Three kinds of graphical models: directed, undirected, factor (there are other important classes, e.g. directed mixed graphs)

• Marginal and conditional independence

• Markov boundaries and d-separation

• Differences between directed and undirected graphs.

• Next lectures:
  – exact inference and propagation algorithm
  – parameter and structure learning in graphs
  – nonparametric approaches to graph learning
Appendix: Some Examples of Directed Graphical Models

- Factor analysis
- Probabilistic PCA
- Hidden Markov models
- Linear dynamical systems
- Switching state-space models
Appendix: Examples of Undirected Graphical Models

- Markov Random Fields (used in Computer Vision)

- Exponential Language Models (used in Speech and Language Modelling)

\[ p(s) = \frac{1}{Z} p_0(s) \exp \left\{ \sum_i \lambda_i f_i(s) \right\} \]

- Products of Experts (widely applicable)

\[ p(x) = \frac{1}{Z} \prod_j p_j(x | \theta_j) \]

- Boltzmann Machines (a kind of Neural Network/Ising Model)
Appendix: Clique Potentials and Undirected Graphs

**Definition:** a *clique* is a fully connected subgraph. By clique we usually mean maximal clique (i.e. not contained within another clique)

$C_i$ denotes the set of variables in the $i^{th}$ clique.

$$p(x_1, \ldots, x_K) = \frac{1}{Z} \prod_i g_i(x_{C_i})$$

where $Z = \sum x_1 \cdots x_K \prod_i g_i(x_{C_i})$ is the normalization.

Associated with each clique $C_i$ is a non-negative function $g_i(x_{C_i})$ which measures “compatibility” between settings of the variables.

**Example:** Let $C_1 = \{A, C\}$, $A \in \{0, 1\}$, $C \in \{0, 1\}$

What does this mean?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$g_1(A, C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Appendix: Hammersley–Clifford Theorem (1971)

**Theorem:** A probability function \( p \) formed by a normalized product of positive functions on cliques of \( G \) is a Markov Field relative to \( G \).

**Definition:** The distribution \( p \) is a *Markov Field relative to* \( G \) if all conditional independence relations represented by \( G \) are true of \( p \).

\( G \) represents the following CI relations: If \( V \in \mathcal{V} \) lies on *all* paths between \( X \) and \( Y \) in \( G \), then \( X \perp \!\!\!\perp Y \mid \mathcal{V} \).

**Proof:** We need to show that if \( p \) is a product of functions on cliques of \( G \) then a variable is conditionally independent of its non-neighbors in \( G \) given its neighbors in \( G \). That is: \( \text{ne}(x_\ell) \) is a Markov Blanket for \( x_\ell \). Let \( x_m \notin \{x_\ell \cup \text{ne}(x_\ell)\} \)

\[
p(x_\ell, x_m, \ldots) = \frac{1}{Z} \prod_i g_i(x_{C_i}) = \frac{1}{Z} \prod_{i: \ell \in C_i} g_i(x_{C_i}) \prod_{j: \ell \notin C_j} g_j(x_{C_j})
\]

\[
= \frac{1}{Z'} f_1(x_\ell, \text{ne}(x_\ell)) f_2(\text{ne}(x_\ell), x_m) = \frac{1}{Z''} p(x_\ell \mid \text{ne}(x_\ell)) p(x_m \mid \text{ne}(x_\ell))
\]

It follows that: \( p(x_\ell, x_m \mid \text{ne}(x_\ell)) = p(x_\ell \mid \text{ne}(x_\ell)) p(x_m \mid \text{ne}(x_\ell)) \iff x_\ell \perp \!\!\!\perp x_m \mid \text{ne}(x_\ell) \).
Appendix: The “Bayes-ball” algorithm

Game: can you get a ball from $X$ to $Y$ without being blocked by $\mathcal{V}$?

Depending on the direction the ball came from and the type of node, the ball can pass through (from a parent to all children, from a child to all parents), bounce back (from any parent to all parents, or from any child to all children), or be blocked.

- An unobserved (hidden) node ($W \notin \mathcal{V}$) passes balls through but also bounces back balls from children.

- An observed (given) node ($W \in \mathcal{V}$) bounces back balls from parents but blocks balls from children.
Machine Learning Summer School

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

August 2009
Inference in a graphical model

Consider the following graph: which represents:

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

**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:**

\[
\begin{align*}
p(A, C = c) &= \sum_{B, D, E} p(A, B, C = c, D, E) \quad [16 \text{ terms}] \\
p(C = c) &= \sum_A p(A, C = c) \quad [2 \text{ terms}] \\
p(A|C = c) &= \frac{p(A, C = c)}{p(C = c)} \quad [2 \text{ terms}]
\end{align*}
\]

Total: \( 16 + 2 + 2 = 20 \) terms have to be computed and summed
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:**

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

\[
= \sum_B p(A)p(B)p(C = c|A, B) \sum_D p(D|B, C = c) \sum_E p(E|C = c, D)
\]

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

Total: \( 4+2+2 = 8 \) terms

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, *ie* there is only one undirected path between any two nodes.

**Goal:** For some node $X$ we want to compute conditional $p(X|e)$ given evidence (i.e. observed, visible variables) $e$.

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!”
Factor graph propagation

Algorithmically and implementationally, it’s often easier to convert directed and undirected graphs into factor graphs, and run factor graph propagation.

\[
p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2)
\equiv f_1(x_1, x_2) f_2(x_2, x_3) f_3(x_2, x_4)
\]

Singly connected vs Multiply connected factor graphs:
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 \( x \):**

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

where \( x \) are the variables that factor \( f \) depends on, and \( \sum_{x\setminus x} \) is a sum over all variables neighboring factor \( f \) except \( x \).
Propagation in Factor Graphs

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

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

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

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

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

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, one can compute the probability of that variable by multiplying all the messages and renormalising:

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

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>$x_1 \rightarrow f_1$</td>
<td>$1(x_1)$</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>$\sum x_1 f_1(x_1, x_2)1(x_1)$</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>$\left(\sum x_1 f_1(x_1, x_2)\right)\left(\sum x_3 f_2(x_3, x_2)\right)$</td>
</tr>
<tr>
<td>$f_3 \rightarrow x_4$</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>

where $1(x)$ is a constant uniform function of $x$
Propagation in Factor Graphs

\[ \begin{align*}
\delta(x_1 = a) \\
1(x_3) \\
\sum x_1 f_1(x_1, x_2) \delta(x_1 = a) &= f_1(x_1 = a, x_2) \\
\sum x_3 f_2(x_3, x_2) 1(x_3) \\
f_1(x_1 = a, x_2) \left( \sum x_3 f_2(x_3, x_2) \right) \\
\sum x_2 f_3(x_2, x_4) f_1(x_1 = a, x_2) \left( \sum x_3 f_2(x_3, x_2) \right)
\end{align*} \]

where \( \delta(x = a) \) is a delta function.
Elimination Rules for Factor Graphs

- **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.
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).
starting with a DAG...
moralize by marrying the parents of each node
remove edge directions
this results in an undirected graph with no additional conditional independence relations
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.
find cliques of the moralized, triangulated graph
The Junction Tree Algorithm 5

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

- the **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
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: Belief Propagation

To update the probability of $X$ given the evidence:

$$\text{BEL}(X) = p(X|e) = \frac{1}{Z} \lambda(X) \pi(X)$$

$$\lambda(X) = \prod_j \lambda_{Y_j}(X)$$

$$\pi(X) = \sum_{U_1 \cdots U_n} p(X|U_1, \ldots, U_n) \prod_i \pi_X(U_i)$$

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

$$\pi_X(U_i) = p(U_i|e_{U_i}^+X)$$

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

$$\lambda_{Y_j}(X) = p(e_{XY_j}^-|X)$$
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 $\sum_X \pi_{Y_j}(X) = 1$
Appendix:

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

Fluffy = pet cat
Moby = pet fish

\[ y = \text{pet cat} \]
\[ Moby = \text{pet fish} \]

Fluffy ate Moby

Fluffy's food bowl is full

Fluffy has flu

Fluffy is hungry

Fluffy has high temperature

Moby is dead

\[ X_1 \]
\[ X_2 \]
\[ X_3 \]
\[ X_4 \]
\[ X_5 \]
\[ X_6 \]
2. Model Parameters

Fluffy = pet cat
Moby = pet fish

\[ P(x_1 = 1) = 0.01 \]
Fluffy ate Moby

\[ P(x_3 = 1 | x_1 = 0) = 0.1 \]
\[ P(x_3 = 1 | x_1 = 1) = 1.0 \]
Moby is dead

\[ P(x_5 = 1 | x_2 = 0) = 0.1 \]
\[ P(x_5 = 1 | x_2 = 1) = 0.9 \]
Fluffy has high temperature

\[ P(x_6 = 1 | x_4 = 0) = 0.9 \]
\[ P(x_6 = 1 | x_4 = 1) = 0.1 \]
Fluffy's food bowl is full

\[ P(x_6 = 1 | x_4 = 1) = 0.01 \]
Fluffy has flu

\[ P(x_4 = 1 | x_1, x_2) \]
\[ x_1, x_2 \rightarrow x_4 = 1 \]
\[ \begin{array}{ccc}
0 & 0 & 0.9 \\
0 & 1 & 0.1 \\
1 & 0 & 0.1 \\
1 & 1 & 0.01 \\
\end{array} \]
1. Observe “Moby is dead”, i.e. $x_3 = 1$

2. Send $\lambda_{x_3}(x_1) \equiv p(e_{x_1 \rightarrow x_3} | x_1) = \begin{bmatrix} 0.1 \\ 1.0 \end{bmatrix}$ 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. \( \text{BEL}(x_4|x_3 = 1, x_6 = 1) = \frac{1}{Z} \begin{bmatrix} 0.18 \\ 0.82 \end{bmatrix} \odot \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.66 \\ 0.34 \end{bmatrix} \)

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. \( \text{BEL}(x_1|x_3 = 1, x_6 = 1) = \frac{1}{Z} \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix} \odot \begin{bmatrix} 0.1 \\ 1.0 \end{bmatrix} \odot \begin{bmatrix} 0.19 \\ 0.82 \end{bmatrix} = \begin{bmatrix} 0.70 \\ 0.30 \end{bmatrix} \Rightarrow \text{Fluffy still innocent!} \)
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)

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

\[ = \sum_{U_1...U_n} p(X|U_1,...,U_n) \prod_i p(U_i|e_{U_iX}^+) \] (4)

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

All equations follow from the conditional independencies in the graph.
Learning parameters

\[
p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)
\]

Assume each variable \( x_i \) is discrete and can take on \( K_i \) values.

The parameters of this model can be represented as 4 tables: \( \theta_1 \) has \( K_1 \) entries, \( \theta_2 \) has \( K_1 \times K_2 \) entries, etc.

These are called **conditional probability tables** (CPTs) with the following semantics:

\[
p(x_1 = k) = \theta_{1,k} \quad p(x_2 = k'|x_1 = k) = \theta_{2,k,k'}
\]

If node \( i \) has \( M \) parents, \( \theta_i \) can be represented either as an \( M + 1 \) dimensional table, or as a 2-dimensional table with \( \left( \prod_{j \in \text{pa}(i)} K_j \right) \times K_i \) entries by collapsing all the states of the parents of node \( i \). Note that \( \sum_{k'} \theta_{i,k,k'} = 1 \).

Assume a data set \( \mathcal{D} = \{ \mathbf{x}^{(n)} \}_{n=1}^N \).

How do we learn \( \theta \) from \( \mathcal{D} \)?
Learning parameters

Assume a data set \( D = \{x^{(n)}\}_{n=1}^{N} \). How do we learn \( \theta \) from \( D \)?

\[
p(x|\theta) = p(x_1|\theta_1)p(x_2|x_1, \theta_2)p(x_3|x_1, \theta_3)p(x_4|x_2, \theta_4)
\]

Likelihood:

\[
p(D|\theta) = \prod_{n=1}^{N} p(x^{(n)}|\theta)
\]

Log Likelihood:

\[
\log p(D|\theta) = \sum_{n=1}^{N} \sum_{i} \log p(x^{(n)}_i|x^{(n)}_{pa(i)}, \theta_i)
\]

This decomposes into sum of functions of \( \theta_i \). Each \( \theta_i \) can be optimized separately:

\[
\hat{\theta}_{i,k,k'} = \frac{n_{i,k,k'}}{\sum_{k''} n_{i,k,k''}}
\]

where \( n_{i,k,k'} \) is the number of times in \( D \) where \( x_i = k' \) and \( x_{pa(i)} = k \), where \( k \) represents a joint configuration of all the parents of \( i \) (i.e. takes on one of \( \prod_{j \in pa(i)} K_j \) values)

ML solution: Simply calculate frequencies!
Deriving the Maximum Likelihood Estimate

\[ p(y|x, \theta) = \prod_{k, \ell} \theta_{k, \ell}^{\delta(x, k) \delta(y, \ell)} \]

Dataset \( D = \{(x^{(n)}, y^{(n)}): n = 1 \ldots, N\} \)

\[
\mathcal{L}(\theta) = \log \prod_n p(y^{(n)}|x^{(n)}, \theta) \\
= \log \prod_n \prod_{k, \ell} \theta_{k, \ell}^{\delta(x^{(n)}, k) \delta(y^{(n)}, \ell)} \\
= \sum_{n,k,\ell} \delta(x^{(n)}, k) \delta(y^{(n)}, \ell) \log \theta_{k,\ell} \\
= \sum_{k,\ell} \left( \sum_n \delta(x^{(n)}, k) \delta(y^{(n)}, \ell) \right) \log \theta_{k,\ell} = \sum_{k,\ell} n_{k,\ell} \log \theta_{k,\ell}
\]

Maximize \( \mathcal{L}(\theta) \) w.r.t. \( \theta \) subject to \( \sum_{\ell} \theta_{k,\ell} = 1 \) for all \( k \).
Assume a model parameterised by $\theta$ with observable variables $Y$ and hidden variables $X$.

**Goal:** maximize parameter log likelihood given observed data.

$$
\mathcal{L}(\theta) = \log p(Y|\theta) = \log \sum_X p(Y, X|\theta)
$$
Maximum Likelihood Learning with Hidden Variables: The EM Algorithm

Goal: maximise parameter log likelihood given observables.

\[ \mathcal{L}(\theta) = \log p(Y|\theta) = \log \sum_X p(Y, X|\theta) \]

The Expectation Maximization (EM) algorithm (intuition):

Iterate between applying the following two steps:

- **The E step:** fill-in the hidden/missing variables
- **The M step:** apply complete data learning to filled-in data.
Maximum Likelihood Learning with Hidden Variables: 
The EM Algorithm

**Goal:** maximise parameter log likelihood given observables.

\[ \mathcal{L}(\theta) = \log p(Y|\theta) = \log \sum_X p(Y, X|\theta) \]

The EM algorithm (derivation):

\[ \mathcal{L}(\theta) = \log \sum_X q(X) \frac{p(Y, X|\theta)}{q(X)} \geq \sum_X q(X) \log \frac{p(Y, X|\theta)}{q(X)} = \mathcal{F}(q(X), \theta) \]

- **The E step:** maximize \( \mathcal{F}(q(X), \theta^{[t]}) \) wrt \( q(X) \) holding \( \theta^{[t]} \) fixed:
  \[ q(X) = p(X|Y, \theta^{[t]}) \]

- **The M step:** maximize \( \mathcal{F}(q(X), \theta) \) wrt \( \theta \) holding \( q(X) \) fixed:
  \[ \theta^{[t+1]} \leftarrow \arg\max_\theta \sum_X q(X) \log p(Y, X|\theta) \]

The E-step requires solving the *inference* problem, finding the distribution over the hidden variables \( p(X|Y, \theta^{[t]}) \) given the current model parameters. This can be done using **belief propagation** or the **junction tree algorithm**.
Maximum Likelihood Learning without and with Hidden Variables

ML Learning with Complete Data (No Hidden Variables)

Log likelihood decomposes into sum of functions of $\theta_i$. Each $\theta_i$ can be optimized separately:

$$\hat{\theta}_{ijk} \leftarrow \frac{n_{ijk}}{\sum_{k'} n_{ijk'}}$$

where $n_{ijk}$ is the number of times in $D$ where $x_i = k$ and $x_{pa(i)} = j$.

Maximum likelihood solution: Simply calculate frequencies!

ML Learning with Incomplete Data (i.e. with Hidden Variables)

Iterative EM algorithm

- **E step**: compute expected counts given previous settings of parameters $E[n_{ijk}|D, \theta^{[t]}]$.

- **M step**: re-estimate parameters using these expected counts

$$\theta_{ijk}^{[t+1]} \leftarrow \frac{E[n_{ijk}|D, \theta^{[t]}]}{\sum_{k'} E[n_{ijk'}|D, \theta^{[t]}]}$$
Bayesian Learning

Apply the basic rules of probability to learning from data.

Data set: $\mathcal{D} = \{x_1, \ldots, x_n\}$ Models: $m$, $m'$ etc. Model parameters: $\theta$

Prior probability of models: $P(m)$, $P(m')$ etc.
Prior probabilities of model parameters: $P(\theta|m)$
Model of data given parameters (likelihood model): $P(x|\theta, m)$

If the data are independently and identically distributed then:

$$P(\mathcal{D}|\theta, m) = \prod_{i=1}^{n} P(x_i|\theta, m)$$

Posterior probability of model parameters:

$$P(\theta|\mathcal{D}, m) = \frac{P(\mathcal{D}|\theta, m)P(\theta|m)}{P(\mathcal{D}|m)}$$

Posterior probability of models:

$$P(m|\mathcal{D}) = \frac{P(m)P(\mathcal{D}|m)}{P(\mathcal{D})}$$
Bayesian parameter learning with no hidden variables

Let $n_{ijk}$ be the number of times $(x_i^{(n)} = k$ and $x_{pa(i)}^{(n)} = j)$ in $D$. For each $i$ and $j$, $\theta_{ij}$ is a probability vector of length $K_i \times 1$.

Since $x_i$ is a discrete variable with probabilities given by $\theta_{i.,.}$, the likelihood is:

$$p(D|\theta) = \prod_{n} \prod_{i} p(x_i^{(n)} | x_{pa(i)}^{(n)}, \theta) = \prod_{i} \prod_{j} \prod_{k} \theta_{ijk}^{n_{ijk}}$$

If we choose a prior on $\theta$ of the form:

$$p(\theta) = c \prod_{i} \prod_{j} \prod_{k} \theta_{ijk}^{\alpha_{ijk}-1}$$

where $c$ is a normalization constant, and $\sum_k \theta_{ijk} = 1 \ \forall i, j$, then the posterior distribution also has the same form:

$$p(\theta|D) = c' \prod_{i} \prod_{j} \prod_{k} \theta_{ijk}^{\tilde{\alpha}_{ijk}-1}$$

where $\tilde{\alpha}_{ijk} = \alpha_{ijk} + n_{ijk}$.

This distribution is called the Dirichlet distribution.
Dirichlet Distribution

The **Dirichlet distribution** is a distribution over the $K$-dim probability simplex.

Let $\theta$ be a $K$-dimensional vector s.t. $\forall j : \theta_j \geq 0$ and $\sum_{j=1}^{K} \theta_j = 1$

$$p(\theta|\alpha) = \text{Dir}(\alpha_1, \ldots, \alpha_K) \overset{\text{def}}{=} \frac{\Gamma(\sum_j \alpha_j)}{\prod_j \Gamma(\alpha_j)} \prod_{j=1}^{K} \theta_{j}^{\alpha_j-1}$$

where the **first term** is a normalization constant\footnote{\[1\]} and $E(\theta_j) = \alpha_j / (\sum_k \alpha_k)$

The Dirichlet is **conjugate to the multinomial distribution**. Let

$$x|\theta \sim \text{Multinomial}(\cdot|\theta)$$

That is, $p(x = j|\theta) = \theta_j$. Then the posterior is also Dirichlet:

$$p(\theta|x = j, \alpha) = \frac{p(x = j|\theta)p(\theta|\alpha)}{p(x = j|\alpha)} = \text{Dir}(\tilde{\alpha})$$

where $\tilde{\alpha}_j = \alpha_j + 1$, and $\forall \ell \neq j : \tilde{\alpha}_\ell = \alpha_\ell$

\footnote{\[1\]} $\Gamma(x) = \frac{\Gamma(x-1)}{x} = \int_0^{\infty} t^{x-1} e^{-t} dt$. For integer $n$, $\Gamma(n) = (n-1)!$
Dirichlet Distributions

Examples of Dirichlet distributions over $\theta = (\theta_1, \theta_2, \theta_3)$ which can be plotted in 2D since $\theta_3 = 1 - \theta_1 - \theta_2$: 

- Dirichlet(1,1,1)
- Dirichlet(2,2,2)
- Dirichlet(10,10,10)
- Dirichlet(2,10,2)
- Dirichlet(2,2,10)
- Dirichlet(0.9,0.9,0.9)
Example

Assume $\alpha_{ijk} = 1 \ \forall i, j, k$.

This corresponds to a **uniform** prior distribution over parameters $\theta$. This is not a very strong/dogmatic prior, since any parameter setting is assumed a priori possible.

After observed data $\mathcal{D}$, what are the parameter posterior distributions?

$$p(\theta_{i,j}|\mathcal{D}) = \text{Dir}(n_{ij} + 1)$$

This distribution predicts, for future data:

$$p(x_i = k|x_{\text{pa}(i)} = j, \mathcal{D}) = \frac{n_{ijk} + 1}{\sum_{k'}(n_{ijk'} + 1)}$$

Adding 1 to each of the counts is a form of smoothing called “Laplace’s Rule”.
Bayesian parameter learning with hidden variables

**Notation:** let $\mathcal{D}$ be the observed data set, $\mathcal{X}$ be hidden variables, and $\theta$ be model parameters. Assume discrete variables and Dirichlet priors on $\theta$

**Goal:** to infer $p(\theta|\mathcal{D}) = \sum_{\mathcal{X}} p(\mathcal{X}, \theta|\mathcal{D})$

**Problem:** since (a) 

$$p(\theta|\mathcal{D}) = \sum_{\mathcal{X}} p(\theta|\mathcal{X}, \mathcal{D}) p(\mathcal{X}|\mathcal{D}),$$

and (b) for every way of filling in the missing data, $p(\theta|\mathcal{X}, \mathcal{D})$ is a Dirichlet distribution, and (c) there are exponentially many ways of filling in $\mathcal{X}$, it follows that $p(\theta|\mathcal{D})$ is a mixture of Dirichlets with exponentially many terms!

**Solutions:**

- Find a single best ("Viterbi") completion of $\mathcal{X}$ (Stolcke and Omohundro, 1993)
- Markov chain Monte Carlo methods
- Variational Bayesian (VB) methods (Beal and Ghahramani, 2003)
## Summary of parameter learning

<table>
<thead>
<tr>
<th></th>
<th>Complete (fully observed) data</th>
<th>Incomplete (hidden / missing) data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML</td>
<td>calculate frequencies</td>
<td>EM</td>
</tr>
<tr>
<td>Bayesian</td>
<td>update Dirichlet distributions</td>
<td>MCMC / Viterbi / VB</td>
</tr>
</tbody>
</table>

- For complete data Bayesian learning is not more costly than ML.
- For incomplete data VB ≈ EM time complexity.
- Other parameter priors are possible but Dirichlet is pretty flexible and intuitive.
- For non-discrete data, similar ideas but generally harder inference and learning.
Structure learning

Given a data set of observations of \((A, B, C, D, E)\) can we learn the structure of the graphical model?

Let \(m\) denote the graph structure = the set of edges.
Structure learning

**Constraint-Based Learning:** Use statistical tests of marginal and conditional independence. Find the set of DAGs whose d-separation relations match the results of conditional independence tests.

**Score-Based Learning:** Use a global score such as the BIC score or Bayesian marginal likelihood. Find the structures that maximize this score.
Score-based structure learning for complete data

Consider a graphical model with structure $m$, discrete observed data $\mathcal{D}$, and parameters $\theta$. Assume Dirichlet priors.

The Bayesian marginal likelihood score is easy to compute:

$$
\text{score}(m) = \log p(\mathcal{D}|m) = \log \int p(\mathcal{D}|\theta, m)p(\theta|m) d\theta
$$

$$
\text{score}(m) = \sum_i \sum_j \left[ \log \Gamma(\sum_k \alpha_{ijk}) - \sum_k \log \Gamma(\alpha_{ijk}) - \log \Gamma(\sum_k \tilde{\alpha}_{ijk}) + \sum_k \log \Gamma(\tilde{\alpha}_{ijk}) \right]
$$

where $\tilde{\alpha}_{ijk} = \alpha_{ijk} + n_{ijk}$. **Note that the score decomposes over $i$**.

One can incorporate structure prior information $p(m)$ as well:

$$
\text{score}(m) = \log p(\mathcal{D}|m) + \log p(m)
$$

**Greedy search algorithm**: Start with $m$. Consider modifications $m \rightarrow m'$ (edge deletions, additions, reversals). Accept $m'$ if $\text{score}(m') > \text{score}(m)$. Repeat.

**Bayesian inference of model structure**: Run MCMC on $m$. 

Bayesian Structural EM for incomplete data

Consider a graphical model with structure $m$, observed data $D$, hidden variables $X$ and parameters $\theta$

The Bayesian score is generally intractable to compute:

$$\text{score}(m) = p(D|m) = \int \sum_{X} p(X, \theta, D|m) d\theta$$

**Bayesian Structure EM** (Friedman, 1998):

1. compute MAP parameters $\hat{\theta}$ for current model $m$ using EM
2. find hidden variable distribution $p(X|D, \hat{\theta})$
3. for a small set of candidate structures compute or approximate

$$\text{score}(m') = \sum_{X} p(X|D, \hat{\theta}) \log p(D, X|m')$$

4. $m \leftarrow m'$ with highest score
Directed Graphical Models and Causality

Causal relationships are a fundamental component of cognition and scientific discovery.

Even though the independence relations are identical, there is a causal difference between

- “smoking” → “yellow teeth”
- “yellow teeth” → “smoking”

**Key idea:** interventions and the do-calculus:

$$p(S|Y = y) \neq p(S|\text{do}(Y = y))$$

$$p(Y|S = s) = p(Y|\text{do}(S = s))$$

Causal relationships are robust to interventions on the parents.

The **key difficulty** in learning causal relationships from observational data is the presence of hidden common causes:
Learning parameters and structure in undirected graphs

\[ p(x|\theta) = \frac{1}{Z(\theta)} \prod_j g_j(x_{C_j}; \theta_j) \text{ where } Z(\theta) = \sum_x \prod_j g_j(x_{C_j}; \theta_j). \]

**Problem:** computing \( Z(\theta) \) is computationally intractable for general (non-tree-structured) undirected models. Therefore, maximum-likelihood learning of parameters is generally intractable, Bayesian scoring of structures is intractable, etc.

**Solutions:**

- directly approximate \( Z(\theta) \) and/or its derivatives (cf. Boltzmann machine learning; contrastive divergence; pseudo-likelihood)
- use approx inference methods (e.g. loopy belief propagation, bounding methods, EP).

See: (Murray and Ghahramani, 2004; Murray et al, 2006) for Bayesian learning in undirected models.
Summary

- Parameter learning in directed models:
  - complete and incomplete data;
  - ML and Bayesian methods

- Structure learning in directed models: complete and incomplete data

- Causality

- Parameter and Structure learning in undirected models
Readings and References

  [http://learning.eng.cam.ac.uk/zoubin/papers/BeaGha06.pdf](http://learning.eng.cam.ac.uk/zoubin/papers/BeaGha06.pdf)


  [http://www.springerlink.com/index/NQ13817217667435.pdf](http://www.springerlink.com/index/NQ13817217667435.pdf)


Learning the structure of deep sparse directed graphical models

Ryan P. Adams\textsuperscript{1}, Hanna M. Wallach\textsuperscript{2}, and Zoubin Ghahramani\textsuperscript{3}

\textsuperscript{1}University of Toronto
\textsuperscript{2}University of Massachusetts at Amherst
\textsuperscript{3}University of Cambridge

Machine Learning Summer School
August 2009
Motivation

- Present some recent research on **graphical model** structure learning...
- ...related to **deep belief networks**...
- ...which uses **Markov chain Monte Carlo** inference...
- ...in a **non-parametric** Bayesian model.
There is a great deal of interest on “deep belief networks”.

Deep belief nets are probabilistic generative models that are composed of multiple layers of stochastic, latent variables. The latent variables typically have binary values and are often called hidden units or feature detectors. The top two layers have undirected, symmetric connections between them and form an associative memory. The lower layers receive top-down, directed connections from the layer above. The states of the units in the lowest layer represent a data vector.

Deep networks

Questions:

- How many layers should there be?
- How wide should each layer be?
- What sorts of units?

Goal: To learn the structure of a deep network.

Approach: A nonparametric Bayesian method that learns the structure of a layered directed deep belief network.
Layered directed deep networks

\[
p(x) = \prod_{i=1}^{K} p(x_i | x_{\pi_i})
\]

Where \( x = (x_1, \ldots, x_K) \) and \( \pi_i \) are the parents of node \( i \).

aka Bayesian networks, probabilistic directed graphical models.

Assume a layered graph structure.

How many layers? How wide should each layer be?
Priors over graph structures

Let $z_{ij}^{(m)} = 1$ mean that $j \in \pi_i$, that is, node $j$ is a parent of node $i$ in layer $m$.

If we specify a sequence of matrices $Z^{(0)}$, $Z^{(1)}$, $Z^{(2)}$, ... we have defined the layered graph structure.
**Previous work**

* *A Non-Parametric Bayesian Method for Inferring Hidden Causes*  
(Wood, Griffiths, Ghahramani, 2006)

Inferring stroke localization from patient symptoms:

**Y** - binary latent factors (diseases, *k*)  
**Z** - graph structure (∼ Indian Buffet Process (IBP))  
**X** - observed binary features (56 symptoms, *s*)

**Noisy-or:**  
\[ P(x_{sp} = 1|Z, Y, \lambda, \epsilon) = 1 - (1 - \lambda)\sum_k z_{sk}y_{kp}(1 - \epsilon) \]

The IBP defines a distribution on sparse binary matrices with a countably infinite number of columns.

Graphical models with an unbounded number of hidden units.
$z_{ik} = 1$ means object $i$ has feature $k$:

$$z_{ik} \sim \text{Bernoulli}(\theta_k)$$

$$\theta_k \sim \text{Beta}(\alpha/K, 1)$$

- Note that $P(z_{ik} = 1|\alpha) = E(\theta_k) = \frac{\alpha/K}{\alpha/K+1}$, so as $K$ grows larger the matrix gets sparser.

- So if $Z$ is $N \times K$, the expected number of nonzero entries is $N\alpha/(1 + \alpha/K) < N\alpha$.

- Even in the $K \to \infty$ limit, the matrix is expected to have a finite number of non-zero entries.

- Two parameter extension $\theta_k \sim \text{Beta}(\alpha\beta/K, \beta)$
First customer starts at the left of the buffet, and takes a serving from each dish, stopping after a Poisson($\alpha$) number of dishes as his plate becomes overburdened.

The $n$th customer moves along the buffet, sampling dishes in proportion to their popularity, serving himself dish $k$ with probability $m_k/n$, and trying Poisson($\alpha/n$) new dishes.

The customer-dish matrix is the feature matrix, $Z$. 
Cascading Indian buffet process

Start with $K^{(0)}$ rows (visible units)

- $Z^{(0)} \sim IBP(\alpha, \beta)$ with $K^{(0)}$ rows and $K^{(1)}$ non-zero columns
- $Z^{(1)} \sim IBP(\alpha, \beta)$ with $K^{(1)}$ rows and $K^{(2)}$ non-zero columns
- $Z^{(2)} \sim IBP(\alpha, \beta)$ with $K^{(2)}$ rows and $K^{(3)}$ non-zero columns
- ...  

This defines a sequences of infinite sparse binary matrices.
Properties of the Cascading IBP

\[ Z^{(m)} \sim IBP(\alpha, \beta) \quad \text{for } m = 0, 1, 2, \ldots \]

- The expected in-degree of each unit (number of parents) is \( \alpha \).
- The expected out-degree of each unit in \( m \) (number of children) is

\[
c(\beta, m) = 1 + \frac{K^{(m-1)} - 1}{1 + \beta}
\]

Note that \( \lim_{\beta \to 0} c(\beta, m) = K^{(m-1)} \) and \( \lim_{\beta \to \infty} c(\beta, m) = 1 \).
- Hidden units are exchangeable at each layer.
- **Theorem:** For \( K^{(m)} \in \mathbb{N} \), \( 0 < \alpha < \infty \), \( 0 < \beta < \infty \), the sequence of \( K^{(m)} \) defined by the CIBP reaches the absorption state 0, with probability one, i.e. \( \lim_{m \to \infty} p(K^{(m)} = 0) = 1 \).
Samples from the prior over structures

\[ \alpha = 1, \beta = 1 \]

\[ \alpha = 1, \beta = \frac{1}{2} \]

\[ \alpha = \frac{1}{2}, \beta = 1 \]

\[ \alpha = 1, \beta = 2 \]

\[ \alpha = \frac{3}{2}, \beta = 1 \]

Samples from the CIBP prior starting from five visible units.
What kinds of units?

We want a model that is flexible enough to learn what types of unit it needs, ranging from binary to linear-Gaussian.

This idea was explored in Nonlinear Gaussian belief networks (NLGBNs) by (Frey and Hinton, 1999).

Let $u^{(m)}$ be the activity of units in layer $m$.

$$ y^{(m)} = (W^{(m+1)} \odot Z^{(m+1)})u^{(m+1)} + \gamma^{(m)} $$

where $W$ is a weight matrix, $\gamma$ is a bias vector and $\odot$ is Hadamard (elementwise) product.

$$ u_k^{(m)} = \sigma(y_k^{(m)} + \epsilon_k^{(m)}) $$

$\sigma$ is a sigmoid function and noise $\epsilon_k^{(m)} \sim \mathcal{N}(0, \frac{1}{\nu_k^{(m)}})$ has precision $\nu_k^{(m)}$. 
NLGBN units

Three modes of operation for the NLGBN unit. The black solid line shows the zero mean distribution, the red dashed line shows a pre-sigmoid mean of $+1$ and the blue dash-dot line shows a pre-sigmoid mean of $-1$.

(a) Binary behavior from small precision.
(b) Roughly Gaussian behavior from medium precision.
(c) Deterministic behavior from large precision.
Inference
using Markov chain Monte Carlo

\( W \sim \mathcal{N} \quad \gamma \sim \mathcal{N} \quad \alpha \sim \mathcal{G} \quad \beta \sim \mathcal{G} \quad \nu \sim \mathcal{G} \)

We design an MCMC scheme to sample from the posterior:

\[
p(\{Z^{(m)}, W^{(m)}\}_{m=1}^{\infty}, \{\gamma^{(m)}, \nu^{(m)}\}_{m=0}^{\infty}, \{u^{(m)}_n\}_{m=1}^{\infty})_{n=1}^{N} | \{x_n\}_{n=1}^{N})
\]

- \(u\) - slice sample
- \(W\) and \(\gamma\) - Gibbs
- \(\nu\) - Gibbs
- \(Z\) - Gibbs (cf Algorithm 8 of CRPs)
Experiments on MNIST data

Small subset
28 × 28 pixels
100 images (10 from each class)
Samples from Posterior over Structures
First-Layer Features
Visible Unit Precisions
This work provides an initial attempt at addressing three issues with layered belief networks.

- It provides a way to learn belief networks that contain an arbitrary number of hidden units with nontrivial joint distributions due to a deep structure.
- It allows the units to have different operating regimes and infer appropriate local representations ranging from discrete binary to nonlinear continuous behavior.
- It provides a way to infer the appropriate directed graph structure of a layered network.

Initial work... many open questions!
Graphical models provide a powerful and intuitive framework for modelling and inference.

Directed, undirected and factor graphs.

Inference by message passing.

Parameter and structure learning.

A recent bit of research on structure learning.

Thanks!
Questions