Graphical Models

Zoubin Ghahramani

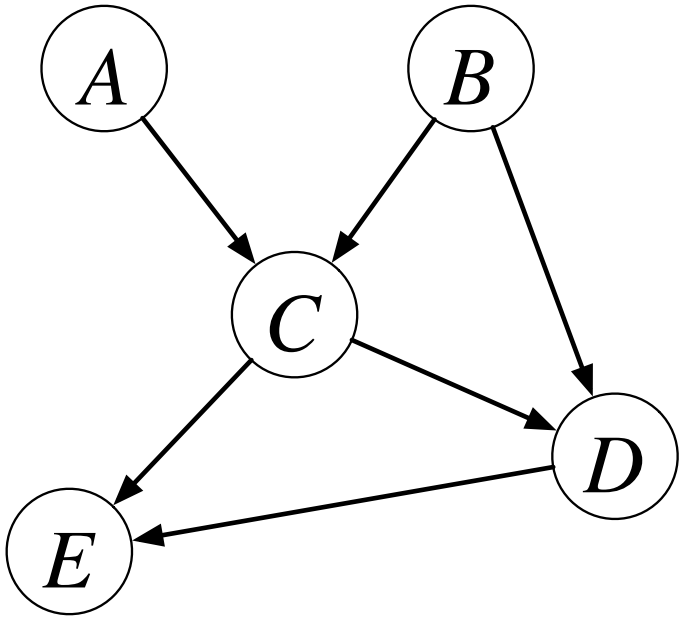
University of Cambridge, UK

Uber AI Labs, San Francisco, USA

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

MLSS 2017
Tübingen

Representing knowledge through graphical models



- Nodes correspond to random variables
- Edges represent statistical dependencies between the variables

Why do we need graphical models?

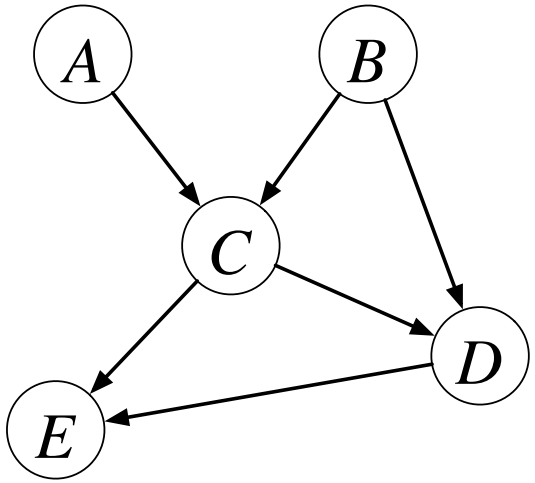
- Graphs are an **intuitive** way of representing and visualising the relationships between many variables. (Examples: family trees, electric circuit diagrams, neural networks)

- 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 $\times$ graph theory $\times$ computer science.

Directed Acyclic Graphical Models (Bayesian Networks)



A DAG Model / Bayesian network[1] corresponds to a factorization of the joint probability distribution:

$$p(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|B, C)p(E|C, D)$$

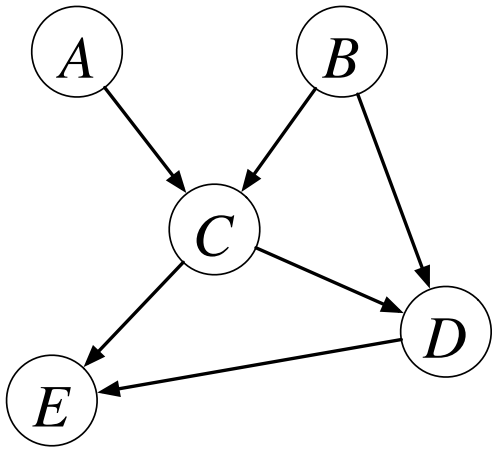
In general:

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

where $\mathsf{pa}(i)$ 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: $X \perp\!\!\!\perp Y | \mathcal{V}$ if $\mathcal{V}$ d-separates X from Y

Definition: $\mathcal{V}$ d-separates X from Y if *every* undirected path[2] 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 $(\rightarrow W \leftarrow)$[3] 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] An undirected path ignores the direction of the edges.

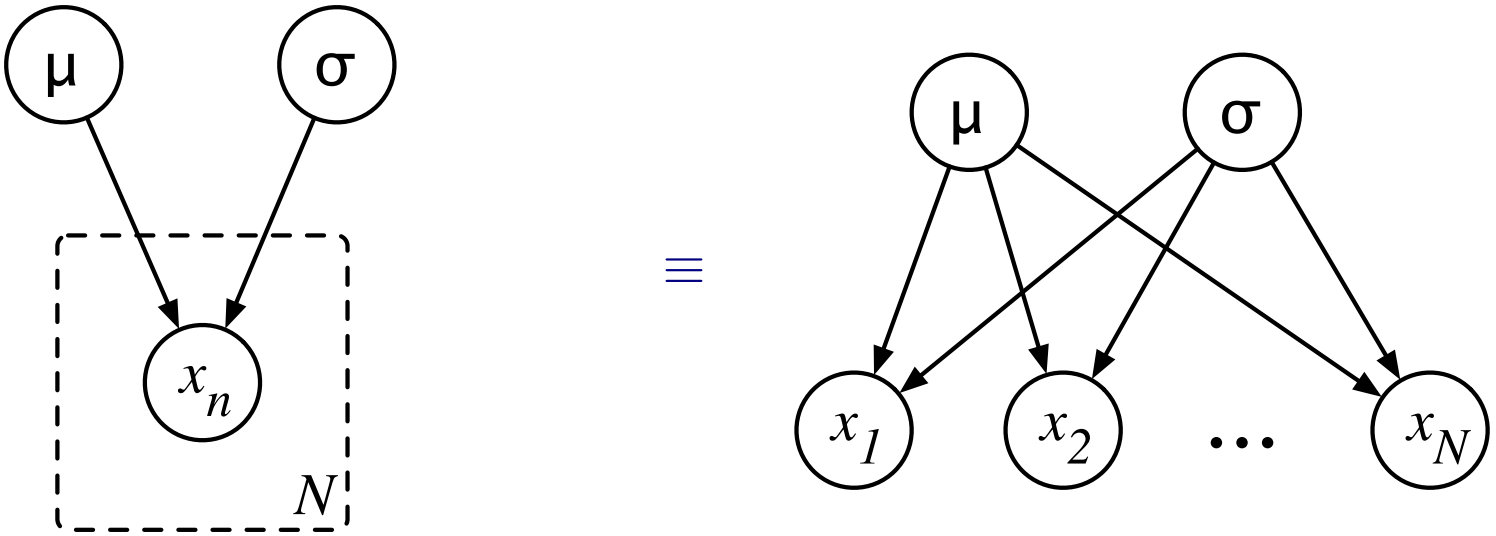
[3] Note that converging arrows *along the path* only refers to what happens on that path. Also called a *collider*.

Directed Graphs for Statistical Models: Plate Notation

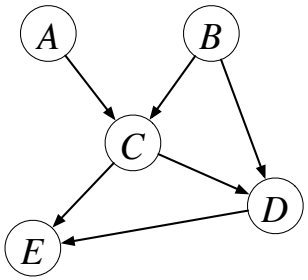
Consider the following simple model. A data set of N points is generated i.i.d. from a Gaussian with mean μ and standard deviation σ:

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

This can be represented graphically as follows:



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:

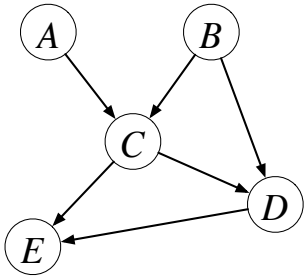
$$p(A, C = c) = \sum_{B,D,E} p(A, B, C = c, D, E) \qquad \text{[16 terms]}$$

$$p(C = c) = \sum_{A} p(A, C = c) \qquad \text{[2 terms]}$$

$$p(A|C = c) = \frac{p(A, C = c)}{p(C = c)} \qquad \text{[2 terms]}$$

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:

$$\begin{aligned} p(A, C = c) &= \sum_{B,D,E} p(A)p(B)p(C = c|A, B)p(D|B, C = c)p(E|C = c, D) \\ &= \sum_{B} p(A)p(B)p(C = c|A, B) \sum_{D} p(D|B, C = c) \sum_{E} p(E|C = c, D) \\ &= \sum_{B} p(A)p(B)p(C = c|A, B) \qquad \text{[4 terms]} \end{aligned}$$

Total: 4+2+2 = 8 terms

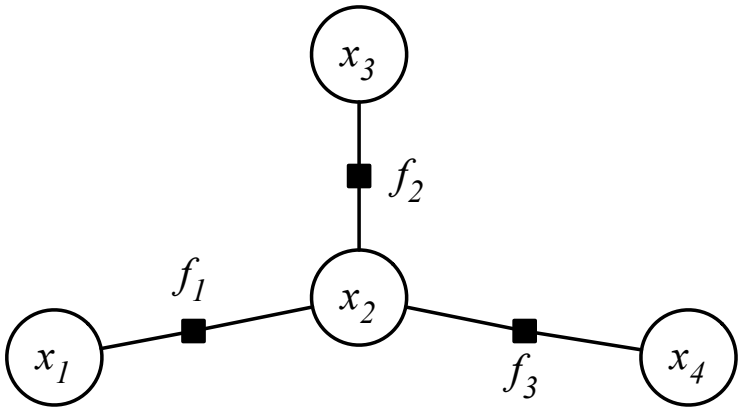
Belief propagation methods use the conditional independence relationships in a graph to do efficient inference (for singly connected graphs, exponential gains in efficiency!).

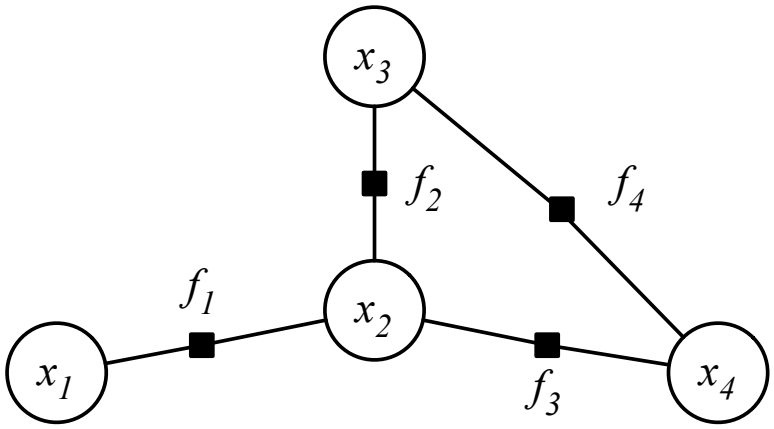
Factor graph propagation

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

$$
\begin{aligned}
p(\mathbf{x}) &= p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) \\
&\equiv f_1(x_1, x_2)f_2(x_2, x_3)f_3(x_2, x_4)
\end{aligned}
$$

Singly connected vs Multiply connected factor graphs:



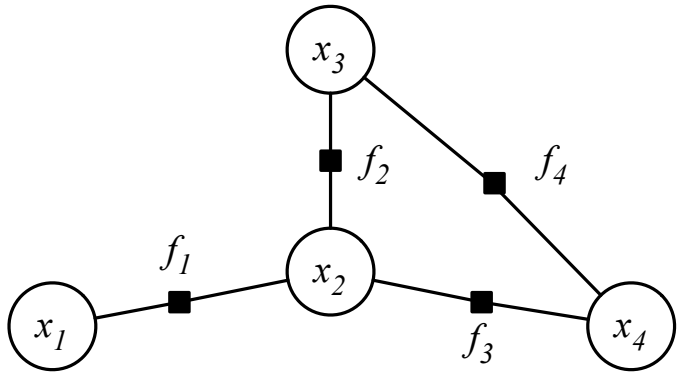


Factor Graphs

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 $\mathrm{n}(x)$ denote the set of factor nodes that are neighbors of x.
Let $\mathrm{n}(f)$ 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 viceversa.

message from variable x to factor f:

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

message from factor f to variable x:

$$\mu_{f \to x}(x) = \sum_{\mathbf{x} \setminus x} \left(f(\mathbf{x}) \prod_{y \in \mathrm{n}(f) \setminus \{x\}} \mu_{y \to f}(y) \right)$$

where $\mathbf{x}$ are the variables that factor f depends on, and $\sum_{\mathbf{x} \setminus x}$ is a sum over all variables neighboring factor f except x.

Propagation in Factor Graphs

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

message from variable x to factor f:

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

message from factor f to variable x:

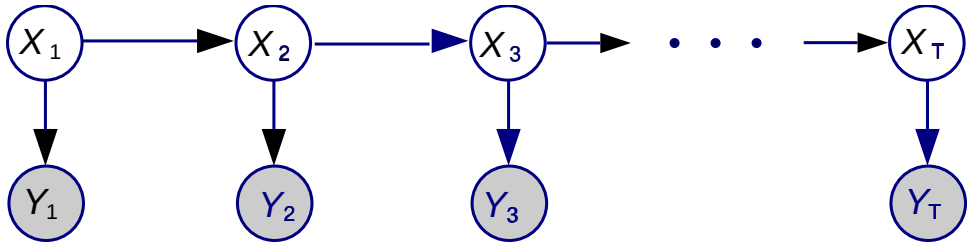
$$\mu_{f \to x}(x) = \sum_{\mathbf{x} \setminus x} \left(f(\mathbf{x}) \prod_{y \in \mathrm{n}(f) \setminus \{x\}} \mu_{y \to f}(y) \right)$$

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:

$$p(x) \propto \prod_{h \in \mathrm{n}(x)} \mu_{h \to x}(x)$$

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.

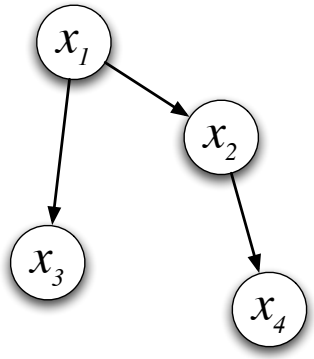
Software for Graphical Models

- BUGS and WinBUGS: inference via Gibbs sampling, not very scalable
- HUGIN: widely used, commercial, focus on exact inference
- Kevin Murphy's Bayes Net Toolbox: Matlab, widely used
- Microsoft's Infer.NET: advanced scalable libraries implementing factor graph propagation, EP, and variational message passing.
- Jeff Bilmes' GMTK: very good at HMMs and related time series models
- *many others*, see `http://people.cs.ubc.ca/~murphyk/Software/bnsoft.html`
- Much of this is subsumed by general Probabilistic Programming frameworks: e.g. Church, WebPPL, Turing, Anglican, Edward, STAN,...

Summary

- inference consists of the problem of computing $p(\text{variables of interest} | \text{observed variables})$
- for singly connected graphs, belief propagation / factor graph propagation solves this problem exactly.
- 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"—run propagation as if graph is simply connected; often works well in practice.

Learning parameters



$$p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)$$

θ_2

x_1 \ x_2			
	0.2	0.3	0.5
	0.1	0.6	0.3

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: θ_1 has K_1 entries, θ_2 has $K_1 \times K_2$ entries, etc.

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

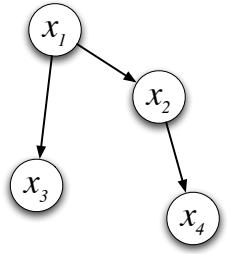
$$p(x_1 = k) = \theta_{1,k} \qquad p(x_2 = k'|x_1 = k) = \theta_{2,k,k'}$$

If node i has M parents, θ_i can be represented either as an $M+1$ dimensional table, or as a 2-dimensional table with $\left(\prod_{j \in \mathrm{pa}(i)} K_j\right) \times K_i$ 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 θ from $\mathcal{D}$?

Learning parameters



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

$$p(\mathbf{x}|\boldsymbol{\theta}) = p(x_1|\theta_1)p(x_2|x_1,\theta_2)p(x_3|x_1,\theta_3)p(x_4|x_2,\theta_4)$$

Likelihood:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\mathbf{x}^{(n)}|\boldsymbol{\theta})$$

Log Likelihood:

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{n=1}^{N}\sum_{i} \log p(x_i^{(n)}|x_{\mathrm{pa}(i)}^{(n)},\theta_i)$$

This decomposes into sum of functions of θ_i. Each θ_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 $\mathcal{D}$ where $x_i = k'$ and $x_{\mathrm{pa}(i)} = k$, where k represents a joint configuration of all the parents of i (i.e. takes on one of $\prod_{j\in\mathrm{pa}(i)} K_j$ values)

ML solution: Simply calculate frequencies!

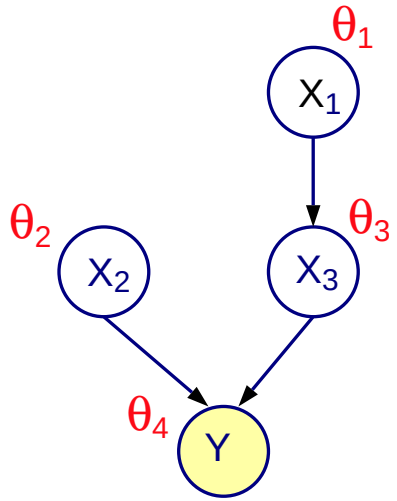
n_2

x_1 \ x_2			
	2	3	0
	3	1	6

$\Rightarrow$ θ_2

x_1 \ x_2			
	0.4	0.6	0
	0.3	0.1	0.6

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.

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: θ

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:

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

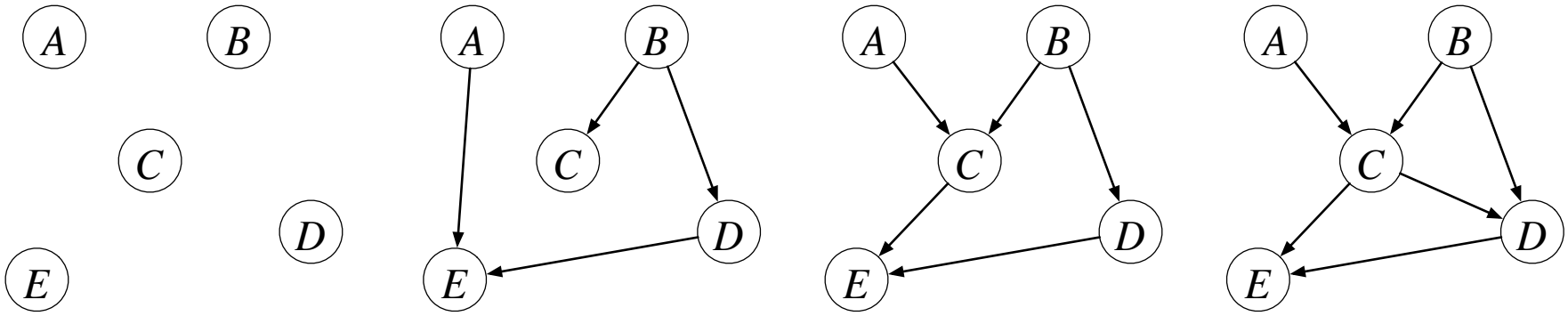
Summary of parameter learning

	Complete (fully observed) data	Incomplete (hidden /missing) data
ML	calculate frequencies	EM
Bayesian	update Dirichlet distributions	MCMC / Viterbi / VB

- For complete data Bayesian learning is not more costly than ML
- For incomplete data VB $\approx$ 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.

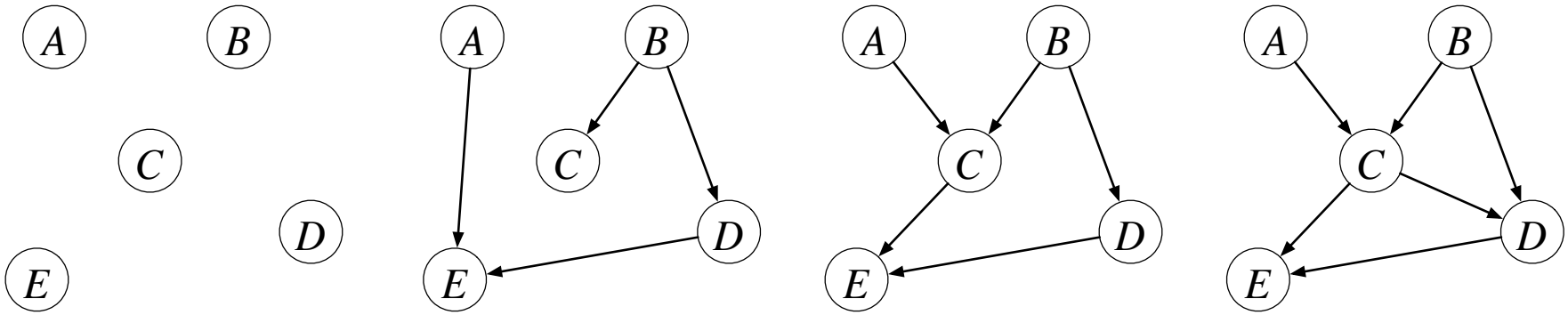
Structure learning in graphical models

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



Let G 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.

Bayesian Methods

Everything follows from two simple rules:

Sum rule: $P(x) = \sum_y P(x, y)$

Product rule: $P(x, y) = P(x)P(y|x)$

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

$P(\mathcal{D}\|\theta)$	likelihood of θ
$P(\theta)$	prior probability of θ
$P(\theta\|\mathcal{D})$	posterior of θ given $\mathcal{D}$

Prediction:

$$P(x|\mathcal{D}, m) = \int P(x|\theta, \mathcal{D}, m)P(\theta|\mathcal{D}, m)d\theta$$

Model Comparison:

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

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

Score-based structure learning for complete data

Consider a graphical model with structure m, discrete observed data $\mathcal{D}$, and parameters θ. 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 $m \to m'$ (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 *in*complete data

Consider a graphical model with structure m, observed data $\mathcal{D}$, hidden variables $\mathcal{X}$ and parameters θ

The Bayesian score is generally intractable to compute:

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

Bayesian Structure EM (Friedman, 1998):

1. compute MAP parameters $\hat{\theta}$ for current model m using EM

2. find hidden variable distribution $p(\mathcal{X}|\mathcal{D}, \hat{\theta})$

3. for a small set of candidate structures compute or approximate

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

4. $m \leftarrow m'$ with highest score

Graphical Models and Probabilistic Programs

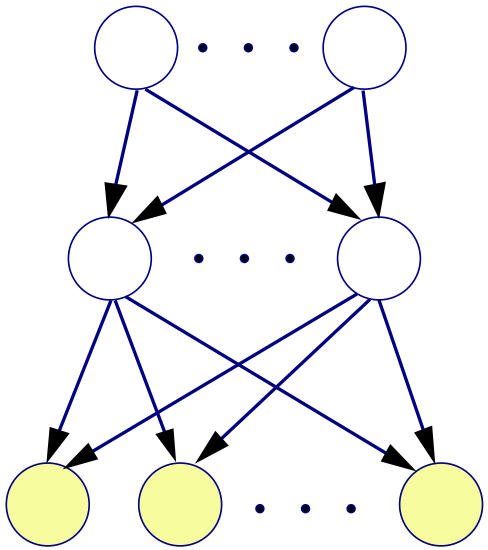
- **Graphical models** express conditional independence relationships between a usually fixed, finite set of variables.

- **Probabilistic programs** define the relationship between the variables through a computer program (simulator) than can generate the variables.

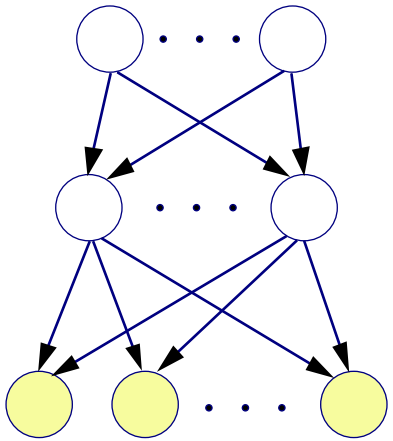
- PPs are generally *more expressive* than directed GMs:
  - they can capture independence relationships and even existence of variables, conditional on the values of other random variables.
  - they can express models with potentially unbounded numbers of random variables
  - they make explicit the functional form of all relationships.

Graphical Models and Deep Learning

- **Graphical models** represent relationships between random variables
- **Deep Learning** represents nonlinear functions as neural networks
- You can mix and match as you wish by
  - introducting hidden variables in the NN
  - using NNs to model the conditional probability distributions



Sigmoid Belief Networks



Connectionist learning of belief networks

Radford M. Neal

Department of Computer Science, University of Toronto, 10 King's College Road, Toronto, Ontario, Canada M5S 1A4

Received January 1991
Revised November 1991

Abstract

Neal, R.M., Connectionist learning of belief networks, Artificial Intelligence 56 (1992) 71–113.

Connectionist learning procedures are presented for "sigmoid" and "noisy-OR" varieties of probabilistic belief networks. These networks have previously been seen primarily as a means of representing knowledge derived from experts. Here it is shown that the "Gibbs sampling" simulation procedure for such networks can support maximum-likelihood learning from empirical data through local gradient ascent. This learning procedure resembles that used for "Boltzmann machines", and like it, allows the use of "hidden" variables to model correlations between visible variables. Due to the directed nature of the connections in a belief network, however, the "negative phase" of Boltzmann machine learning is unnecessary. Experimental results show that, as a result, learning in a sigmoid belief network can be faster than in a Boltzmann machine. These networks have other advantages over Boltzmann machines in pattern classification and decision making applications, are naturally applicable to unsupervised learning problems, and provide a link between work on connectionist learning and work on the representation of expert knowledge.

- Explicit link between feedforward neural networks (aka connectionist networks) and graphical models (aka belief networks).
- Gibbs samples over hidden units.
- A Bayesian nonparametric version of this model which samples over number of hidden units, number of layers, and types of hidden units is given in (Adams, Wallach, and Ghahramani, 2010)

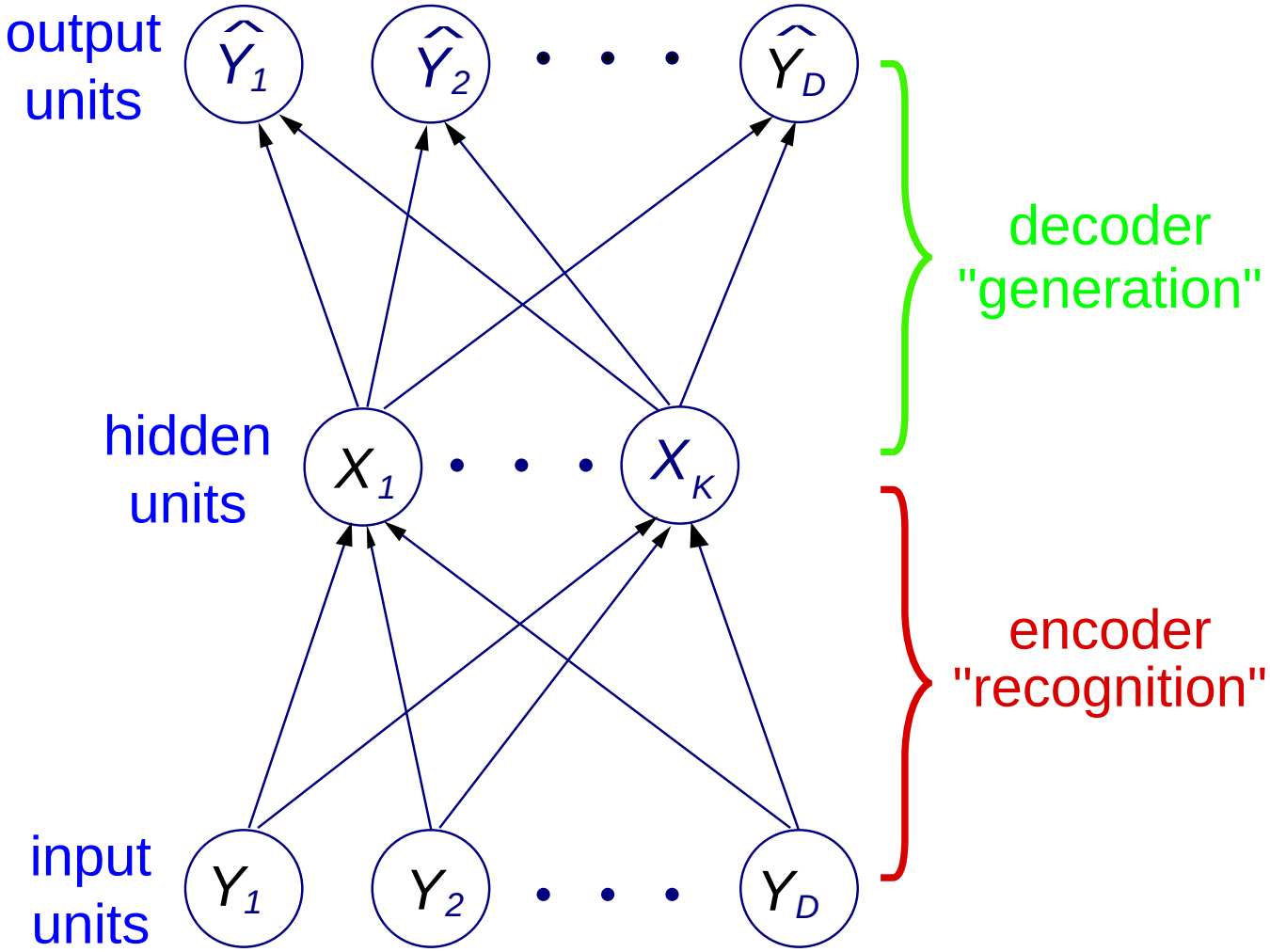
LEARNING THE STRUCTURE OF DEEP SPARSE GRAPHICAL MODELS

By Ryan P. Adams*, Hanna M. Wallach and Zoubin Ghahramani

University of Toronto, University of Massachusetts and University of Cambridge

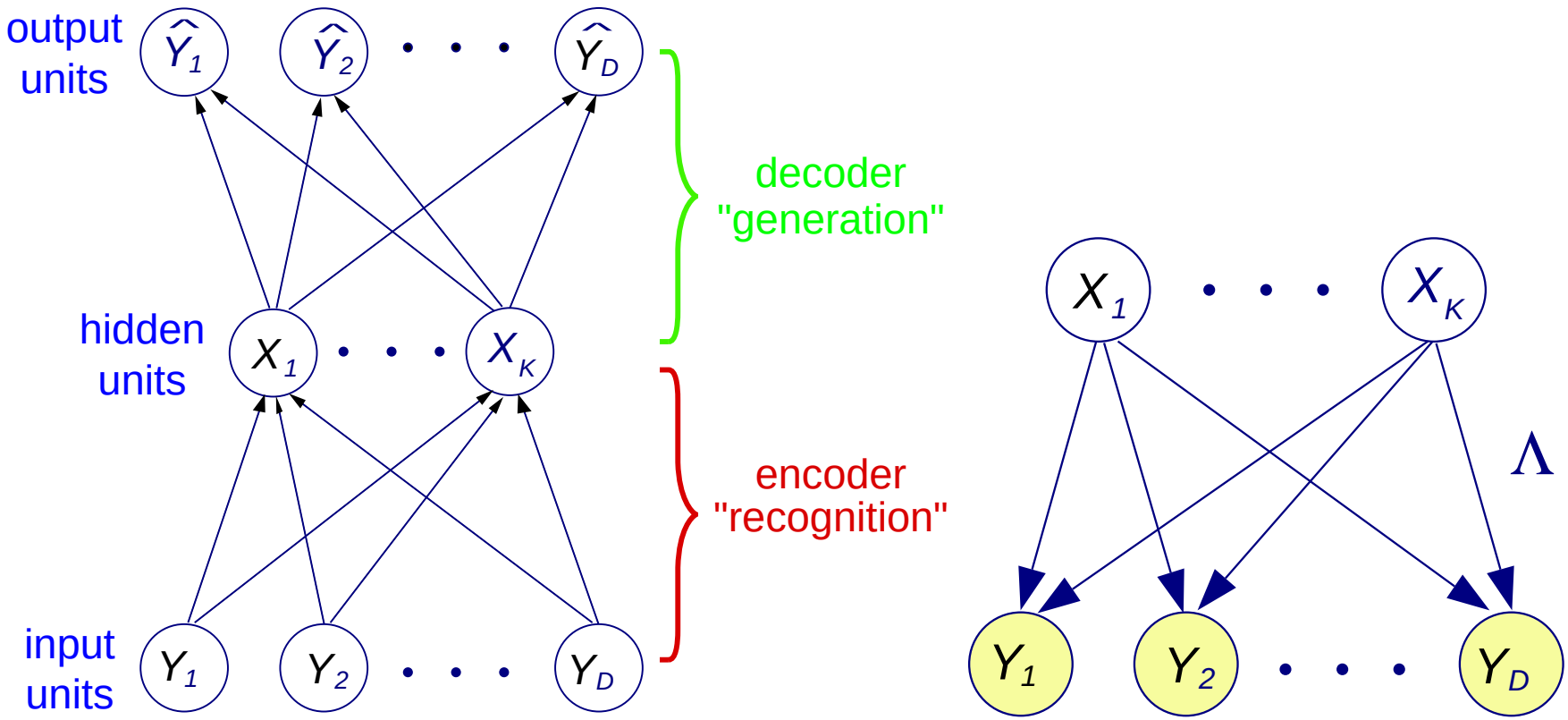
Deep belief networks are a powerful way to model complex probability distributions. However, learning the structure of a belief network, particularly one with hidden units, is difficult. The Indian buffet process has been used as a nonparametric Bayesian prior on the directed structure of a belief network with a single infinitely wide hidden layer. In this paper, we introduce the cascading Indian buffet process (CIBP), which provides a nonparametric prior on the structure of a layered, directed belief network that is unbounded in both depth and width, yet allows tractable inference. We use the CIBP prior with the nonlinear Gaussian belief network so each unit can additionally vary its behavior between discrete and continuous representations. We provide Markov chain Monte Carlo algorithms for inference in these belief networks and explore the structures learned on several image data sets.

Autoencoders

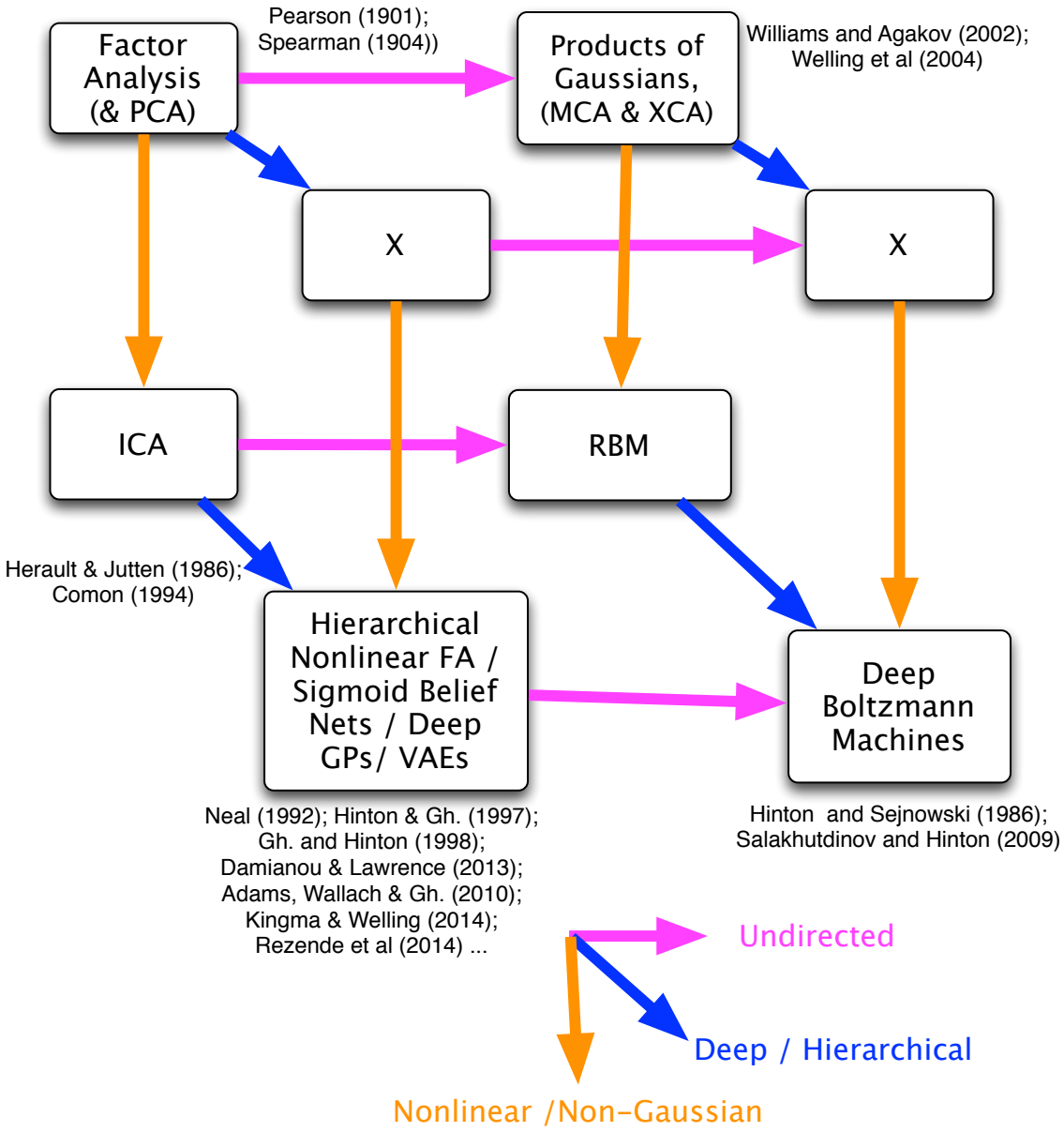


(my figure from 1998 - ideas from mid-1980s)

Autoencoders and Factor Analysis



Latent Factor Models, RBMs, Variational Autoencoders, etc



VAE = nonlinear factor analysis + recognition model + variational inference + sampling

A Note on Generative Adversarial Networks (GANs)

Assume a generative model that maps from a vector of latent variables $\mathbf{x_n}$ to observed variables $\mathbf{y_n}$, through a *nonlinear* function, implemented as a neural network with parameters (weights) θ.

The likelihood is intractable:

$$p(D_{\text{train}}|\theta) = \prod_n p(\mathbf{y}_n|\theta) = \prod_n \int p(\mathbf{y}_n|\mathbf{x}_n, \theta) p(\mathbf{x}_n) d\mathbf{x}_n$$

We can use many methods to approximate the likelihood (variational inference, MCMC, EP, VAEs, etc). Alternatively, since we can **sample** data D_{sampled} from this model, we can use a **two sample test** to train the model:

test if D_{sampled} and D_{train} are drawn from the same distribution.

A GAN is nonlinear factor analysis, trained using two sample test, as a surrogate loss function, where the test is implemented with a discriminative model.

(see also other two-sample-test surrogates for the loss function, e.g. MMD (Dziugaite, Roy & Ghahramani, 2015; Li, Swersky, & Zemel, 2015))

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" $\rightarrow$ "yellow teeth"
- "yellow teeth" $\rightarrow$ "smoking"

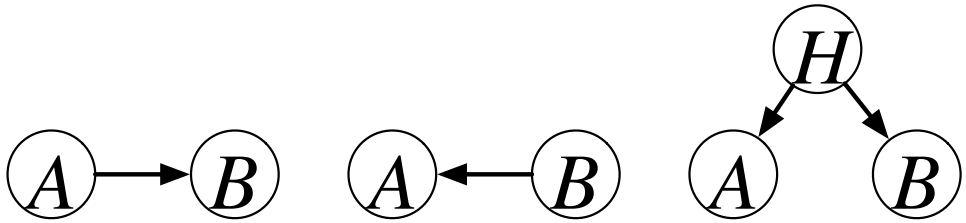
Key idea: interventions and the do-calculus:

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

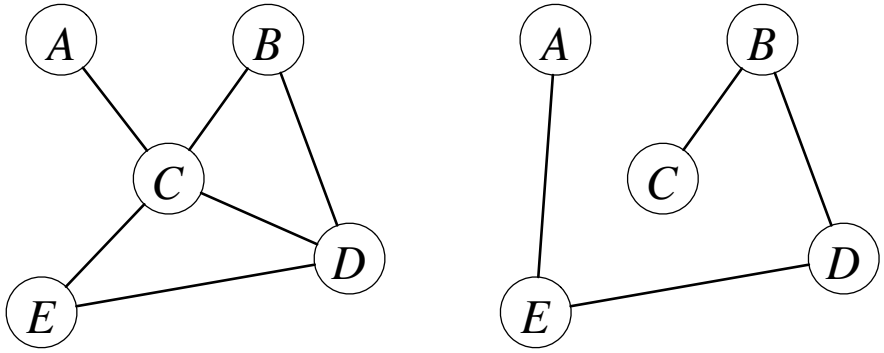
$$p(Y|S = s) = p(Y|\text{do}(S = s))$$

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



$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_j g_j(\mathbf{x}_{C_j}; \boldsymbol{\theta}_j)$ where $Z(\boldsymbol{\theta}) = \sum_{\mathbf{x}} \prod_j g_j(\mathbf{x}_{C_j}; \boldsymbol{\theta}_j)$.

Problem: computing $Z(\boldsymbol{\theta})$ 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:

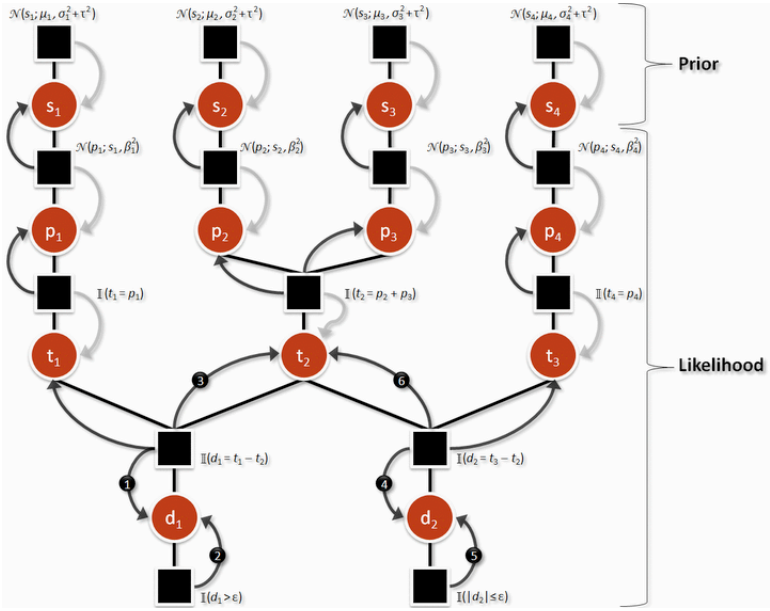
- directly approximate $Z(\boldsymbol{\theta})$ and/or its derivatives (cf. Boltzmann machine learning; contrastive divergence; pseudo-likelihood)
- use approx inference methods (e.g. loopy belief propagation, bounding methods, EP).

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

Scaling Bayesian Methods

Case Studies:

- Microsoft XBox Live TrueSkill
- Microsoft AdPredictor
- Netflix Bayesian PMF



Approaches:

- Approximate inference
- Parallel (MPI) and cloud / distributed (Hadoop, MapReduce) data and inference
- Subsample data

Summary

- Probabilistic modelling and Bayesian inference are two sides of the same coin
- Bayesian machine learning treats learning as a probabilistic inference problem
- Themes:
  - **Graphical models:** an intuitive and computationally useful representation for probabilistic modelling.

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

Readings and References

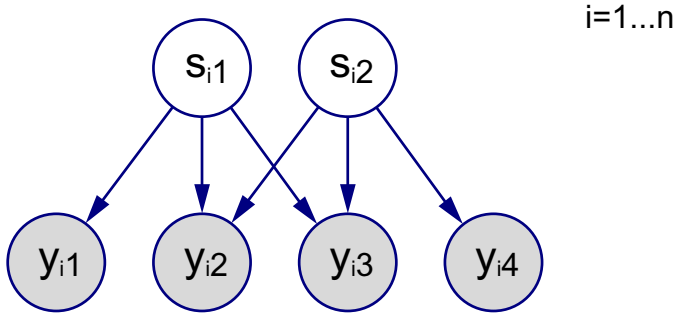
- Beal, M.J. and Ghahramani, Z. (2006) Variational Bayesian learning of directed graphical models with hidden variables. *Bayesian Analysis* 1(4):793–832.
  http://learning.eng.cam.ac.uk/zoubin/papers/BeaGha06.pdf
- Friedman, N. (1998) The Bayesian structural EM algorithm. In *Uncertainty in Artificial Intelligence (UAI-1998)*. http://robotics.stanford.edu/ nir/Papers/Fr2.pdf
- Ghahramani, Z. (2004) Unsupervised Learning. In Bousquet, O., von Luxburg, U. and Raetsch, G. *Advanced Lectures in Machine Learning*. 72-112.
  http://learning.eng.cam.ac.uk/zoubin/papers/ul.pdf
- Heckerman, D. (1995) A tutorial on learning with Bayesian networks. In *Learning in Graphical Models*.
  http://research.microsoft.com/pubs/69588/tr-95-06.pdf
- Koller, D. and Friedman, N. (2009) Probabilistic Graphical Models: Principles and Techniques. MIT Press.
- Wood, F., Griffiths, T.L. and Ghahramani, Z. (2006) A Non-Parametric Bayesian Method for Inferring Hidden Causes. In *Uncertainty in Artificial Intelligence (UAI-2006)*, 536–543.

Appendix

Variational Bayesian Learning of Graph Structures

A case study for discrete directed graphs

- Bipartite structure: only hidden variables can be parents of observed variables.
- **Two** binary hidden variables, and **four** five-valued discrete observed variables.



- Conjugate prior is Dirichlet, Conjugate-Exponential model, so VB-EM algorithm is a straightforward modification of EM.

- **Experiment:** There are 136 distinct structures (out of 256) with 2 latent variables as potential parents of 4 conditionally independent observed vars.

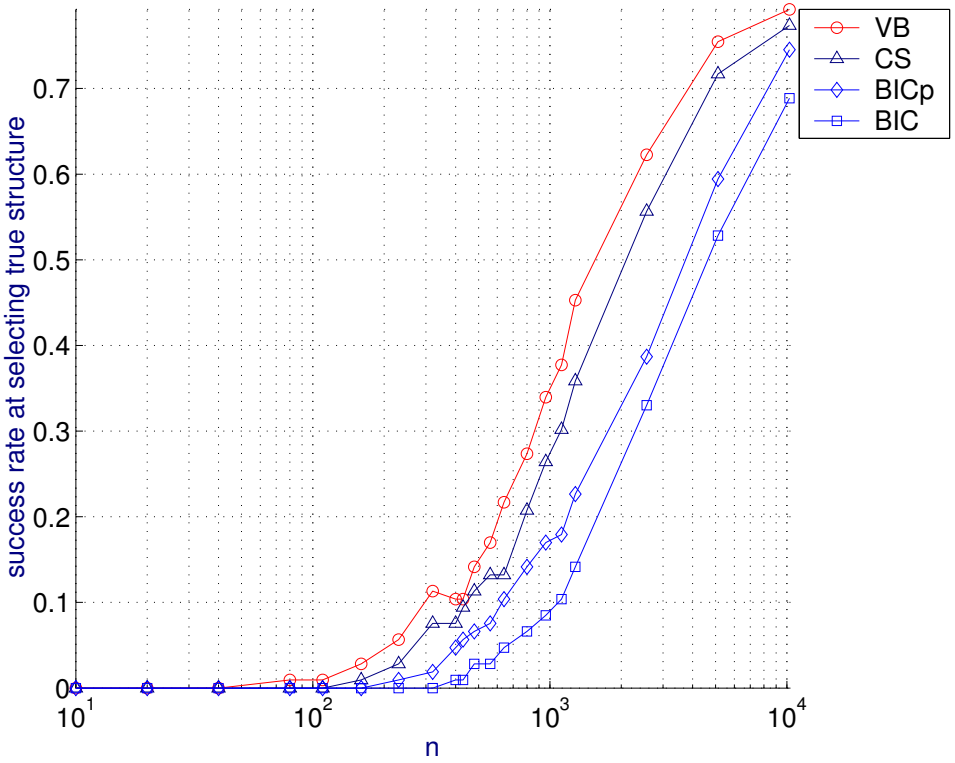
- **Score** each structure for twenty varying size data sets:

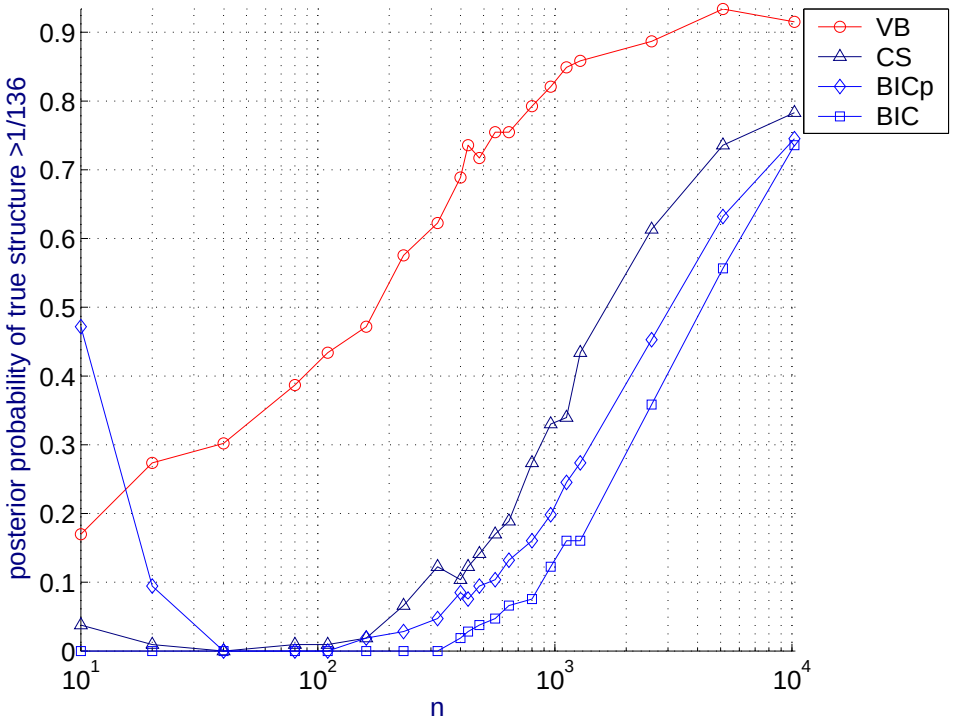
$n \in \{10, 20, 40, 80, 110, 160, 230, 320, 400, 430, 480, 560, 640, 800, 960, 1120, 1280, 2560, 5120, 10240\}$

using 3 methods: BIC, VB, and a **gold standard** Annealed Importance Sampling AIS

- 2720 graph scores computed, times for each: BIC (1.5s), VB (4s), AIS (400s).

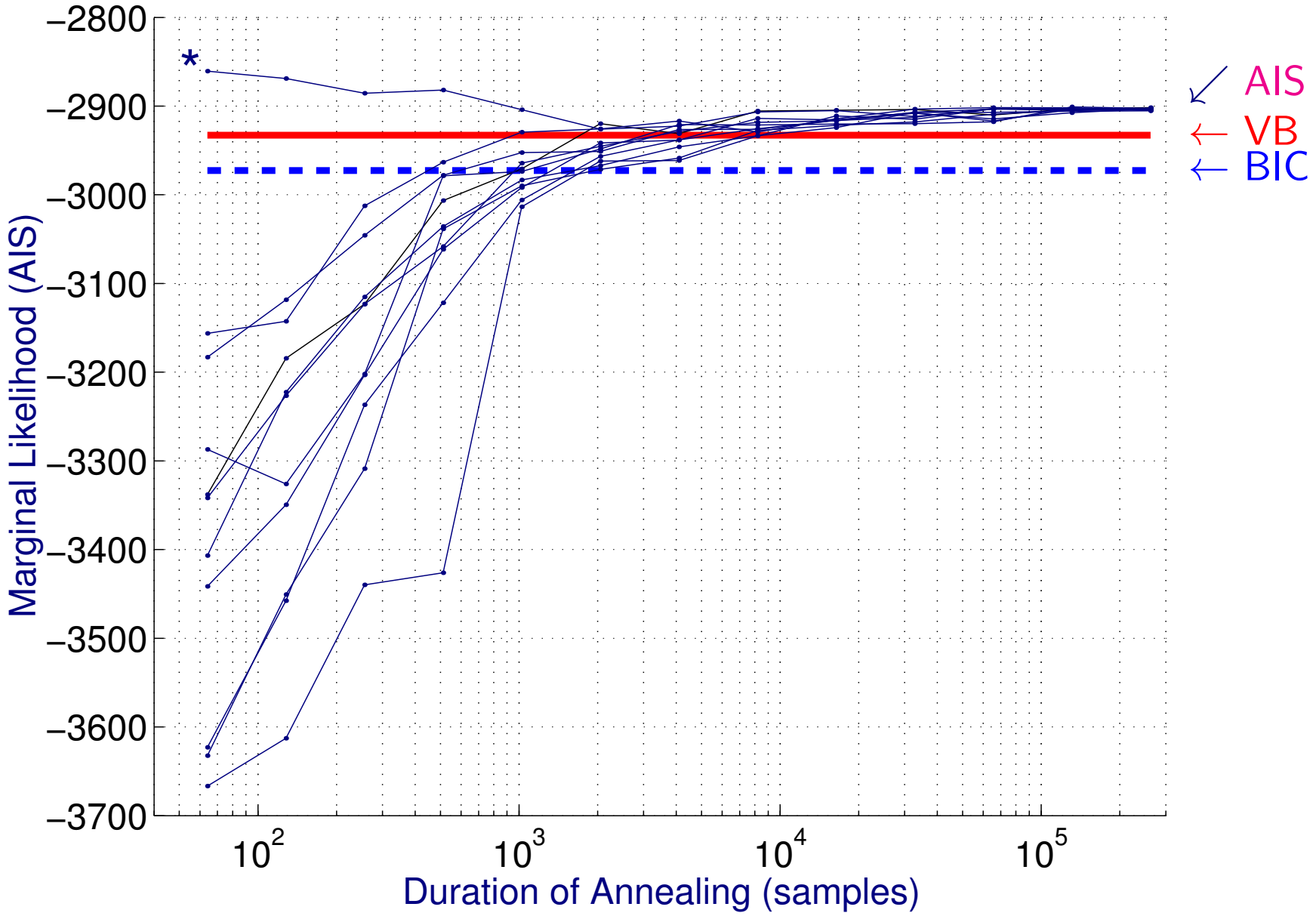
Results, averaged over about 100 parameter draws





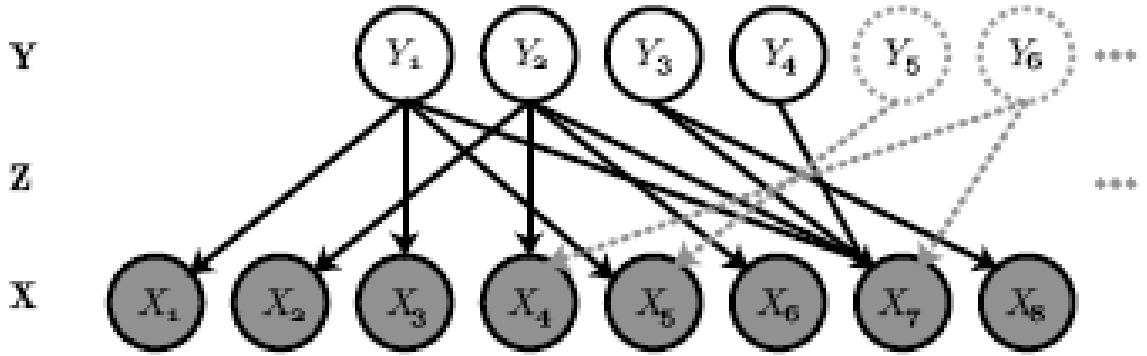
VB is also more accurate than Cheeseman-Stutz (CS) approximation to the marginal likelihood. In fact we can prove than VB $\geq$ CS (Beal and Ghahramani, *Bayesian Analysis*, 2006).

How tight is VB bound?



About 10^4 sweeps of sampling needed to achieve VB lower bound.

How many latent variables should there be?



$\mathbf{Y}$ - latent factors (e.g. diseases)
$\mathbf{Z}$ - graph structure (binary adjacency matrix)
$\mathbf{X}$ - observed binary features (e.g. symptoms)

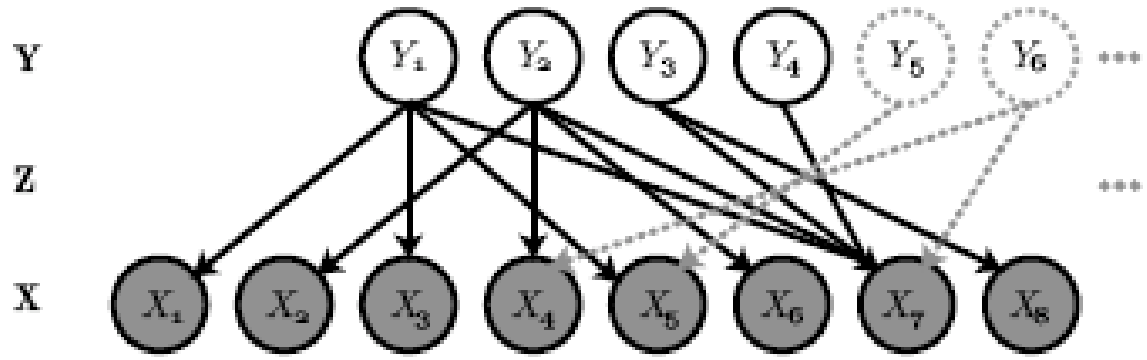
Solution 1: Do model comparison for $m = 1$, $m = 2$, ...

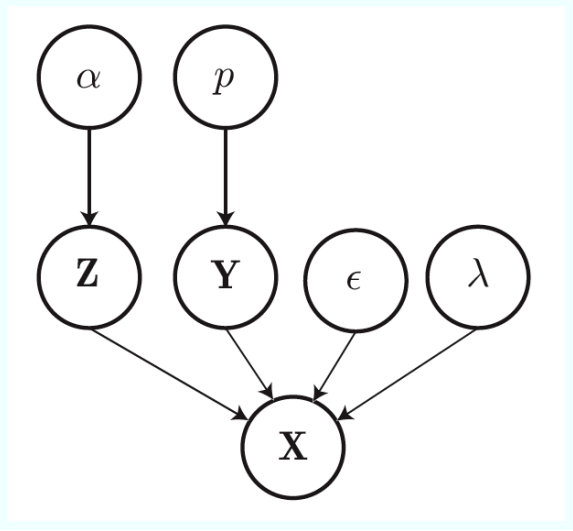
Solution 2: Assume potentially $m = \infty$ of which we only observe a finite number.

Note: this is analogous to the question of how many mixture components to use (model selection for finite mixture model vs infinite mixture model using Dirichlet processs mixtures).

Graphical models with infinitely many latent variables

"A Non-Parametric Bayesian Method for Inferring Hidden Causes" (Frank Wood, Tom Griffiths, & Ghahramani, *Uncertainty in Artificial Intelligence*, 2006)





$\mathbf{Y}$ - binary latent factors (diseases)
$\mathbf{Z}$ - graph structure
$\mathbf{X}$ - observed binary features (symptoms)

"Noisy-or" observations: $P(x_{it} = 1|\mathbf{Z}, \mathbf{Y}, \lambda, \epsilon) = 1 - (1 - \lambda)^{\sum_k z_{ik} y_{kt}}(1 - \epsilon)$

What should we use as $P(\mathbf{Z})$?

The matrix $\mathbf{Z}$ is a binary matrix of size (N = number of observed variables) $\times$ (K = number of latent variables).

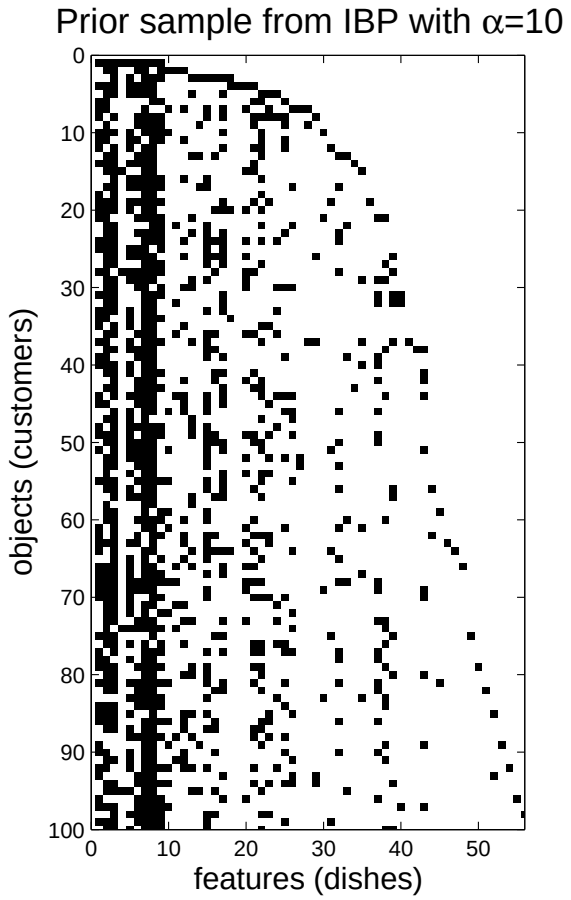
But $K \to \infty$.

We can define a consistent distribution over such infinite sparse binary matrices using the "Indian Buffet Process" (IBP) (cf Chinese restaurant process, Aldous 1985; Pitman 2002).

A sample from prior shown on right.

Note "rich get richer" property.

We can derive a Gibbs sampler for this model.



Graphical models with infinitely many latent variables

Gibbs sampling traces

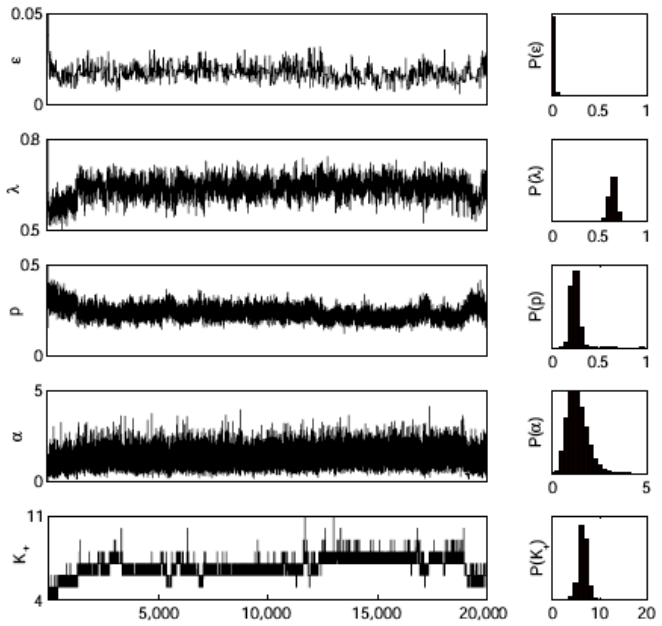


Figure 5: Trace plots and histograms for the Gibbs sampler applied to the signs exhibited by 50 stroke patients. The left column shows the current value of $\epsilon, \lambda, p, \alpha$, and K_+ as the sampler progressed, where K_+ is obtained by examining the current $\mathbf{Z}$ sample. The right column shows histograms of the same variables computed over the samples.

Comparison to RJMCMC

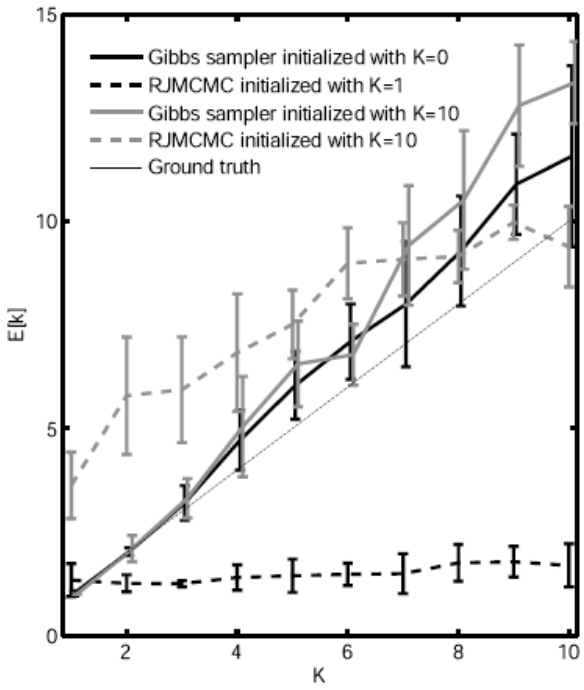


Figure 3: Learning the number of hidden causes using both RJMCMC and Gibbs sampling. Each line show the mean and standard deviation of the expected value of the dimensionality of the model (K for RJMCMC, and K_+ for Gibbs) taken over 500 iterations of sampling for each of 10 datasets.

Seems to work reliably, and mixed better than RJMCMC.

Graphical models with infinitely many latent variables

(with Frank Wood and Tom Griffiths)
Inferring stroke localization from patient symptoms:

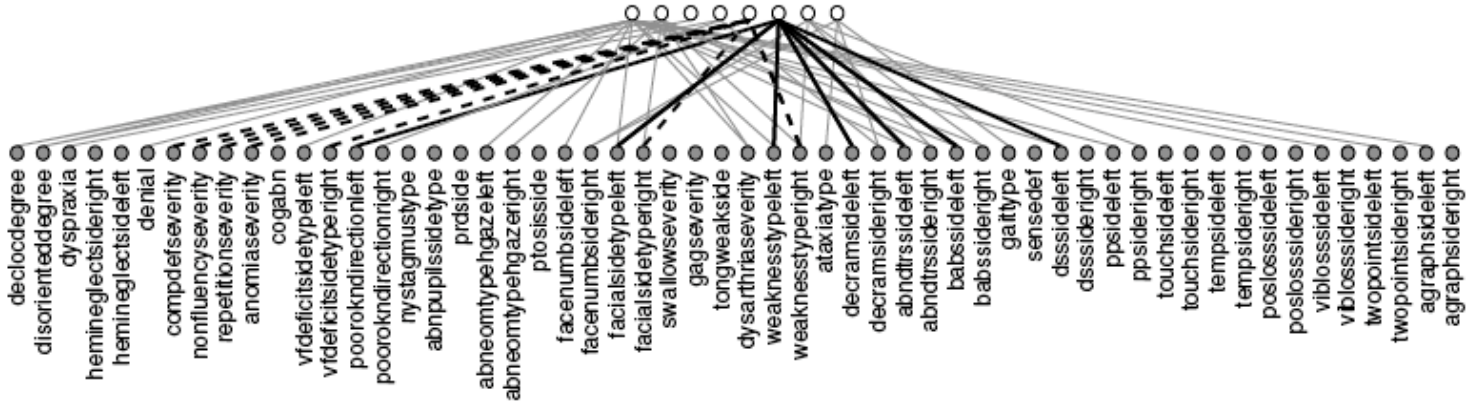
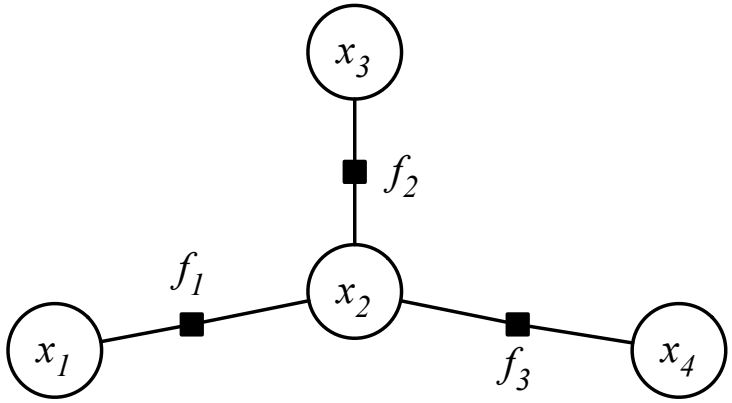


Figure 6: Causal structure with highest posterior probability. Two grouping of signs are highlighted. In solid black, we find a grouping of poor optokinetic nystagmus, lack of facial control, weakness, decreased rapid alternating movements, abnormal deep tendon reflexes, Babinski sign, and double simultaneous stimulation neglect, all on the left side, consistent with a right frontal/parietal infarct. In dashed black, we find a grouping of comprehension deficit, non-fluency, repetition, anomia, visual field deficit, facial weakness, and general weakness, with the latter three on the right side, generally consistent with a left temporal infarct.

(50 stroke patients, 56 symptoms/signs)

Propagation in Factor Graphs



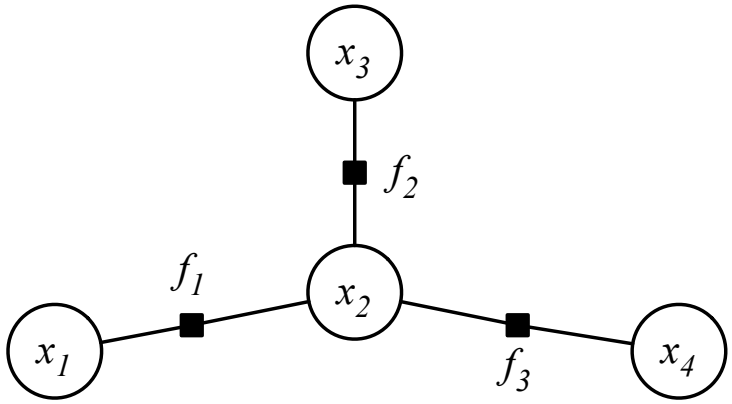
initialise all messages to be constant functions

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

message direction	message value
$x_1 \to f_1$	$1(x_1)$
$x_3 \to f_2$	$1(x_3)$
$f_1 \to x_2$	$\sum_{x_1} f_1(x_1, x_2) 1(x_1)$
$f_2 \to x_2$	$\sum_{x_3} f_2(x_3, x_2) 1(x_3)$
$x_2 \to f_3$	$\left(\sum_{x_1} f_1(x_1, x_2)\right) \left(\sum_{x_3} f_2(x_3, x_2)\right)$
$f_3 \to x_4$	$\sum_{x_2} f_3(x_2, x_4) \left(\sum_{x_1} f_1(x_1, x_2)\right) \left(\sum_{x_3} f_2(x_3, x_2)\right)$

where $1(x)$ is a constant uniform function of x

Propagation in Factor Graphs



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

message direction	message value
$x_1 \to f_1$	$\delta(x_1 = a)$
$x_3 \to f_2$	$1(x_3)$
$f_1 \to x_2$	$\sum_{x_1} f_1(x_1, x_2)\delta(x_1 = a) = f_1(x_1 = a, x_2)$
$f_2 \to x_2$	$\sum_{x_3} f_2(x_3, x_2)1(x_3)$
$x_2 \to f_3$	$f_1(x_1 = a, x_2)\left(\sum_{x_3} f_2(x_3, x_2)\right)$
$f_3 \to x_4$	$\sum_{x_2} f_3(x_2, x_4) f_1(x_1 = a, x_2)\left(\sum_{x_3} f_2(x_3, x_2)\right)$

where $\delta(x = a)$ is a delta function