Unsupervised Learning

Lecture 6: Hierarchical and Nonlinear Models

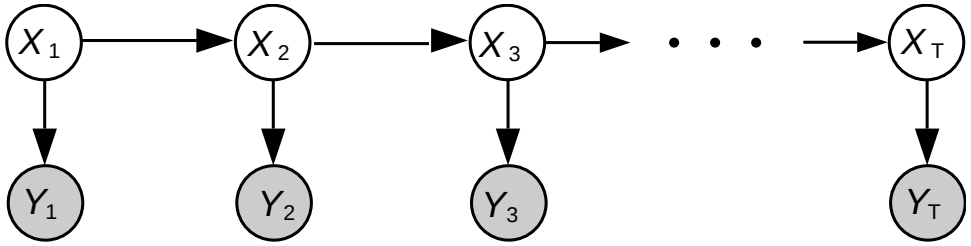
Zoubin Ghahramani
zoubin@gatsby.ucl.ac.uk

Gatsby Computational Neuroscience Unit, and
MSc in Intelligent Systems, Dept Computer Science
University College London

Term 1, Autumn 2005

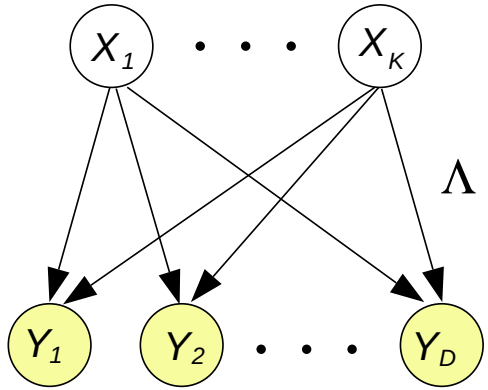
Why we need nonlinearities

Linear systems have limited modelling capability.



Consider linear-Gaussian state-space models: *Only certain dynamics can be modelled.*

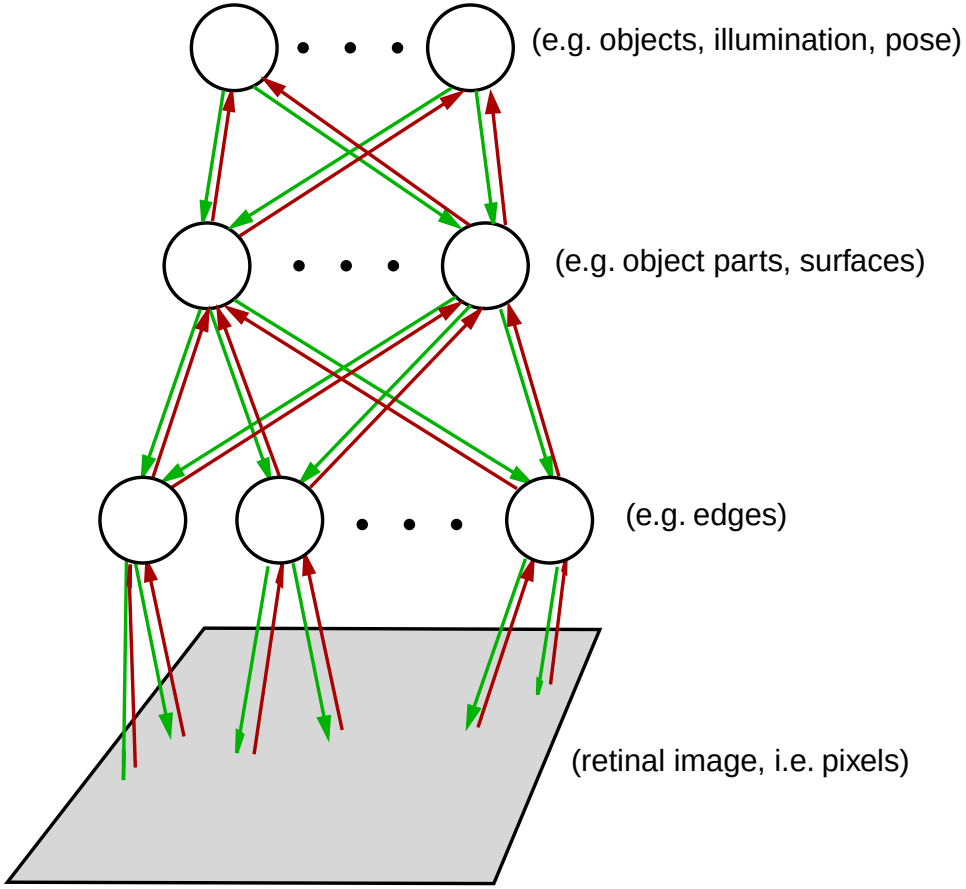
Factor analysis does not model non-Gaussian data well.



Data on a Nonlinear Manifold

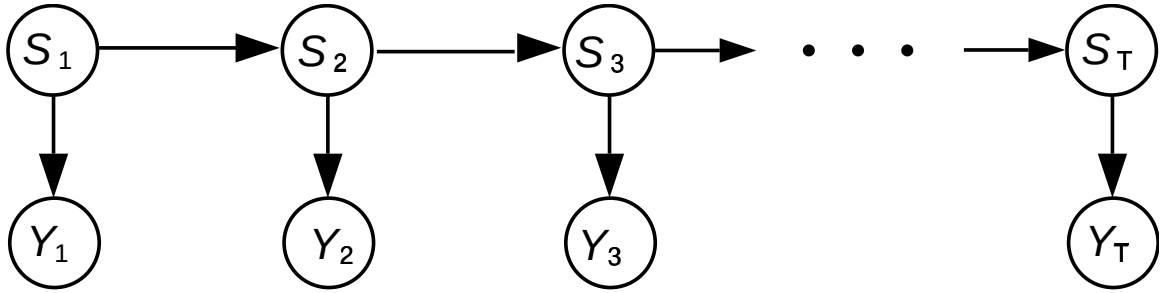
Why we need hierarchical models

Many generative processes can be naturally described at different levels of detail.



Biology seems to have developed hierarchical representations.

Why we need distributed representations



Consider a hidden Markov model. *To capture N bits of information about the history of the sequence, an HMM requires $K = 2^N$ states!*

In a *distributed representation* each data point is represented by a vector of (discrete or continous) attibutes. These attributed might be *latent* (i.e. hidden).

> For example, you could cluster American voters into `Republican`, `Democrat`, `Independent` and `Undecided`, but this is **not** a distributed representation since each person is described by a single 4-valued discrete variable. A distributed representation would be: (`Democrat`, `Single`, `Black`, `Female`, `18-35 years old`, `City-dweller`)

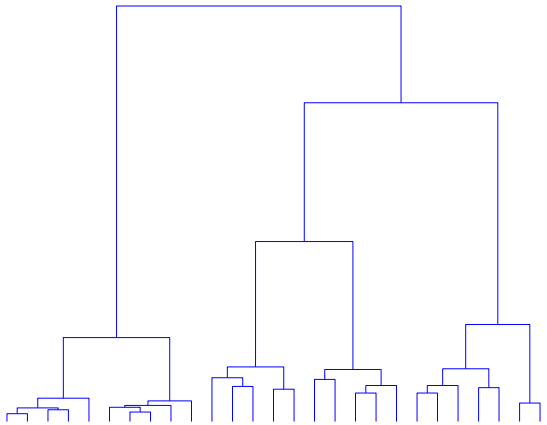
Some more complex generative unsupervised learning methods

- Hierarchical clustering: clustering algorithms in which clusters are organised hierarchically
- Nonlinear dimensionality reduction methods: nonlinear generalizations of PCA and factor analysis
- Factorial hidden Markov models and dynamic Bayesian networks: time series models with distributed representations
- Nonlinear dynamical systems
- Independent components analysis (ICA): linear factor models with non-Gaussian factors
- Boltzmann machines: undirected model for binary data with binary latent variables
- Sigmoid Belief networks: directed (neural-netork-like) model for binary data

Hierarchical Clustering

(e.g. Duda and Hart, 1973)

Data $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$
Initialise number of clusters $c = n$
Initialise $\mathcal{D}_i = \{\mathbf{x}^{(i)}\}$ for $i = 1, \ldots, c$
while $c > 1$ **do**
  Find nearest pair of clusters $\mathcal{D}_i$ and $\mathcal{D}_j$
  Merge $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \mathcal{D}_j$, Delete $\mathcal{D}_j$, $c \leftarrow c - 1$
end while



Distance Measures:

$$
\begin{aligned}
d_{\min}(\mathcal{D}_i, \mathcal{D}_j) &= \min_{\mathbf{x} \in \mathcal{D}_i, \mathbf{x}' \in \mathcal{D}_j} \|\mathbf{x} - \mathbf{x}'\| && \text{nearest neighbour} \\
d_{\max}(\mathcal{D}_i, \mathcal{D}_j) &= \max_{\mathbf{x} \in \mathcal{D}_i, \mathbf{x}' \in \mathcal{D}_j} \|\mathbf{x} - \mathbf{x}'\| && \text{furthest neighbour} \\
d_{\text{avg}}(\mathcal{D}_i, \mathcal{D}_j) &= \frac{1}{n_i n_j} \sum_{\mathbf{x} \in \mathcal{D}_i} \sum_{\mathbf{x}' \in \mathcal{D}_j} \|\mathbf{x} - \mathbf{x}'\| && \text{average distance} \\
d_{\text{mean}}(\mathcal{D}_i, \mathcal{D}_j) &= \|\mathbf{m}_i - \mathbf{m}_j\| && \text{centre of mass}
\end{aligned}
$$

Hierarchical clustering is very widely used, e.g. in bioinformatics, because it is often natural to think of data points at multiple level of granularity, or as having been generated by an evolutionary process.

Nonlinear Dimensionality Reduction

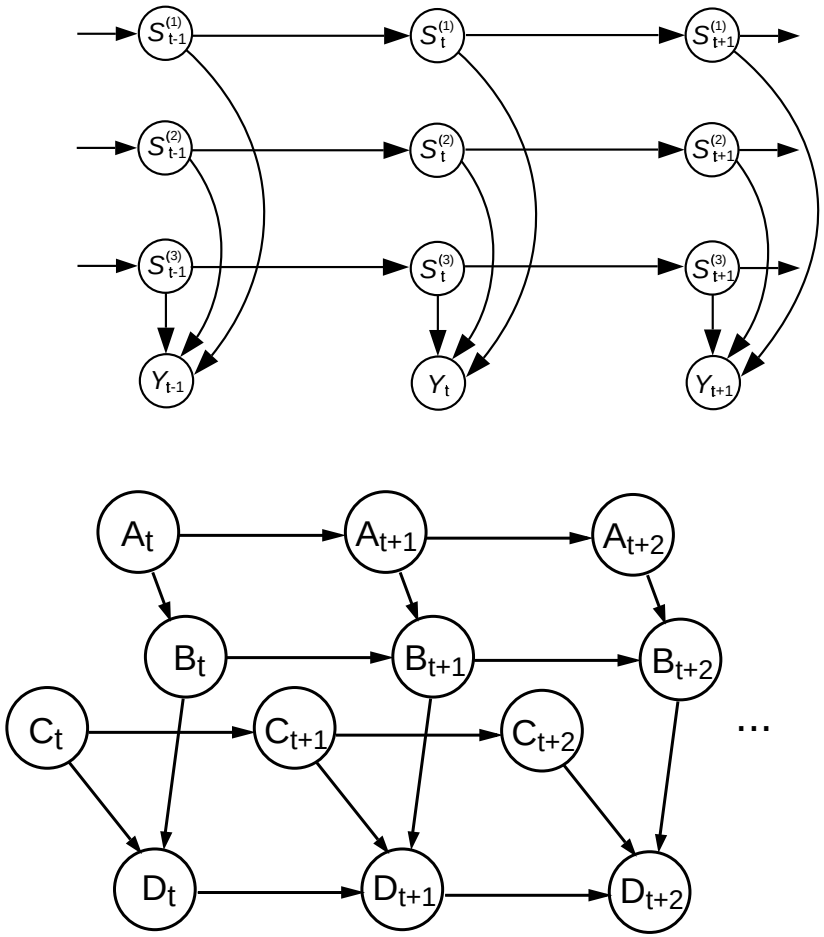
There are many ways of generalising PCA and FA models to deal with data which lies on a nonlinear manifold:

- Principal curves
- Autoencoders
- Generative topographic mappings (GTM) and Kohonen self-organising maps (SOM)
- Density networks
- Multi-dimensional scaling (MDS)
- Isomap
- Locally linear embedding (LLE)

Nonlinear Manifold of Hand Gestures

Unfortunately, we don't have time to cover these methods in the course...

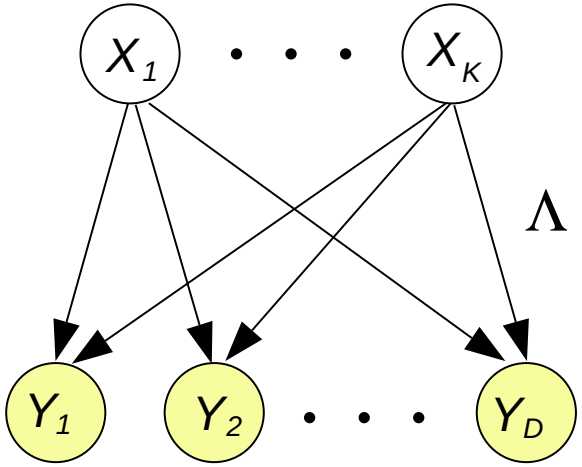
Factorial Hidden Markov Models and Dynamic Bayesian Networks



These are hidden Markov models with many hidden state variables (i.e. a *distributed representation* of the state).

Blind Source Separation

Independent Components Analysis



- Just like Factor Analysis but $P(x_k)$ is *non-Gaussian*:

$$y_d = \sum_{k=1}^{K} \Lambda_{dk}\, x_k + \epsilon_d$$

- Equivalently we can create non-Gaussian factors by passing Gaussian factors through an elementwise nonlineary, e.g. $x_k = g(z_k)$.

- The special case of $K = D$, and observation noise assumed to be zero has been studied extensively (standard ICA):

$$\mathbf{y} = \Lambda\mathbf{x} \quad \text{which implies} \quad \mathbf{x} = W\mathbf{y} \quad \text{where} \quad W = \Lambda^{-1}$$

  where $\mathbf{x}$ are the independent components (factors), $\mathbf{y}$ are the observations, and W is the unmixing matrix. See: `http://www.cnl.salk.edu/~tony/ica.html`

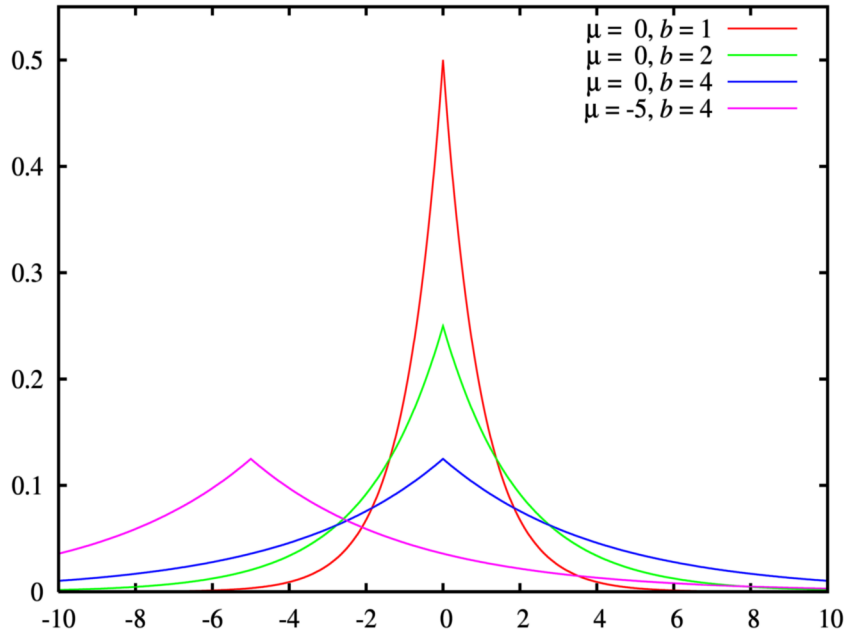
- Inference and learning are easy in standard ICA. Many extensions are possible.

Kurtosis

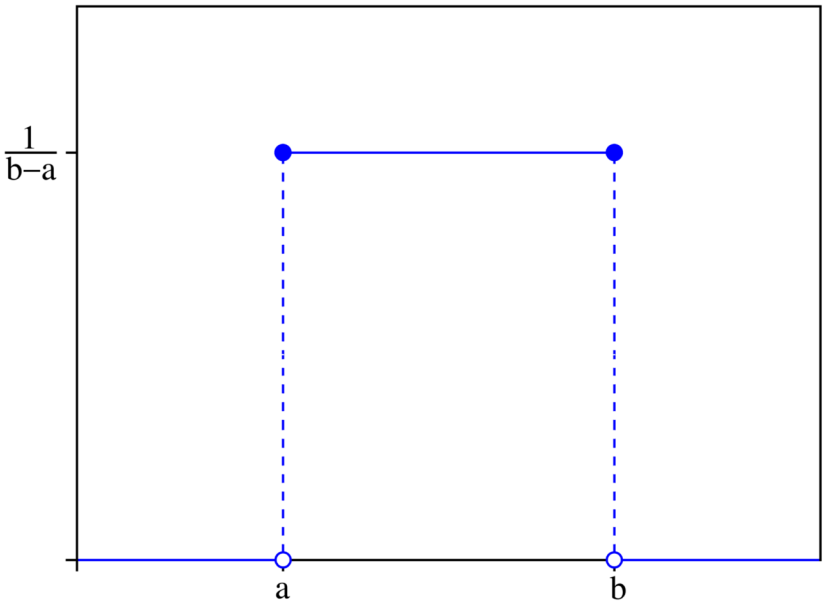
The kurtosis (or excess kurtosis) measures how "peaky" or "heavy-tailed" a distribution is.

$$K = \frac{E((x-\mu)^4)}{E((x-\mu)^2)^2} - 3$$

where $\mu = E(x)$ is the mean of x.
Gaussian distributions have zero kurtosis.



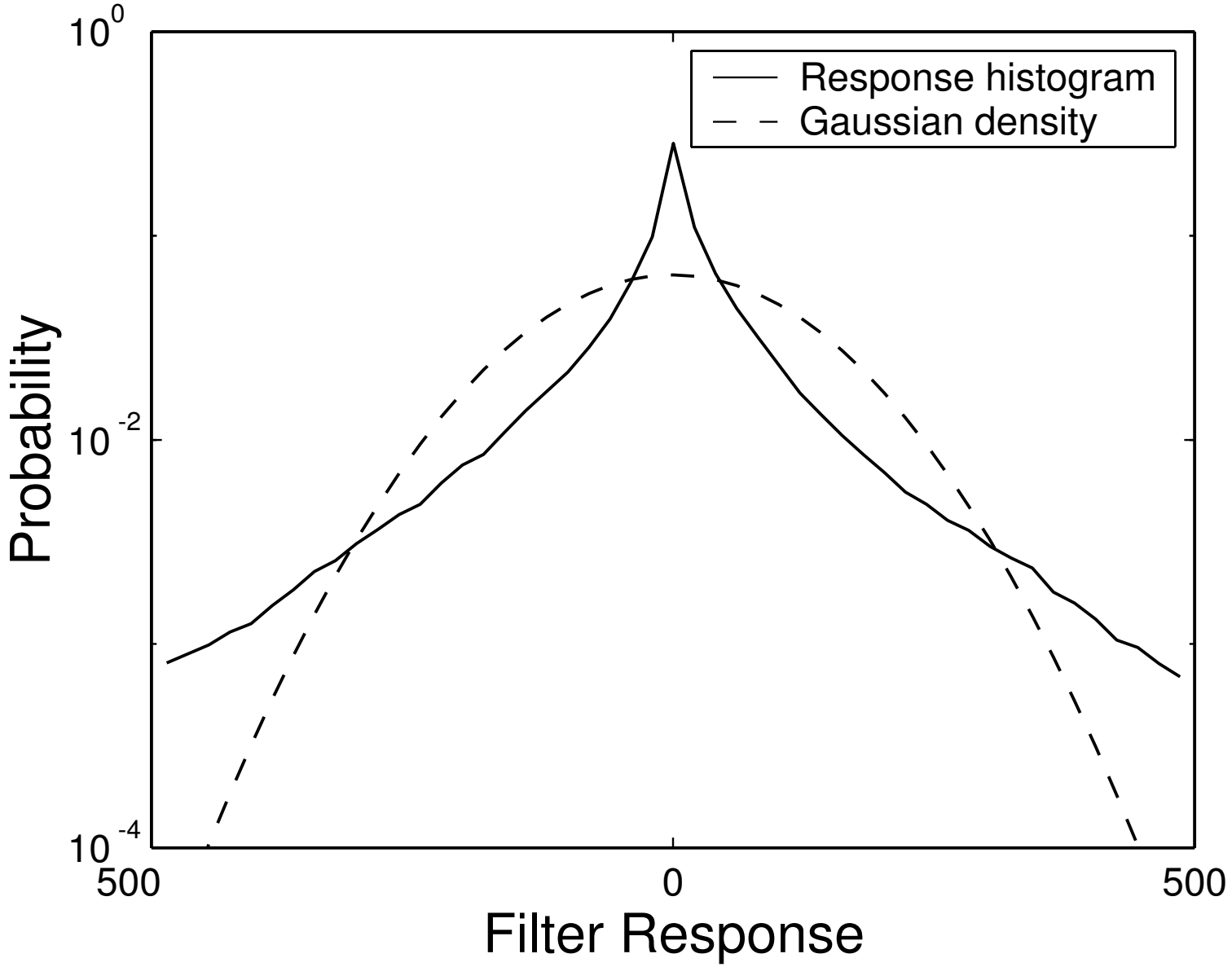
Heavy tailed distributions have positive kurtosis (leptokurtic).



Light tailed distributions have negative kurtosis (platykurtic).

Why are heavy-tailed distributions interesting?

Natural Scenes and Sounds



Natural Scenes

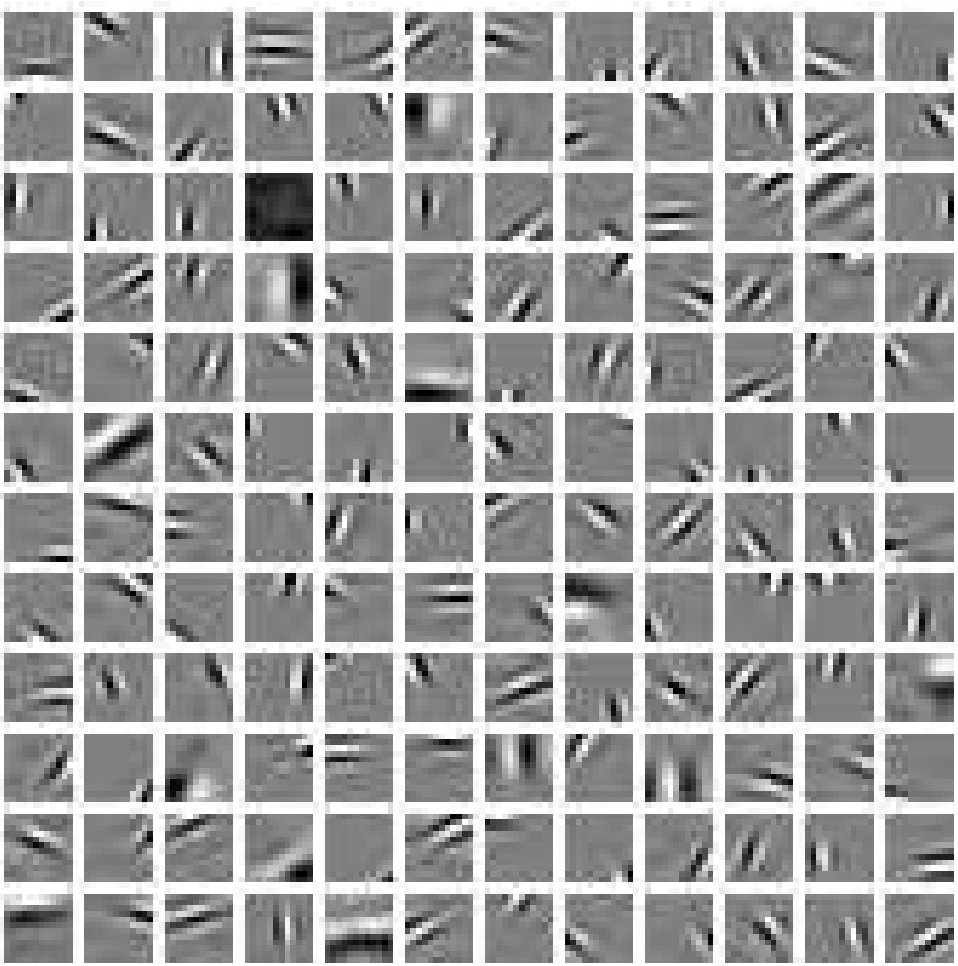


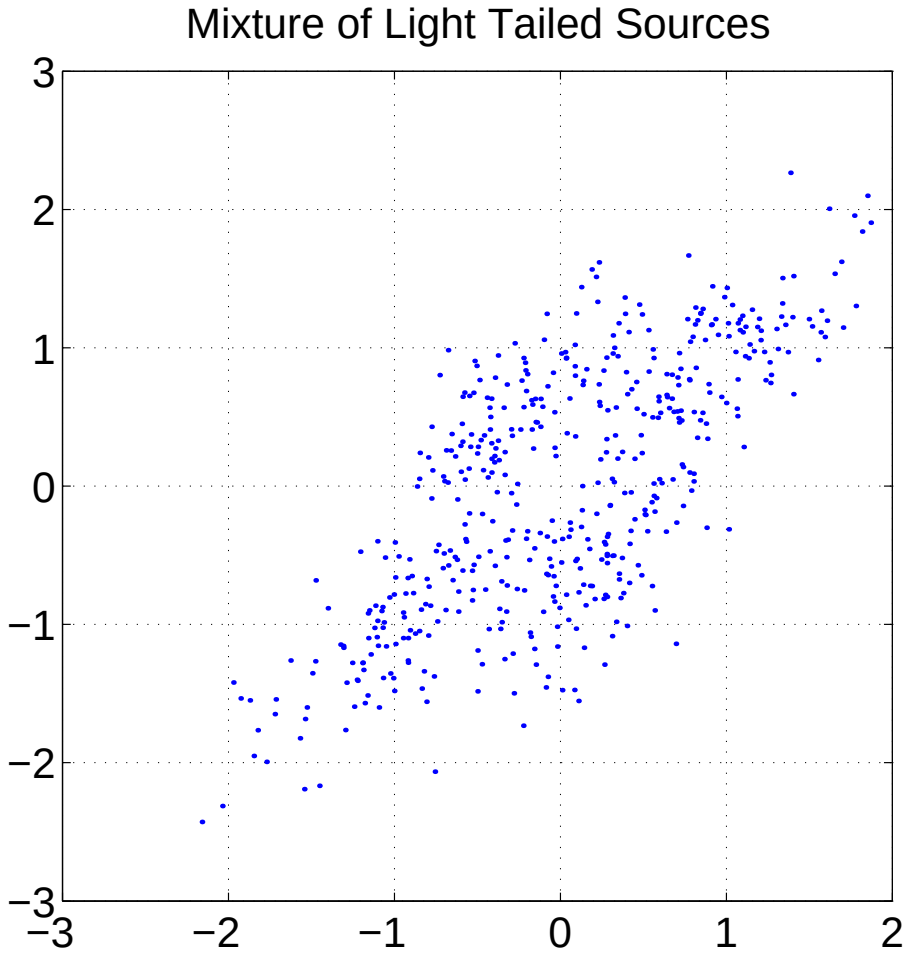
Figure 7: Example basis functions derived using sparseness criterion see (Olshausen & Field 1996).

Applications of ICA and Related Methods

- Separating auditory sources
- Analysis of EEG data
- Analysis of functional MRI data
- Natural scene analysis
- ...

ICA: The magic of unmixing

ICA (with heavy tailed noise) tries to find the directions with outliers.



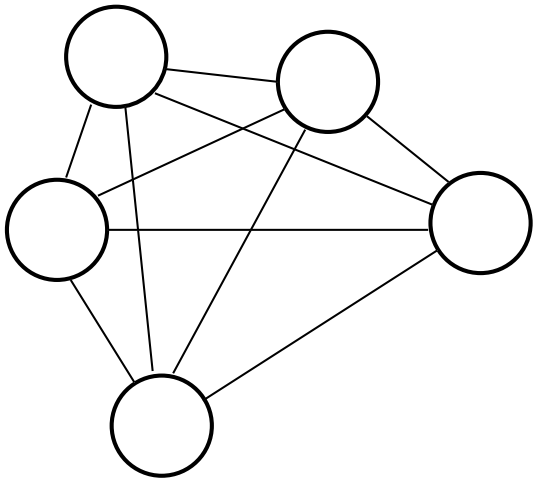
How ICA Relates to Factor Analysis and Other Models

- **Factor Analysis (FA)**: Assumes the factors are Gaussian.
- **Principal Components Analysis (PCA)**: Assumes no noise on the observations: $\Psi = \lim_{\epsilon \to 0} \epsilon I$
- **Independent Components Analysis (ICA)**: Assumes the factors are non-Gaussian (and no noise).
- **Mixture of Gaussians**: A single discrete-valued "factor": $x_k = 1$ and $x_j = 0$ for all $j \neq k$.
- **Mixture of Factor Analysers**: Assumes the data has several clusters, each of which is modeled by a single factor analyser.
- **Linear Dynamical Systems**: Time series model in which the factor at time t depends linearly on the factor at time $t - 1$, with Gaussian noise.

Extensions of ICA

- Fewer or more sources than "microphones" ($K \neq D$) – e.g. Lewicki and Sejnowski (1998).
- Allow noise on microphones
- Time series versions with convolution by linear filter
- Time-varying mixing matrix
- Discovering number of sources

Boltzmann Machines



Undirected graphical model (i.e. a Markov network) over a vector of binary variables $s_i \in \{0, 1\}$. Some variables may be hidden, some may be visible (observed).

$$P(\mathbf{s}|W, \mathbf{b}) = \frac{1}{Z} \exp\left\{ \sum_{ij} W_{ij} s_i s_j - \sum_i b_i s_i \right\}$$

where Z is the normalization constant (partition function).

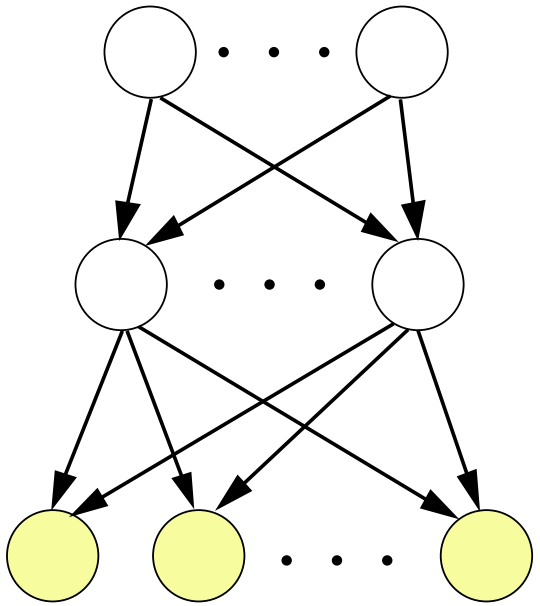
Learning algorithm: a gradient version of EM

- E step involves computing averages w.r.t. $P(\mathbf{s}_H|\mathbf{s}_V, W, \mathbf{b})$ ("clamped phase"). This could be done via a propagation algorithm or (more usually) an approximate method such as Gibbs sampling.

- The M step requires gradients w.r.t. Z, which can be computed by averages w.r.t. $P(\mathbf{s}|W, \mathbf{b})$ ("unclamped phase").

$$\Delta W_{ij} = \eta[\langle s_i s_j \rangle_c - \langle s_i s_j \rangle_u]$$

Sigmoid Belief Networks



Directed graphical model (i.e. a Bayesian network) over a vector of binary variables $s_i \in \{0, 1\}$.

$$P(\mathbf{s}|W, \mathbf{b}) = \prod_i P(s_i|\{s_j\}_{j<i}, W, \mathbf{b})$$

$$P(s_i = 1|\{s_j\}_{j<i}, W, \mathbf{b}) = \frac{1}{1 + \exp\{-\sum_{j<i} W_{ij} s_j - b_i\}}$$

A probabilistic version of sigmoid multilayer perceptrons ("neural networks").

Learning algorithm: a gradient version of EM

- E step involves computing averages w.r.t. $P(\mathbf{s}_H|\mathbf{s}_V, W, \mathbf{b})$. This could be done via the Belief Propagation algorithm (if singly connected) or (more usually) an approximate method such as Gibbs sampling or mean field (see later lectures).

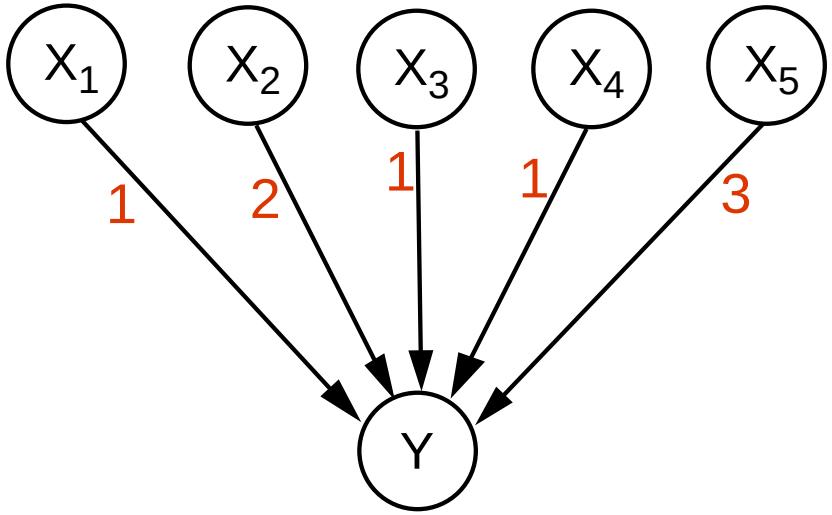
- Unlike Boltzmann machines, there is no partition function, so no need for an unclamped phase in the M step.

Intractability

For many probabilistic models of interest, exact inference is not computationally feasible. This occurs for two (main) reasons:

- distributions may have complicated forms (non-linearities in generative model)

- "explaining away" causes coupling from observations
  observing the value of a child induces dependencies amongst its parents (high order interactions)



$$Y = X_1 + 2\,X_2 + X_3 + X_4 + 3\,X_5$$

We can still work with such models by using *approximate inference* techniques to estimate the latent variables.

Approximate Inference

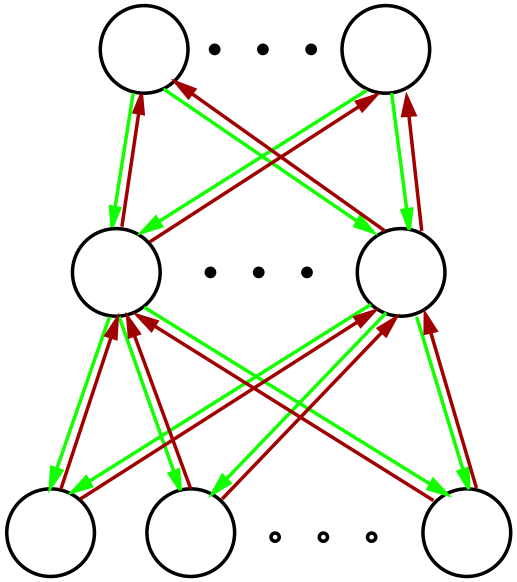
- **Sampling**: Approximate posterior distribution over hidden variables by a set of random samples. We often need **Markov chain Monte carlo** methods to sample from difficult distributions.

- **Linearization**: Approximate nonlinearities by Taylor series expansion about a point (e.g. the approximate mean of the hidden variable distribution). Linear approximations are particularly useful since Gaussian distributions are closed under linear transformations.

- **Recognition Models**: Approximate the hidden variable posterior distribution using an explicit *bottom-up* recognition model/network.

- **Variational Methods**: Approximate the hidden variable posterior $p(H)$ with a tractable form $q(H)$. This gives a lower bound on the likelihood that can be maximised with respect to the parameters of $q(H)$.

Recognition Models

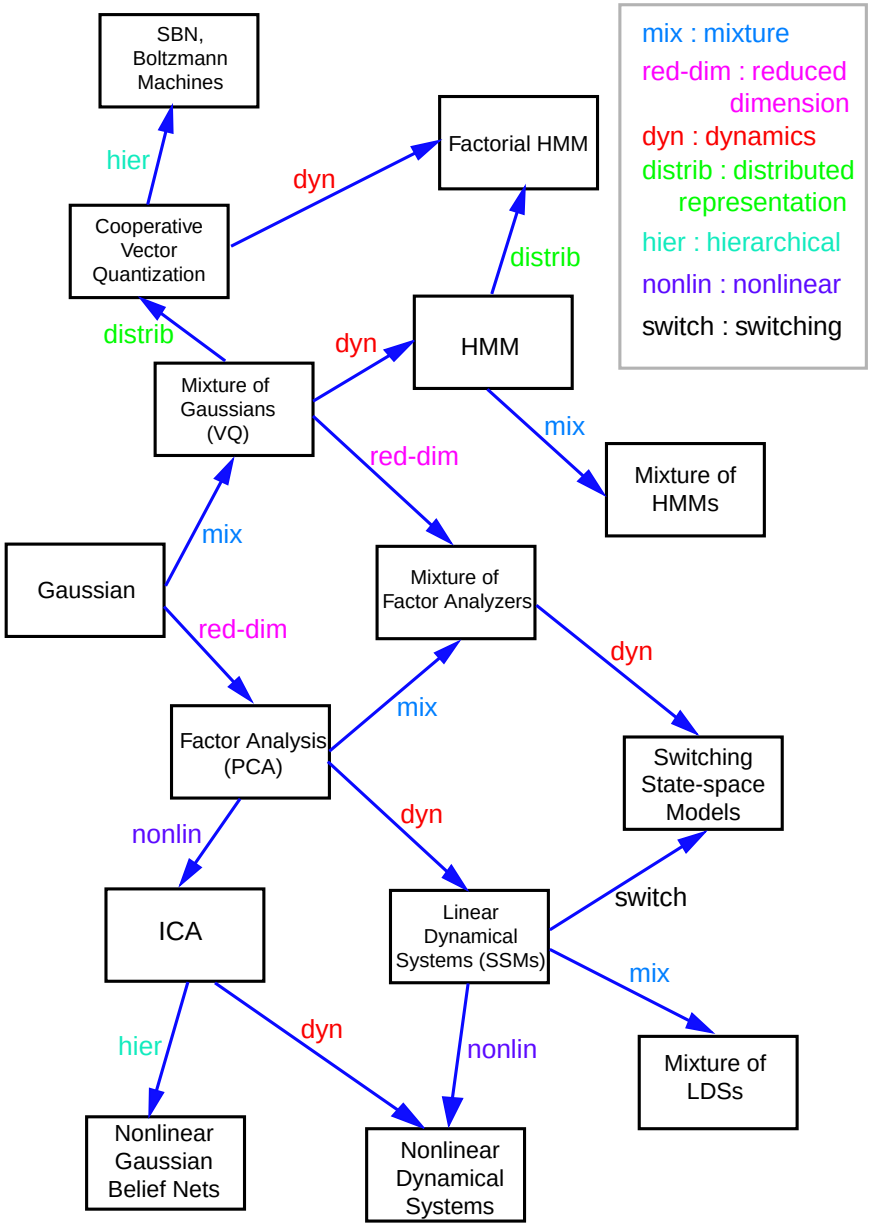


- a model is trained in a supervised way to recover the hidden causes (latent variables) from the observations

- this may take the form of explicit recognition network (e.g. Helmholtz machine) which mirrors the generative network (tractability at the cost of restricted approximating distribution)

- inference is done in a single *bottom-up* pass (no iteration required)

A Generative Model for Generative Models



Some Readings and References

Note: this list has not been updated in a few years

1. Attias, H. (1999) Independent Factor Analysis. Neural Computation 11:803-851.
2. Bell, A. and Sejnowski, T. (1995) An information maximization approach to blind source separation and blind deconvolution. Neural Computation 7:1129–1159.
3. Comon, P. (1994) Independent components analysis, a new concept? Signal Processing. 36:287–314.
4. Ghahramani, Z. and Beal, M.J. (2000) Graphical models and variational methods. In Saad & Opper (eds) Advanced Mean Field Method—Theory and Practice. MIT Press. Also available from my web page.
5. Ghahramani, Z. and Jordan, M.I. (1997) Factorial Hidden Markov Models. Machine Learning 29:245-273. `http://www.gatsby.ucl.ac.uk/~zoubin/papers/fhmmML.ps.gz`
6. David MacKay's Notes on ICA.
7. Lewicki, M. S. and Sejnowski, T. J. (1998) Learning overcomplete representations. Neural Computation.
8. Olshausen and Field (1996) Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature 381:607-609.
9. Barak Pearlmutter and Lucas Parra paper.
10. Roweis, S.T. and Ghahramani, Z. (1999) A Unifying Review of Linear Gaussian Models. Neural Computation 11(2). `http://www.gatsby.ucl.ac.uk/~zoubin/papers/lds.ps.gz or lds.pdf`
11. Max Welling's Notes on ICA: `http://www.gatsby.ucl.ac.uk/~zoubin/course01/WellingICA.ps`

Appendix: Matlab Code for ICA

```
% ICA using tanh nonlinearity and batch covariant algorithm
% (c) Zoubin Ghahramani
%
% function [W, Mu, LL]=ica(X,cyc,eta,Winit);
%
% X - data matrix (each row is a data point),   cyc - cycles of learning (default = 200)
% eta - learning rate (default = 0.2),          Winit - initial weight
%
% W - unmixing matrix,  Mu - data mean,         LL - log likelihoods during learning

function [W, Mu, LL]=ica(X,cyc,eta,Winit);

if nargin<2,  cyc=200; end;
if nargin<3,  eta=0.2; end;
[N D]=size(X);                  % size of data
Mu=mean(X); X=X-ones(N,1)*Mu;   % subtract mean
if nargin>3,    W=Winit;        % initialize matrix
else,   W=rand(D,D);  end;
LL=zeros(cyc,1);                % initialize log likelihoods

for i=1:cyc,
  U=X*W';
  logP=N*log(abs(det(W)))-sum(sum(log(cosh(U))))-N*D*log(pi);
  W=W+eta*(W-tanh(U')*U*W/N);                   % covariant algorithm
  % W=W+eta*(inv(W)-X'*tanh(U)/N)';             % standard algorithm
  LL(i)=logP; fprintf('cycle %g log P= %g\n',i,logP);
end;
```