Unsupervised Learning

Week 2: Latent Variable Models

Zoubin Ghahramani

`zoubin@gatsby.ucl.ac.uk`

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

Autumn 2003

The Gaussian Model (review)

$$p(\mathbf{y}|\mu,\Sigma) = |2\pi\Sigma|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\mathbf{y}-\mu)^\top \Sigma^{-1}(\mathbf{y}-\mu)\right\}$$

Data set $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$;

Likelihood is p(data|model): $p(Y|\mu,\Sigma) = \prod_{n=1}^{N} p(\mathbf{y}_n|\mu,\Sigma)$

Goal: find μ and Σ that maximise log likelihood:

$$\mathcal{L} \quad = \quad \log \prod_{n=1}^{N} p(\mathbf{y}_n|\mu,\Sigma) = -\frac{N}{2}\log|2\pi\Sigma| - \frac{1}{2}\sum_n (\mathbf{y}_n-\mu)^\top \Sigma^{-1}(\mathbf{y}_n-\mu)$$

Note: equivalently, minimise $-\mathcal{L}$, which is *quadratic* in μ
Procedure: take derivatives and set to zero:

$$\frac{\partial \mathcal{L}}{\partial \mu} = 0 \qquad \Rightarrow \qquad \hat{\mu} = \frac{1}{N}\sum_n \mathbf{y}_n \qquad \text{(sample mean)}$$

$$\frac{\partial \mathcal{L}}{\partial \Sigma} = 0 \qquad \Rightarrow \qquad \hat{\Sigma} = \frac{1}{N}\sum_n (\mathbf{y}_n-\hat{\mu})(\mathbf{y}_n-\hat{\mu})^\top \qquad \text{(sample covariance)}$$

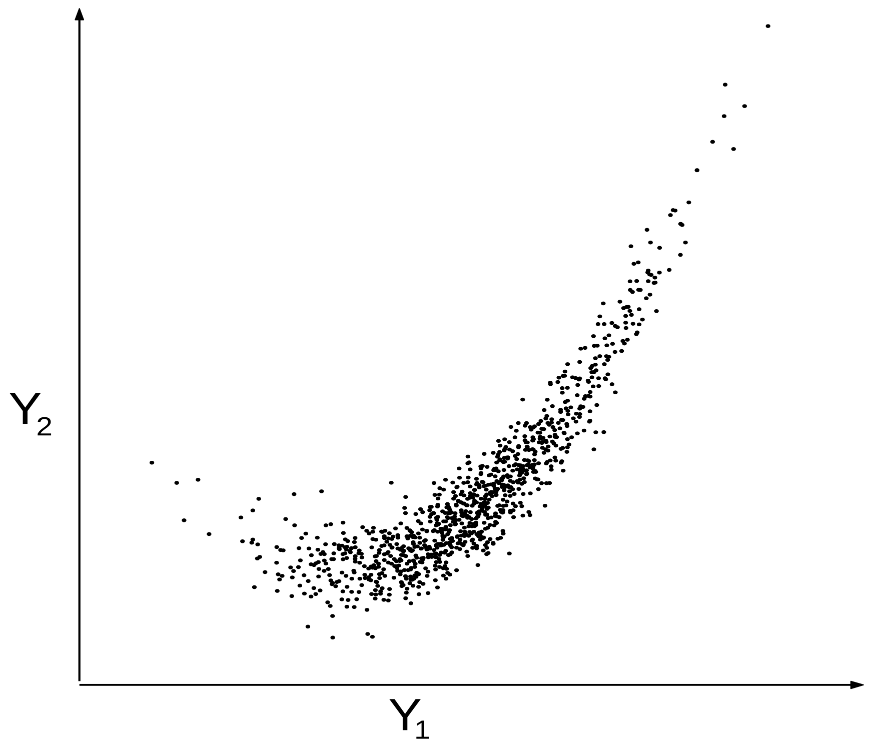
Three Limitations of Gaussians

- What about higher order statistical structure in the data?

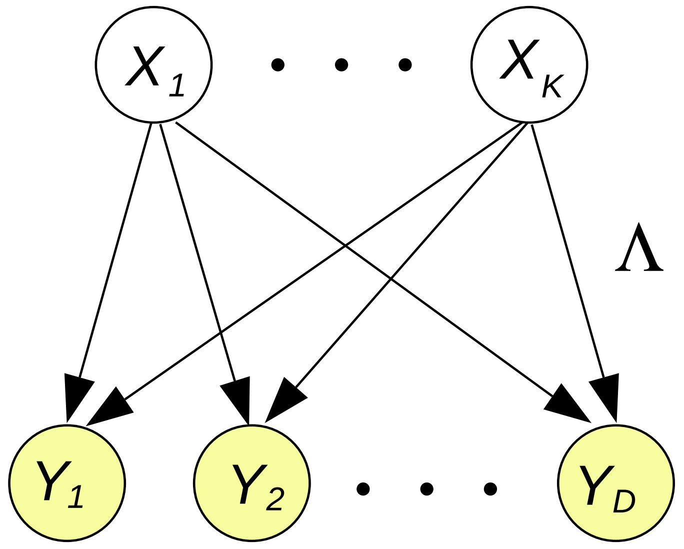
  $\Rightarrow$ nonlinear and hierarchical models
- What happens if there are outliers?

  $\Rightarrow$ other noise models
- There are $D(D+1)/2$ parameters in the multi-variate Gaussian model.
  What if D is very large?

  $\Rightarrow$ dimensionality reduction



Factor Analysis



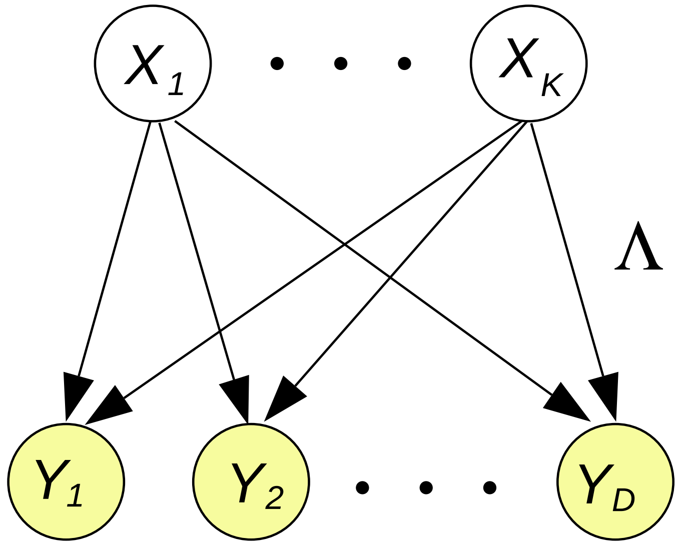
Linear generative model: $y_d = \sum_{k=1}^{K} \Lambda_{dk}\, x_k + \epsilon_d$

- x_k are independent $\mathcal{N}(0,1)$ Gaussian factors
- ϵ_d are independent $\mathcal{N}(0,\Psi_{dd})$ Gaussian noise
- $K < D$

So, $\mathbf{y}$ is Gaussian with: $p(\mathbf{y}) = \int p(\mathbf{x})p(\mathbf{y}|\mathbf{x})d\mathbf{x} = \mathcal{N}(0, \Lambda\Lambda^\top + \Psi)$
where Λ is a $D \times K$ matrix, and Ψ is diagonal.

Dimensionality Reduction: Finds a low-dimensional projection of high dimensional data that captures the correlation structure of the data.

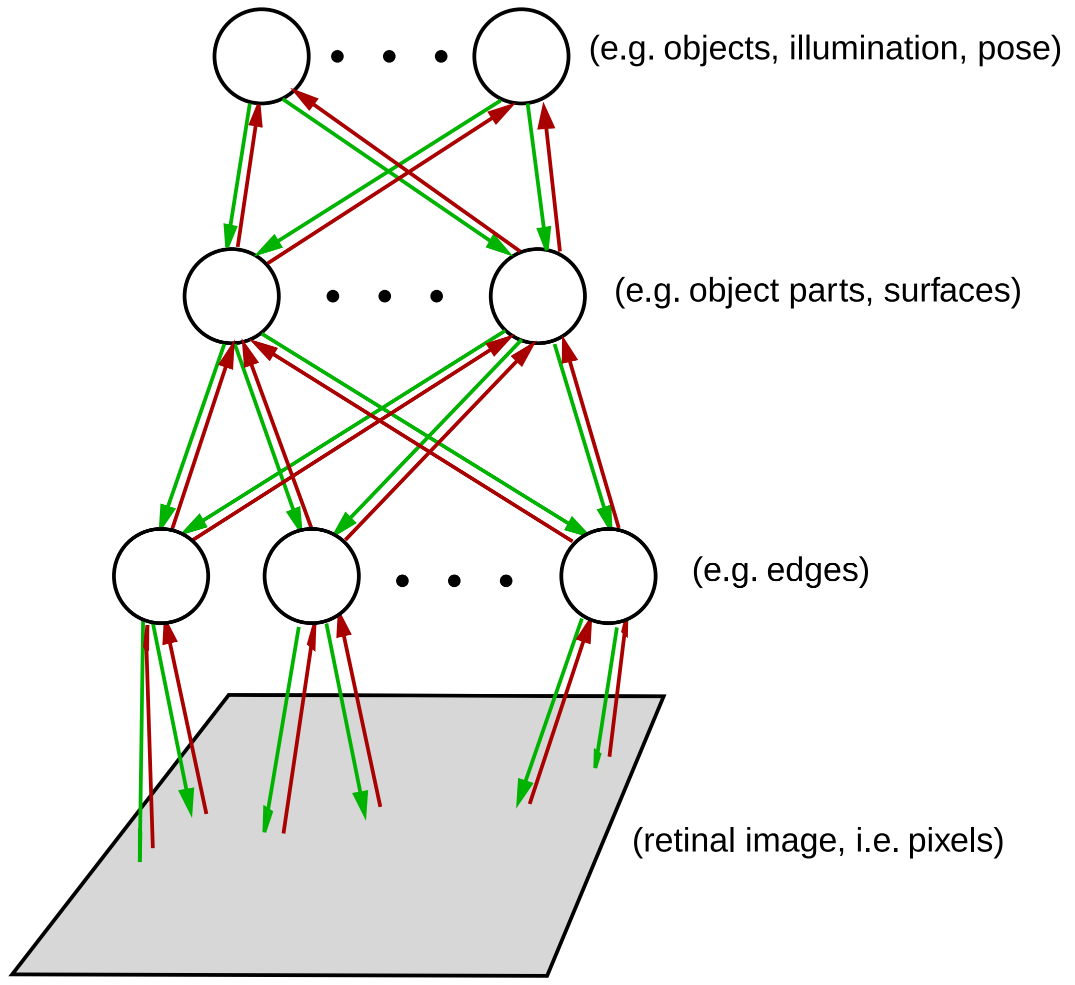
Factor Analysis (cont.)



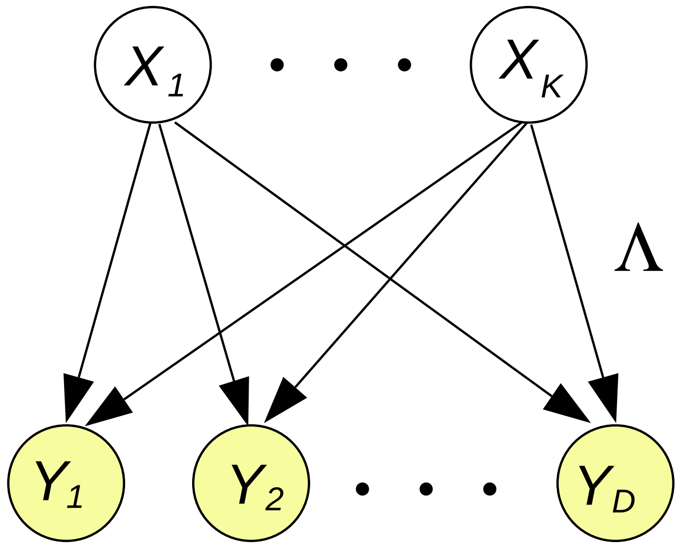
- ML learning finds Λ and Ψ given data
- parameters (corrected for symmetries): $$DK + D - \frac{K(K-1)}{2} < \frac{D(D+1)}{2}$$
- no closed form solution for ML params: $\mathcal{N}(0, \Lambda\Lambda^\top + \Psi)$
- [Bayesian treatment would also have priors over Λ and Ψ and would average over them for prediction.]

Latent Variable Models

Explain correlations in $\mathbf{y}$ by assuming some latent variables $\mathbf{x}$


(e.g. objects, illumination, pose)
(e.g. object parts, surfaces)
(e.g. edges)
(retinal image, i.e. pixels)

Principal Components Analysis



Noise variable becomes infinitesimal compared to the scale of the data: $\Psi = \lim_{\sigma^2 \to 0} \sigma^2 I$

Equivalently: reconstruction cost becomes infinite compared to the cost of coding the hidden units under the prior.

$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\beta\mathbf{y}, I - \beta\Lambda)$$

$$\beta = \lim_{\sigma^2 \to 0} \Lambda^T(\Lambda\Lambda^T + \sigma^2 I)^{-1} = (\Lambda^T\Lambda)^{-1}\Lambda^T$$

Eigenvalues and Eigenvectors

λ is an eigenvalue and $\mathbf{x}$ is an eigenvector of A if:

$$A\mathbf{x} = \lambda\mathbf{x}$$

and $\mathbf{x}$ is a unit vector ($\mathbf{x}^\top\mathbf{x} = 1$).

Interpretation: the operation of A in direction $\mathbf{x}$ is a scaling by λ.

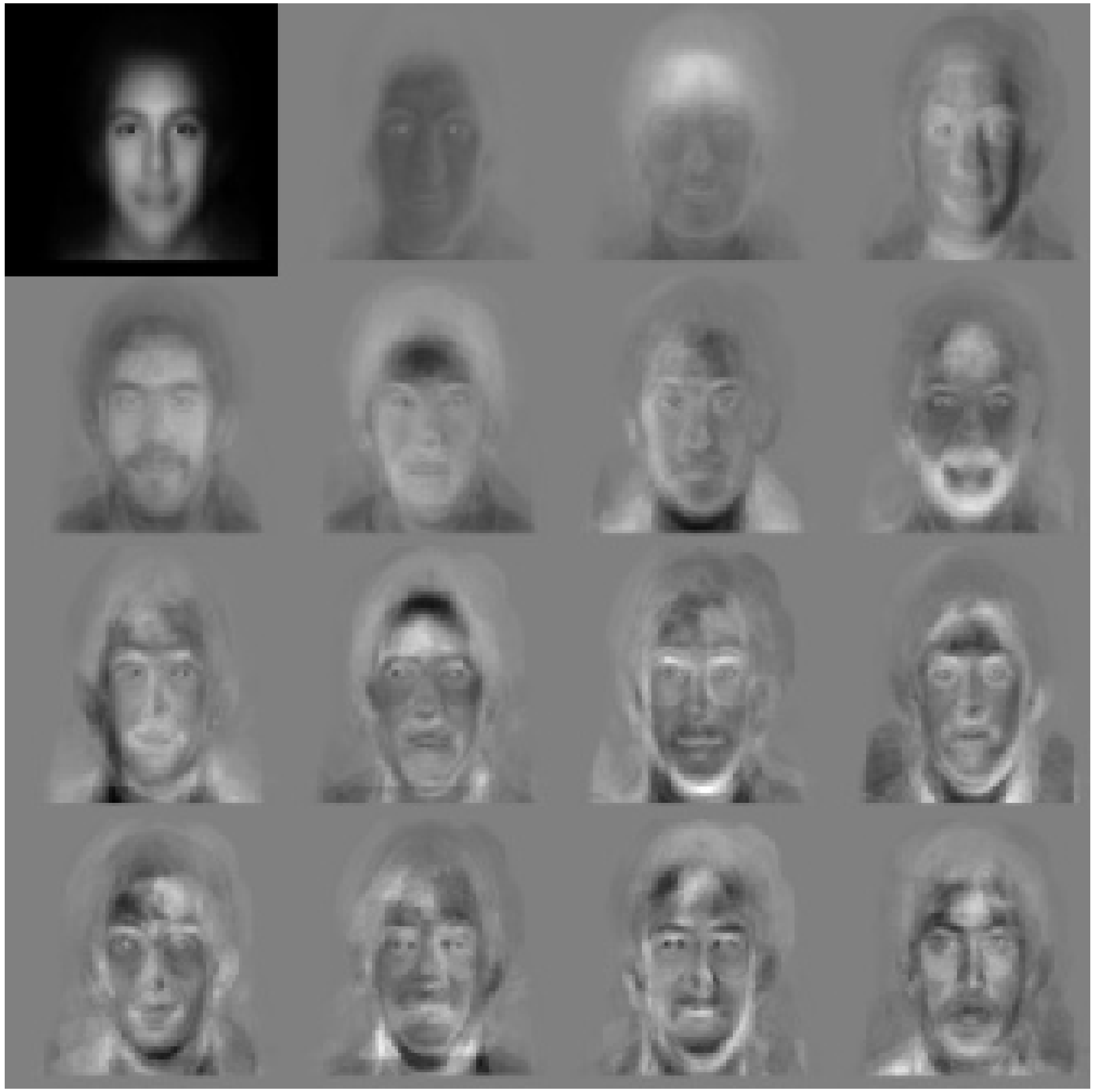
The K Principal Components are the K eigenvectors with the largest eigenvalues of the data covariance matrix (i.e. K directions with the largest variance).

Note: Σ can be decomposed:

$$\Sigma = USU^\top$$

where S is diag($\sigma_1^2, \ldots, \sigma_D^2$) and U is a an orthonormal matrix.

Example of PCA: Eigenfaces



from www-white.media.mit.edu/vismod/demos/facerec/basic.html

Mutual Information and PCA

Problem: Given $\mathbf{y}$, find $\mathbf{x} = A\mathbf{y}$ with columns of A unit vectors, s.t. $I(\mathbf{x};\mathbf{y})$ is maximised (assuming that $P(\mathbf{y})$ is Gaussian).

$$I(\mathbf{x};\mathbf{y}) = H(\mathbf{x}) + H(\mathbf{y}) - H(\mathbf{x},\mathbf{y}) = H(\mathbf{x})$$

So we want to maximize the entropy of $\mathbf{x}$. What is the entropy of a Gaussian?

$$H(\mathbf{z}) \quad = \quad -\int d\mathbf{z}\, p(\mathbf{z}) \ln p(\mathbf{z}) = \frac{1}{2}\ln|\Sigma| + \frac{D}{2}(1 + \ln 2\pi)$$

Therefore we want the distribution of $\mathbf{x}$ to have largest volume (i.e. det of covariance matrix).

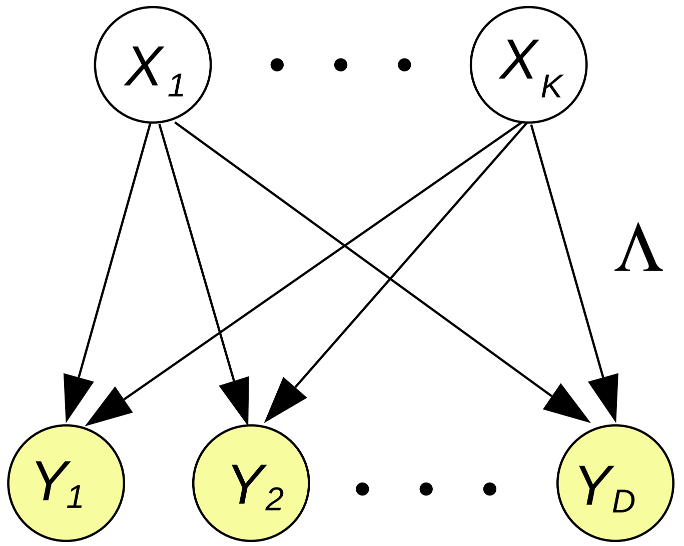
$$\Sigma_x = A\Sigma_y A^\top = AUS_yU^\top A^\top$$

So, A should be aligned with the columns of U which are associated with the largest eigenvalues (variances).

FA vs PCA

- PCA is rotationally invariant; FA is not
- FA is measurement scale invariant; PCA is not
- FA defines a probabilistic model; PCA does not

Probabilistic PCA



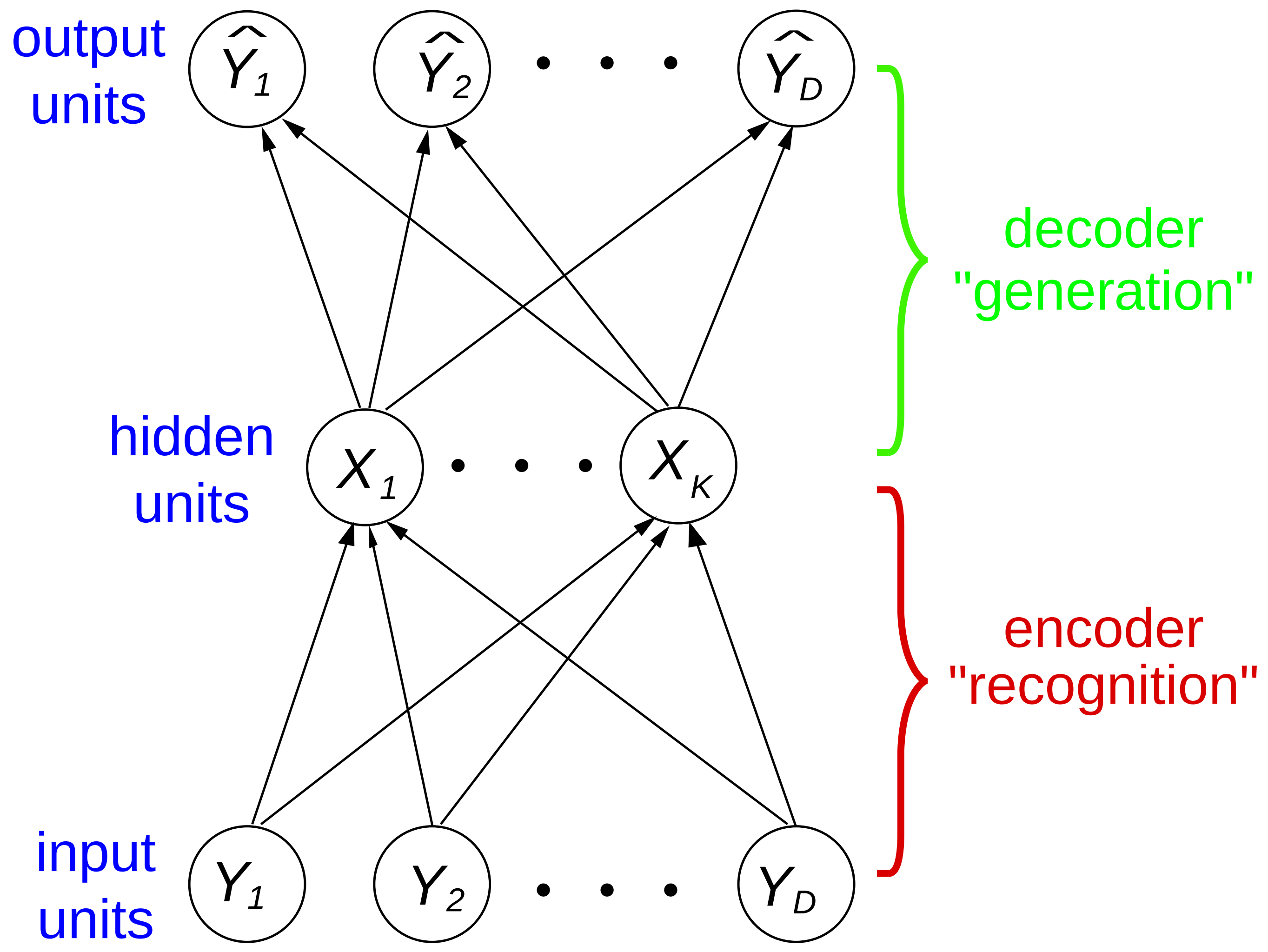
Linear generative model: $y_d = \sum_{k=1}^{K} \Lambda_{dk}\, x_k + \epsilon_d$

- x_k are independent $\mathcal{N}(0,1)$ Gaussian factors
- ϵ_d are independent $\mathcal{N}(0,\sigma^2)$ Gaussian noise
- $K < D$

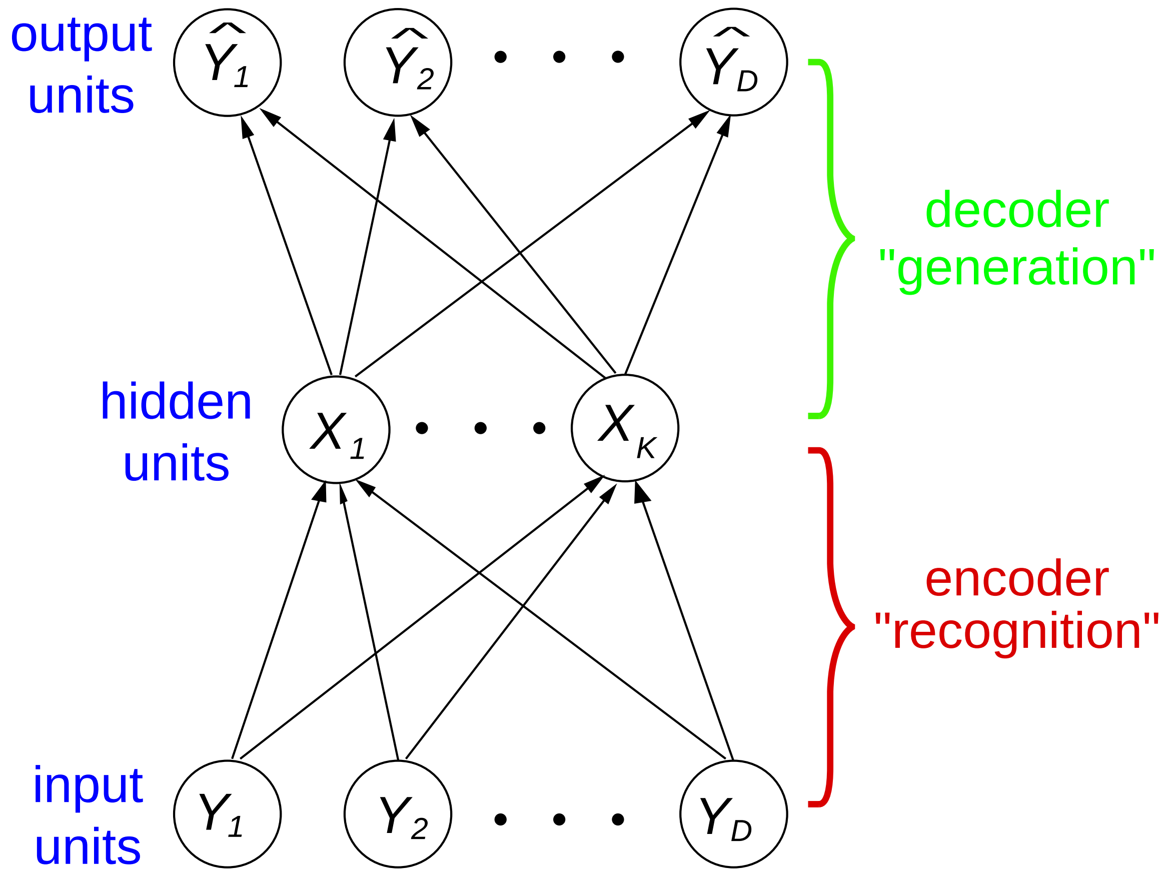
PPCA is factor analysis with isotropic noise: $\Psi = \sigma^2 I$

Finds the same principal subspace as PCA but provides a well-defined probabilistic model.

Network Interpretations and Encoder-Decoder Duality



From Supervised Learning to PCA



A linear autoencoder neural network trained to minimise squared error learns to perform PCA (Baldi & Hornik, 1989).

Gradient Methods of Learning FA

Write down negative log likelihood:

$$\frac{1}{2}\log|2\pi(\Lambda\Lambda^\top + \Psi)| + \frac{1}{2}\mathbf{y}^\top(\Lambda\Lambda^\top + \Psi)^{-1}\mathbf{y}$$

Optimize w.r.t. Λ and Ψ (need matrix calculus) subject to constraints

We will soon see an easier way to learn latent variable models...

Limitations of Gaussian, FA and PCA models

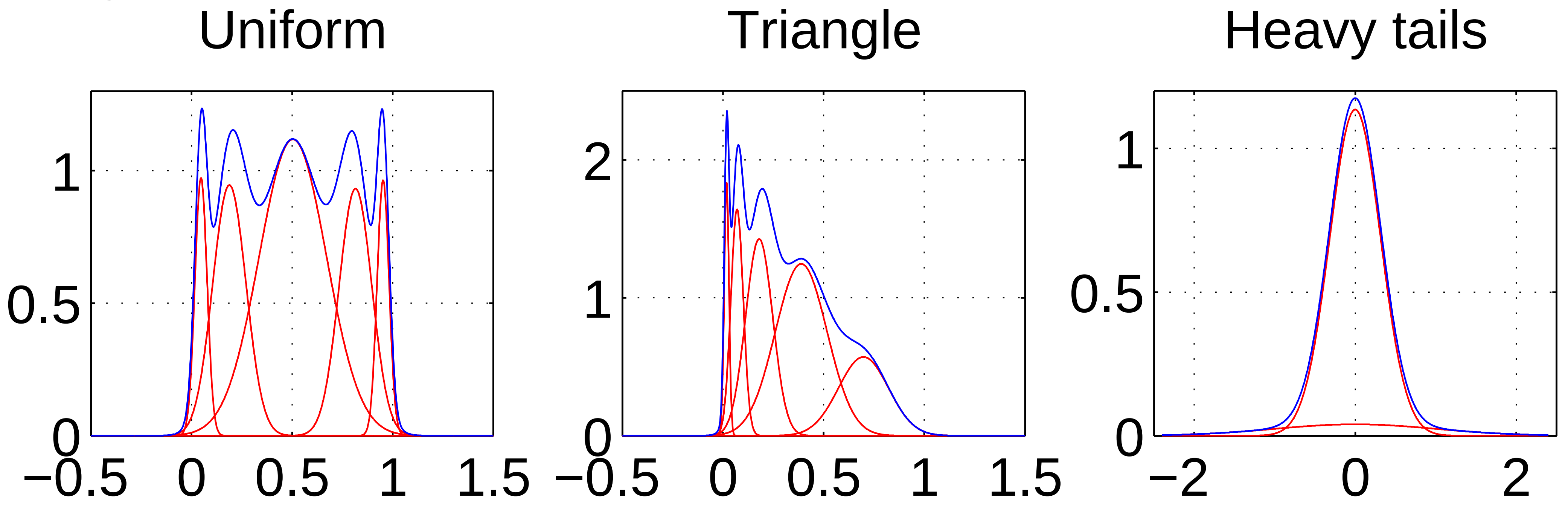
- Gaussian, FA and PCA models are easy to understand and use in practise.
- They are a convenient way to find interesting directions in very high dimensional data sets, eg as preprocessing
- Their problem is that they make very strong assumptions about the distribution of the data, only the mean and variance of the data are taken into account.

The class of densities which can be modelled is too restrictive.
By using *mixtures* of simple distributions, such as Gaussians, we can expand the class of densities greatly.

Mixtures of Gaussians – MoG

Use probabilistic mixtures of simple (eg Gaussian) density models.

Some examples where non-Gaussian densities are modelled (aproximated) as a mixture of Gaussians. The red curves show the (weighted) Gaussians, and the blue curve the resulting density.



The advantage of this mixture approach, is that given enough mixture components we can model (almost) any density (as accurately as desired), but we still only need to work with the well known Gaussian form.

The MoG likelihood

Here a set of k Gaussians, each with a separate mean, μ_i, and covariance Σ_i are weighted together with non-negative and normalised weights π_i:

$$\pi_i \geq 0, \quad \text{and} \quad \sum_{i=1}^{k} \pi_i = 1.$$

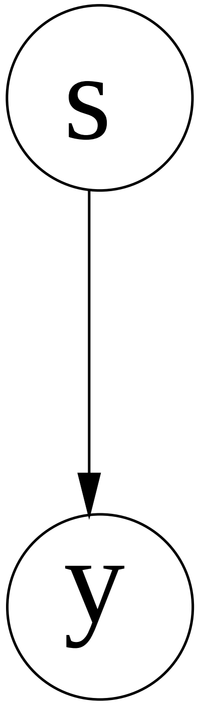
The probability of an observation $y^{(c)}$ under mixture component i is Gaussian:

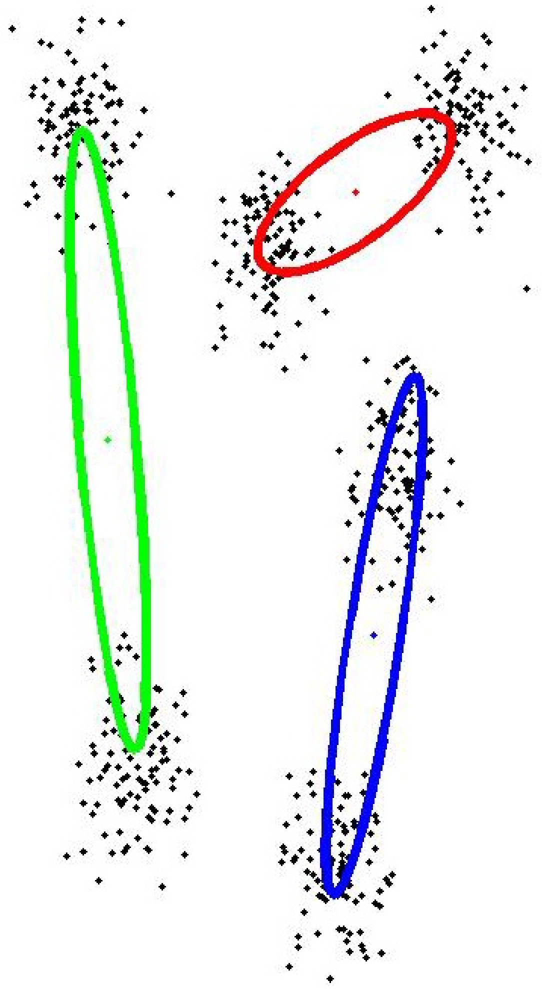
$$p(y^{(c)}|\mu_i, \Sigma_i) = |2\pi\Sigma_i|^{-1/2} \exp\big(-\frac{1}{2}(y^{(c)} - \mu_i)^\top \Sigma_i^{-1}(y^{(c)} - \mu_i)\big).$$

The probability of an observation $y^{(c)}$ under the entire mixture model is a weighted sum of Gaussian densities:

$$\begin{aligned} p(y^{(c)}|\mu, \Sigma, \pi) &= \sum_{i=1}^{k} \pi_i p(y^{(c)}|\mu_i, \Sigma_i) \\ &= \sum_{i=1}^{k} \pi_i |2\pi\Sigma_i|^{-1/2} \exp\big(-\frac{1}{2}(y^{(c)} - \mu_i)^\top \Sigma_i^{-1}(y^{(c)} - \mu_i)\big), \end{aligned}$$

MoG as a Latent Variable Model for Clustering





$$P(S^{(c)} = i|\pi) = \pi_i$$

$$p(y^{(c)}|S^{(c)} = i, \mu, \Sigma) = \mathcal{N}(\mu_i, \Sigma_i)$$

The MoG likelihood, continued

The probability of a set of n observations, $y = \{y^{(1)}, \dots, y^{(n)}\}$ (the likelihood):

$$p(y|\mu, \Sigma, \pi) = \prod_{c=1}^{n} \sum_{i=1}^{k} \pi_i \, p(y^{(c)}|\mu_i, \Sigma_i)$$
$$= \prod_{c=1}^{n} \sum_{i=1}^{k} \pi_i |2\pi\Sigma_i|^{-1/2} \exp\big(-\frac{1}{2}(y^{(c)} - \mu_i)^\top \Sigma_i^{-1} (y^{(c)} - \mu_i)\big).$$

Here, the observations are thought of as being generated *independently* from the mixture (given the parameters). The log of the likelihood is:

$$\log p(y|\mu, \Sigma, \pi) = \log \prod_{c=1}^{n} \sum_{i=1}^{k} \pi_i p(y^{(c)}|\mu_i, \Sigma_i) = \sum_{c=1}^{n} \log \sum_{i=1}^{k} \pi_i p(y^{(c)}|\mu_i, \Sigma_i)$$
$$= \sum_{c=1}^{n} \log \sum_{i=1}^{k} \pi_i (2\pi\Sigma_i)^{-1/2} \exp\big(-\frac{1}{2}(y^{(c)} - \mu_i)^\top \Sigma_i^{-1} (y^{(c)} - \mu_i)\big).$$

Maximum likelihood (ML) training a MoG model

The log likelihood is: $\mathcal{L} = \sum_{c=1}^{n} \log \sum_{i=1}^{k} \pi_i p(y^{(c)}|\mu_i, \Sigma_i)$

Its partial derivative wrt $\theta_i = \{\mu_i, \Sigma_i\}$ is

$$\frac{\partial \log p(y|\pi, \mu, \Sigma)}{\partial \theta_i} = \sum_{c=1}^{n} \frac{\pi_i}{\sum_{j=1}^{k} \pi_j p(y^{(c)}|\mu_j, \Sigma_j)} \frac{\partial p(y^{(c)}|\mu_i, \Sigma_i)}{\partial \theta_i}$$

Using the identity $\partial p/\partial\theta = p \times \partial \log p/\partial\theta$, it can be re-written as:

$$\frac{\partial \log p(y|\pi, \mu, \Sigma)}{\partial \theta_i} = \sum_{c=1}^{n} r_i^{(c)} \frac{\partial \log p(y^{(c)}|\mu_i, \Sigma_i)}{\partial \theta_i},$$

where we have defined the responsibilities of component i for data point c as:

$$r_i^{(c)} = \frac{\pi_i p(y^{(c)}|\mu_i, \Sigma_i)}{\sum_{j=1}^{k} \pi_j p(y^{(c)}|\mu_j, \Sigma_j)} = P(S^{(c)} = i|y^{(c)}, \mu, \Sigma) \tag{1}$$

Derivatives of log likelihood

For the means we get:

$$\frac{\partial \log p(y|\pi,\mu,\Sigma)}{\partial \mu_i} = \sum_{c=1}^{n} r_i^{(c)} \Sigma_i^{-1} (y^{(c)} - \mu_i) \tag{2}$$

and for the *precisions* (inverse variances, Σ_i^{-1}):

$$\frac{\partial \log p(y|\pi,\mu,\Sigma)}{\Sigma_i^{-1}} = \frac{1}{2} \sum_{c=1}^{n} r_i^{(c)} \big(\Sigma_i - (y^{(c)} - \mu_i)(y^{(c)} - \mu_i)^\top \big).$$

Finally, the partial derivative wrt the mixing proportions is:

$$\frac{\partial \log p(y|\pi,\mu,\Sigma)}{\partial \pi_i} = \sum_{c=1}^{n} \frac{p(y^{(c)}|\mu_i,\Sigma_i)}{\sum_{j=1}^{k} \pi_j p(y^{(c)}|\mu_j,\Sigma_j)}$$

These equations together can be used for gradient based learning; eg taking small steps in the direction of the gradient (or using conjugate gradients).

The k-means algorithm

Assume for simplicity, that $\pi_i = 1/k$; assume further, that $\Sigma_i = \sigma^2 I$, where $\sigma^2 \to 0$. Then the responsibilities become discrete:

$$r_i^{(c)} = \delta\big(i, \mathrm{argmax}_j\, p(y^{(c)}|\mu_j, \Sigma_j)\big),$$

being 1 for the most likely component and 0 otherwise. We can then solve directly for the means μ_i by setting eq. (2) to zero.
The **k-means algorithm** iterates:

- For each data point $y^{(c)}$, set $r_i^{(c)} = 1$ if μ_i is closest to $y^{(c)}$, o.w. $r_i^{(c)} = 0$
- For each mean μ_i, set $\mu_i = \dfrac{\sum_c r_i^{(c)} y^{(c)}}{\sum_c r_i^{(c)}}$

This usually converges in a few iterations and it has the advantage that there is no learning rate.
However, the assumptions we made are quite **limiting**

The EM algorithm: a heuristic “derivation”

One could to find a procedure which jumps in parameter space, without quite as severe restrictions as k-means.

One idea is to neglect the dependency of the responsibilities on the parameters when calculating derivatives. If we do that, we can explicitly solve for the parameters, given the responsibilities.

We get the following two step algorithm:

- Evaluate the responsibilities (given the parameters), using eq. (1).

- Optimize the parameters (given the responsibilities)

$$\mu_i = \sum_{c=1}^{n} r_i^{(c)} y^{(c)} \Big/ \sum_{c=1}^{n} r_i^{(c)}, \quad \text{and} \quad \Sigma_i = \sum_{c=1}^{n} r_i^{(c)} (y^{(c)} - \mu_i)(y^{(c)} - \mu_i)^\top \Big/ \sum_{c=1}^{n} r_i^{(c)}.$$

In fact, we will show in the next lectures that this procedure is guaranteed not to decrease the likelihood in each iteration. The algorithm is called the EM algorithm.

Problems

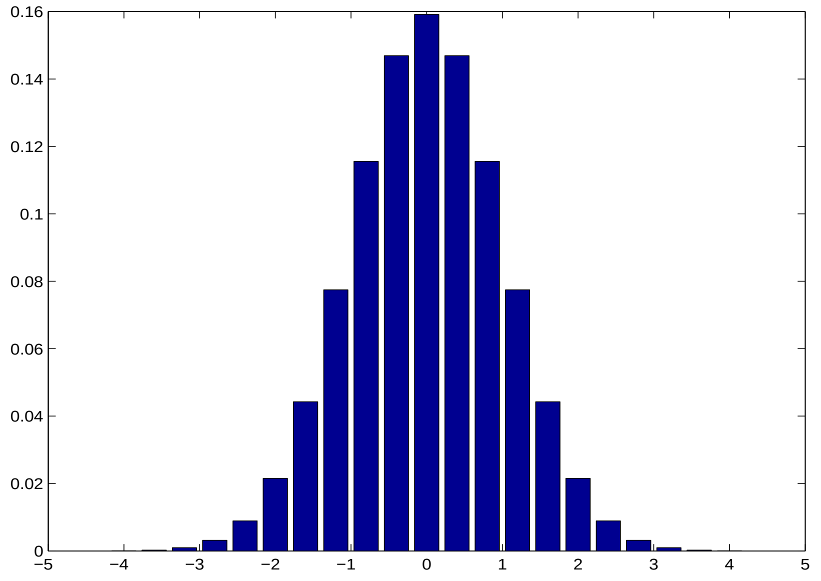
There are several problems with the new algorithms:

- slow convergence for the gradient based method
- gradient based method may develop invalid covariance matrices
- local minima; the end configuration may depend on the starting state
- how do you adjust k? Using the likelihood alone is no good.
- singularities; components with a single data point will have their covariance going to zero and the likelihood will tend to infinity.

Appendix: Coding Interpretation of Factor Analysis: Coding Under a Gaussian

Remember, from Shannon's source coding theorem:

$$\begin{aligned} l(x) &= -\log P(x) \approx -\log[p(x)\Delta] = -\log p(x) - \log \Delta \\ &= \frac{(x-\mu)^2}{2\sigma^2} + \frac{1}{2}\log 2\pi + \log \sigma - \log \Delta \end{aligned}$$



Note as $\Delta \Rightarrow 0$ then $l(x) \Rightarrow \infty$.

Coding Interpretation of Factor Analysis

Multivariate: $l(y) = \frac{1}{2}\sum_d \frac{(y_d - \mu_d)^2}{\sigma_d^2} + \frac{D}{2}\log 2\pi + \sum_d \log \sigma_d - D \log \Delta$

Alternative, two-stage code...

First code the K factors: $l(x) = \frac{1}{2}\sum_k x_k^2 + \frac{K}{2}\log 2\pi - K \log \Delta$

Then code the data given the factors:

$$l(y|x) = \frac{1}{2}\sum_d \frac{(y_d - \sum_k \Lambda_{dk} x_k)^2}{\Psi_d^2} + \frac{D}{2}\log 2\pi + \sum_d \log \Psi_d - D \log \Delta$$

How should we choose the x ?

Deterministic vs stochastic codes and “bits back”

Coding Interpretation of PCA

First code the K factors:

$$l(x) = \frac{1}{2}\sum_k x_k^2 + \frac{K}{2}\log 2\pi - K\log\Delta$$

Then code the data given the factors:

$$l(y|x) = \frac{1}{2}\sum_d \frac{(y_d - \sum_k \Lambda_{dk} x_k)^2}{\sigma^2} + \frac{D}{2}\log 2\pi + D\log\sigma - D\log\Delta$$

Since $\sigma \to 0$ the cost of coding the factors is negligible compared to the cost of coding the data.

Appendix: Coding under a mixture: Naive

Consider the following communication game: a **sender** wishes to communicate a large i.i.d. data set $(x_1, x_2, \ldots, x_n)$ losslessly (to within some discretization Δ) to a **receiver**. The sender and receiver agree to use a mixture of Gaussians (MoG) model. The sender also wishes to communicate some **additional data** which has already been compressed so that it looks like a bucket of m random bits.

Let's consider three qualitatively different ways in which the sender can encode the data:

- **Naive.** Discretize the range of data $(\min(x), \max(x))$ into equal bins of size Δ. Send $\min(x)$ and $\max(x)$ and then for each data point the identity of the bin it fell into. The total code length will be $n \log_2((\max(x) - \min(x))/\Delta) + m$. This is optimal only if the data is uniformly distributed.

Coding under a mixture: Deterministic

- **Deterministic Three-Part Code**

  1. Send θ, the parameters of the MoG. This has to be done only once (which code should be used here? maybe sender and receiver need to have agreed on a coding scheme, i.e. a $p(\theta)$...).

For each data point x_i

  2. Send the discrete identity s_i of the best (i.e. most probable) Gaussian given the data point:

$$s_i = \text{argmax}_k \, P(S_i = k | x_i, \theta)$$

  If all the Gaussians have equal prior and equal covariance this is the identity of the Gaussian whose mean is closest to x_i. (You can think of the s_i as the discrete latent variable associated with x_i).

  3. Send the code for x_i given the identity of the "best" Gaussian. This corresponds to the "reconstruction coding cost" to within Δ:

$$-\log p(x_i|\mu_{s_i}, \Sigma_{s_i}) - \log \Delta$$

  Since your code is deterministic, you also need to spend m bits to send the additional data.

The total coding cost for this deterministic scheme is:

$$-\log p(\theta) - \sum_i \text{argmin}_k \left[\log p(x_i|S_i = k) + \log P(S_i = k)\right] - n \log \Delta + m \qquad (3)$$

Coding under a mixture: Stochastic

- **Stochastic Three-part Code**

1. Send θ, as above.

For each data point, x_i

2. Send a discrete identity s_i "sampled" from some arbitrary distribution: $s_i \sim Q_i(S)$
3. Send the code for x_i given the identity of the sampled Gaussian:

$$-\log p(x_i|\mu_{s_i}, \Sigma_{s_i}) - \log \Delta$$

Every time you supposedly "sampled", you actually looked at the bucket of random bits, and selected s_i deterministically using the "random" bits and the distribution Q_i. You will need to use up $-\log Q_i(s_i)$ bits from the bucket to code s_i. (In practice this can be done using block arithmetic codes). Fortunately, the receiver can reverse-engineer exactly what you've done, and figure out what these random bits were, so you only need to send the remaining bits. The bits you saved are called the **bits-back** (Hinton and Zemel, 1994).

The total expected coding cost for this stochastic scheme is:

$$\begin{aligned} -\log p(\theta) - \sum_i \sum_k Q_i(S_i = k) \left[\log p(x_i|S_i = k) + \log P(S_i = k)\right] - n \log \Delta \\ + \left[m - \left(-\sum_i \sum_k Q_i(S_i = k) \log Q_i(S_i = k)\right)\right] \end{aligned} \qquad (4)$$

Coding under a mixture (cont)

What is the optimal stochastic code?
Minimizing (4) with respect to each Q_i we find that the stochastic code with the minimum expected coding cost is:

$$Q_i^*(S) = P(S|x_i, \theta)$$

This can be proven by re-writing (4) using KL-divergences.

So the optimal thing to do is to pick latent codes randomly from the **posterior distribution** of the latent variables for each data point. If your mixture model is the correct distribution for the data, this stochastic code will achieve the Shannon lower limit, while the deterministic code will not.

Note 1 One can use exactly the same argument for a stochastic three-part code for factor analysis, the main difference being that the latent variables are continuous (and thus need also a discretization Δ).

Note 2 The same argument can be generalized to the question of how one should code the parameter θ. If sender and receiver agree on a prior, then using a similar bits-back argument, the lowest expected coding cost is achieved sending a θ sampled from the posterior, **not** by sending the most probable θ. This point is usually confused in the MDL/MML literature. See Wallace and Dowe (1999) for a nice explanation.

Suggested Readings

- David MacKay's Textbook http://wol.ra.phy.cam.ac.uk/itprnn/book.ps.gz, chapter 21, pages 295-306, draft 2.2.4, August 31, 2001
- Hinton and Zemel (1994) Autoencoders, minimum description length, and the Helmholtz free energy. In *Adv in Neur Info Proc Syst 6*. Morgan Kaufmann. See www.cs.toronto.edu/∼hinton/
- Minka, T. Tutorial on linear algebra.
  http://www-white.media.mit.edu/∼tpminka/papers/matrix.html
- Roweis and Ghahramani (1999) A Unifying Review of Linear Gaussian Models. Neural Computation 11(2). Sections 1-5.3 and 6-6.1. See also Appendix A.1-A.2. http://www.gatsby.ucl.ac.uk/∼zoubin/papers/lds.ps.gz
- Wallace, C. S. and Dowe, D. L. (1999) Minimum message length and Kolmogorov complexity. *The Computer Journal* 42(4):270–283.
- Welling, M. (2000) Linear models. class notes.
  http://www.gatsby.ucl.ac.uk/∼zoubin/course01/PCA.ps