DIMENSIONALITY REDUCTION

Sam Roweis

University of Toronto
Department of Computer Science

[Google: "Sam Toronto"]

Ground Rules

- Please interrupt often and loudly!
- The goal of this talk is to teach you a few things, to get you thinking, and to answer some questions; not to intimidate you. (I have other talks to intimidate you if you want...)
- Warning: I will try to cover one graduate seminar of material in 80 minutes, so we are going to do it multiscale.

Coarse Resolution (Elevator)

- Unsupervised Learning / Density Estimation / Manifolds
- Linear (Projection) Methods
- Locally Linear (Alignment) Methods
- Nonlinear (Embedding) Methods

Medium Resolution (Coffee)

- Unsupervised Learning / Density Estimation / Manifolds

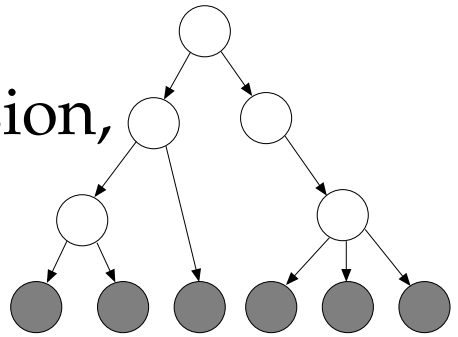
  Data $\mathbf{x}$ from the real world is "correlated mush". (e.g. pixels)
  What we really want are $\mathbf{y}$: the "things going on underneath".
  (e.g. pose, expression, lighting, camera position)

  Mathematically $\mathbf{x} = f(\mathbf{y})$+noise
  We measure $\boldsymbol{x}_n$, want to learn both $f(\cdot)$ and $\mathbf{y}_n$.
  Sounds hard! Yes! Must make assumptions about $f(\cdot)$.
  Typical assumputions: domain is low dimensional; smooth.

- Linear (Projection) Methods
- Locally Linear (Alignment) Methods
- Nonlinear (Embedding) Methods

Goals of Unsupervised Learning

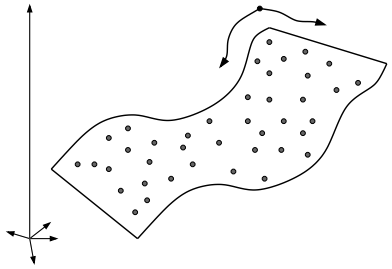
- **Density Modeling?**
  Gives a quantitative **familiarity** signal $P(x)$.
  Can be used for pattern classification, compression, outlier rejection, missing data imputation,...
  Density models can also generate new data.



- **Learning Good Codes for Data**
  Automatically discover **internal representations** of inputs to be used in further analysis: visualization, denoising, interpolation, extrapolation, compression, classification, measuring similarity of complex data. "Good" $\approx$ topographic, sparse, invariant,...
- **Building A Magic Box**
  Ideally, unsupervised learning produces a magic box. Turning a few knobs/levers generates all possible data on the manifold. Furthermore, given data, the box estimates the knob settings to generate it. Learn hidden causes which parameterize relevant modes of variability.

Dimensionality Reduction & Manifolds

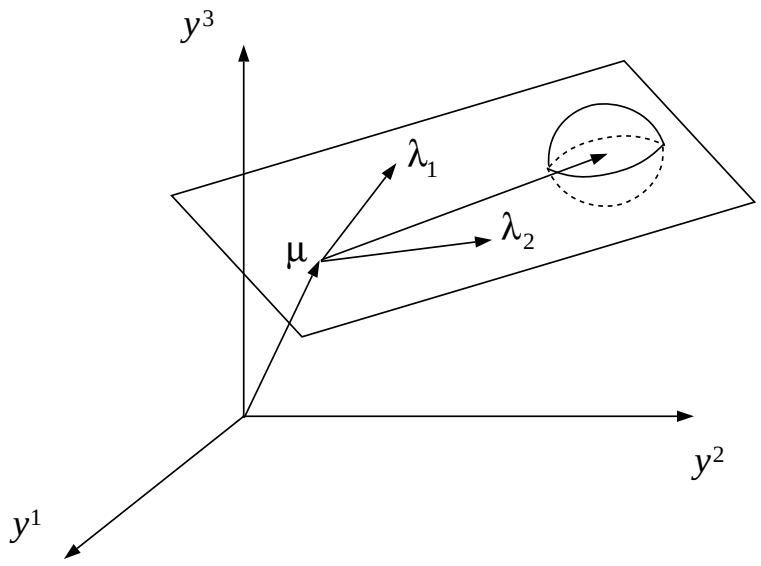
- **Dimensionality Reduction**
  Need to vastly decrease size of inputs while preserving important similarities and differences.
  Improve efficiency of statistical algorithms.
- **Most Inputs are Redundant**
  Data are points in a high dimensional space.
  Coherent structure in the world generates strong correlations between components.
  Geometrically, observations lie on or near thin, connected low dimensional manifolds.

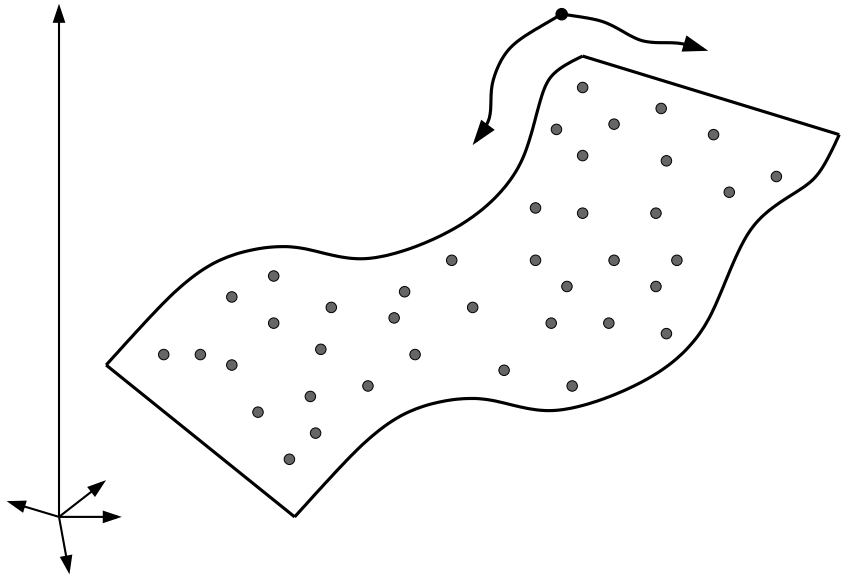


- **Many Manifolds are Nonlinear**
  We want to model the curved geometry of high-dimensional manifolds. Linearity can be a useful approximation in local domains, but globally too strong.
  Most interesting data has nonlinear structure.

Continuous Latent Variable Models

- Often there are some unknown *underlying causes* of the data.
- Mixture models use a discrete class variable: clustering.
- Often, it is more appropriate to think in terms of continuous *factors* which control the data we observe. Geometrically, this is equivalent to thinking of a data *subspace* or *manifold*.

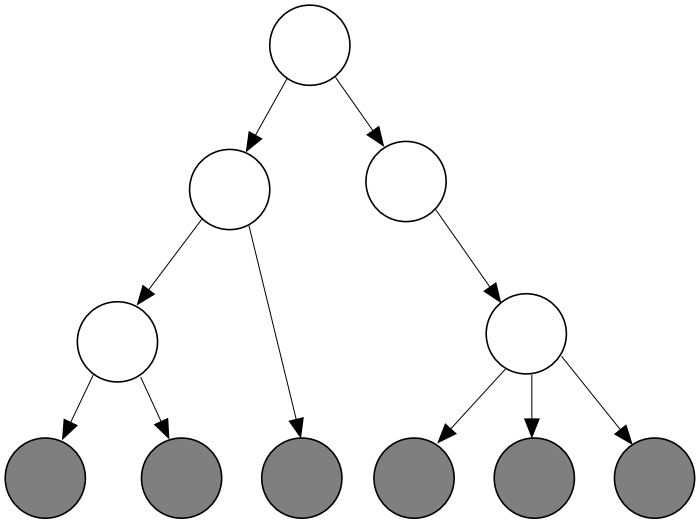


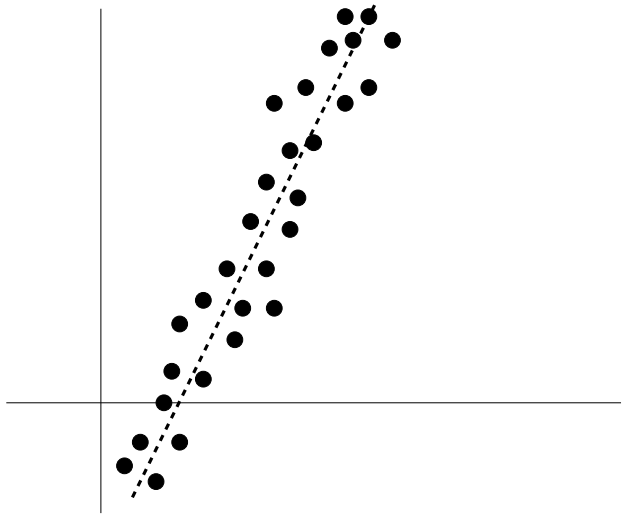


- To generate a datapoint from such a model, first generate a point within the manifold then add noise.
  Intrinsic coordinates of point $\equiv$ components of latent variable.

Dimensionality Reduction vs. Clustering

- Training such "factor models" is called *dimensionality reduction*. Think of this as (non)linear regression with missing inputs.
- Continuous causes can sometimes be much more efficient at representing information than discrete causes.
- For example, if there are two factors, with about 256 settings each we can describe the latent causes with two 8-bit numbers.
- If we tried to cluster we would need $2^{16} \approx 10^5$ clusters.





Let's Get Going...

- Unsupervised Learning / Density Estimation / Manifolds
- Linear (Projection) Methods

  Assume $\mathbf{x} = \mathbf{A}\mathbf{y}$+noise; i.e. $f()$ is linear.
  Now we can recover $\mathbf{A}$ by searching for the "postcard that best passes through our cloud of mush", after which we can recover individual $\mathbf{y}_n$ by solving $A\mathbf{y}_n = \boldsymbol{x}_n$.

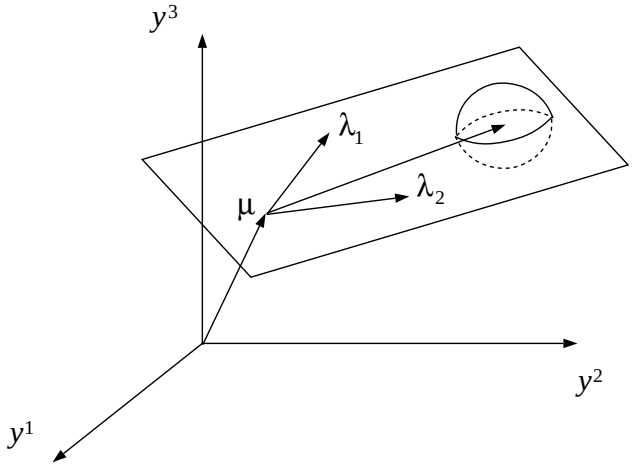
  Factor Analysis, PCA, ICA, MDS

  Snap-shot methods and SVD

- Locally Linear (Alignment) Methods
- Nonlinear (Embedding) Methods

Factor Analysis

- When we assume that the subspace is *linear* and that the underlying latent variable has a Gaussian distribution we get a model known as *factor analysis*:
  — data $\mathbf{y}$ (p-dim);
  — latent variable $\mathbf{z}$ (k-dim)



$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0, I)$$
$$p(\mathbf{y}|\mathbf{z}, \theta) = \mathcal{N}(\mathbf{y}|\mu + \Lambda\mathbf{z}, \Psi)$$

where μ is the mean vector, Λ is the p by k *factor loading matrix*, and Ψ is the *sensor noise covariance* (usually diagonal).

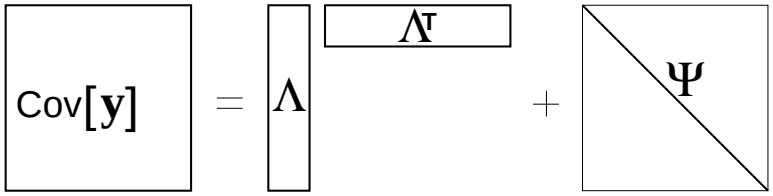
- Important: since the product of Gaussians is still Gaussian, the joint distribution $p(\mathbf{z}, \mathbf{y})$, the other marginal $p(\mathbf{y})$ and the conditional $p(\mathbf{z}|\mathbf{y})$ are also Gaussian.

Constrained Covariance Gaussian

- Marginal density for factor analysis ($\mathbf{y}$ is p-dim, $\mathbf{z}$ is k-dim):

$$p(\mathbf{y}|\theta) = \mathcal{N}(\mathbf{y}|\mu\ ,\ \Lambda\Lambda^\top + \Psi)$$

- So the effective covariance is the low-rank outer product of two long skinny matrices plus a diagonal matrix:



- In other words, factor analysis is just a constrained Gaussian model. (If Ψ were not diagonal then we could model any Gaussian and it would be pointless.)
- We can do maximum likelihood learning of Ψ, Λ using (surprise, surprise) the EM algorithm.
- It is easy to find μ: just take the mean of the data.

Probabilistic Principal Component Analysis

- In Factor Analysis, we can write the marginal density explicitly:

$$p(\mathbf{y}|\theta) = \int_{\mathbf{z}} p(\mathbf{z})p(\mathbf{y}|\mathbf{z},\theta)d\mathbf{z} = \mathcal{N}(\mathbf{y}|\mu\ ,\ \Lambda\Lambda^\top + \Psi)$$

- Noise Ψ mut be restricted for model to be interesting. (Why?)
- In Factor Analysis Ψ is restricted to be *diagonal* (axis-aligned).
- What if we further restrict $\Psi = \sigma^2 I$ (ie *spherical*)?
- We get Probabilistic Principal Component Analysis (PPCA):

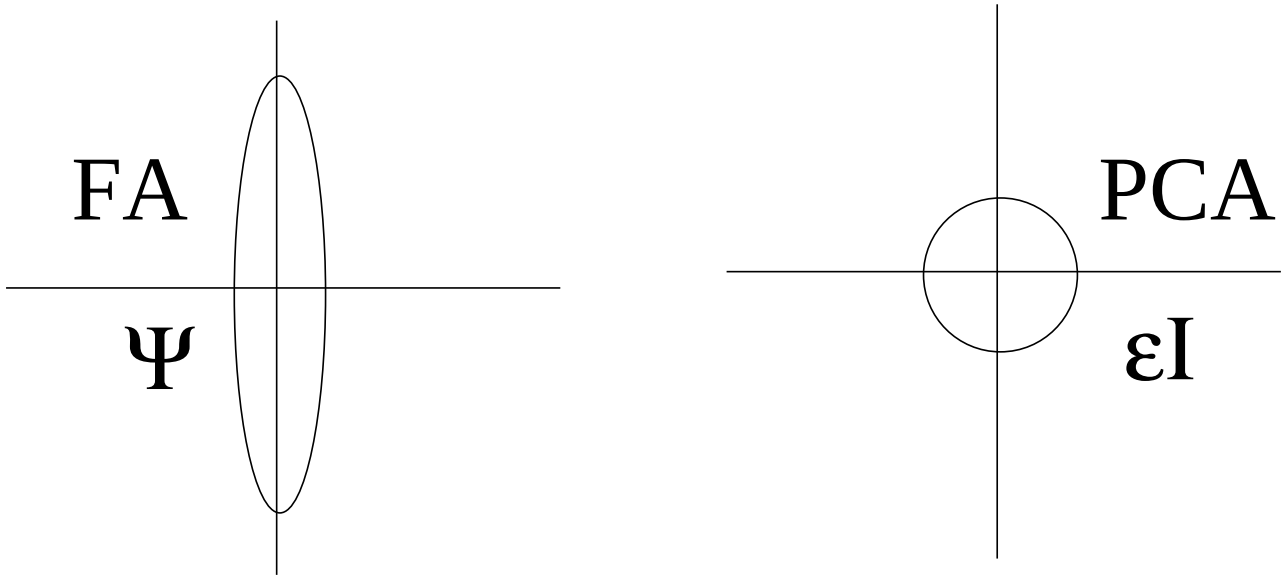
$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0, I)$$
$$p(\mathbf{y}|\mathbf{z},\theta) = \mathcal{N}(\mathbf{y}|\mu + \Lambda\mathbf{z}, \sigma^2 I)$$

  μ is the mean vector,
  columns of Λ are the *principal components* (usually orthogonal),
  σ^2 is the *global sensor noise*.

- This model can also be fit using EM. (Or directly...)

Gaussians are Footballs in High-D

- Recall the intuition that Gaussians are hyperellipsoids.
- Mean == centre of football
  Eigenvectors of covariance matrix == axes of football
  Eigenvalues == lengths of axes
- In FA our football is an axis aligned cigar.
  In PPCA our football is a sphere of radius σ^2.



Inference is Linear

- In both FA and PCA we can analytically infer the posterior over the latent variables given an observation vector:

$$
\begin{aligned}
p(\mathbf{z}|\mathbf{y}) &= \mathcal{N}(\mathbf{z}|\mathbf{m}, \mathbf{V}) \\
\mathbf{V} &= I - \Lambda^\top(\Lambda\Lambda^\top + \Psi)^{-1}\Lambda \\
&= (I + \Lambda^\top\Psi^{-1}\Lambda)^{-1} \\
\mathbf{m} &= \Lambda^\top(\Lambda\Lambda^\top + \Psi)^{-1}(\mathbf{y} - \mu) \\
&= \mathbf{V}\Lambda^\top\Psi^{-1}(\mathbf{y} - \mu)
\end{aligned}
$$

- Note: inference of the posterior mean is just a linear operation.

$$\mathbf{m} = \beta(\mathbf{y} - \mu)$$

  β can be computed beforehand given model parameters.

- Also: posterior covariance does not depend on observed data!

$$\mathrm{cov}[\mathbf{z}|\mathbf{y}] = \mathbf{V} = (I + \Lambda^\top\Psi^{-1}\Lambda)^{-1}$$

Likelihood Functions

- For both FA and PPCA, the data model is Gaussian.
  Thus, the likelihood function is simple:

$$\begin{aligned}\ell(\theta;\mathcal{D}) &= -\frac{N}{2}\log|\Lambda\Lambda^\top+\Psi| - \frac{1}{2}\sum_n(\mathbf{y}^n-\mu)^\top(\Lambda\Lambda^\top+\Psi)^{-1}(\mathbf{y}^n-\mu)\\ &= -\frac{N}{2}\log|\mathbf{V}| - \frac{1}{2}\text{trace}\left[\mathbf{V}^{-1}\sum_n(\mathbf{y}^n-\mu)(\mathbf{y}^n-\mu)^\top\right]\\ &= -\frac{N}{2}\log|\mathbf{V}| - \frac{1}{2}\text{trace}\left[\mathbf{V}^{-1}\mathbf{S}\right]\end{aligned}$$

  $\mathbf{V}$ is model covariance; $\mathbf{S}$ is sample data covariance.

- In other words, we are trying to make the constrained model covariance as close as possible to the observed covariance, where "close" means the trace of the ratio.

- Thus, the sufficient statistics are the same as for the Gaussian: mean $\sum_n \mathbf{y}^n$ and covariance $\sum_n(\mathbf{y}^n-\mu)(\mathbf{y}^n-\mu)^\top$.

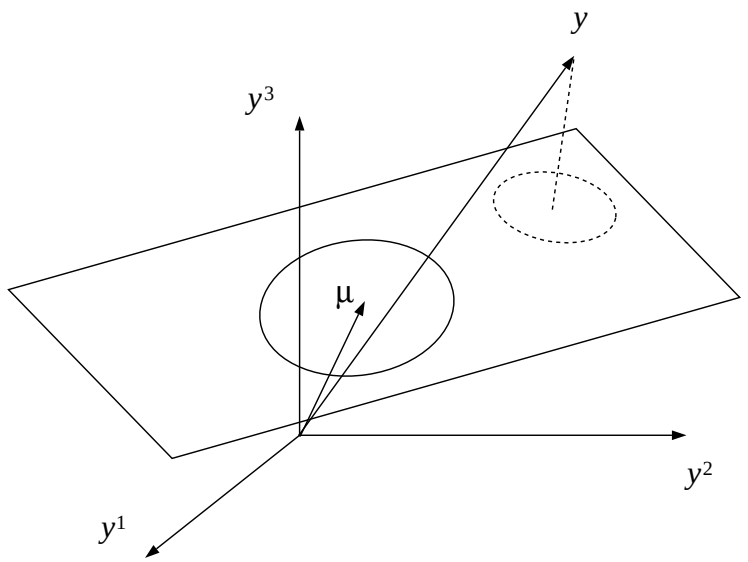
Direct Fitting

- For FA the parameters are coupled in a way that makes it impossible to solve for the ML params directly.
  We must use EM or other nonlinear optimization techniques.
- But for PCA, the ML params can be solved for directly:
  The k^{th} column of Λ is the k^{th} largest eigenvalue of the sample covariance $\mathbf{S}$ times the associated eigenvector.
- The global sensor noise σ^2 is the sum of all the eigenvalues smaller than the k^{th} one.
- This technique is good for initializing FA also.
- We can't make the sensor noise unconstrained, or else we would always get a perfect fit!

Zero Noise Limit

- The traditional PCA model is actually a limit as $\sigma^2 \to 0$.
  The model we saw is actually called "probabilistic PCA".
- However, the ML parameters Λ^* are the same.
  The only difference is the global sensor noise σ^2.
- In the zero noise limit inference is easier: orthogonal projection.

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



Snap-Shot Method

- If your data is zero mean, knowing the eigenvectors of the Gram matrix $G_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$ is just as good as knowing the eigenvectors of the sample covariance $\mathbf{S} = \sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\top$.
- Why? Because:

$$\begin{aligned} \mathbf{w} &= \mathbf{X}^\top \mathbf{v} \\ \mathbf{S}\mathbf{v} &= \alpha \mathbf{v} \\ \mathbf{X}\mathbf{X}^\top \mathbf{v} &= N\alpha \mathbf{v} \\ \mathbf{X}^\top \mathbf{X}\mathbf{X}^\top \mathbf{v} &= N\alpha \mathbf{X}^\top \mathbf{v} \\ \mathbf{X}^\top \mathbf{X}\mathbf{w} &= N\alpha \mathbf{w} \end{aligned}$$

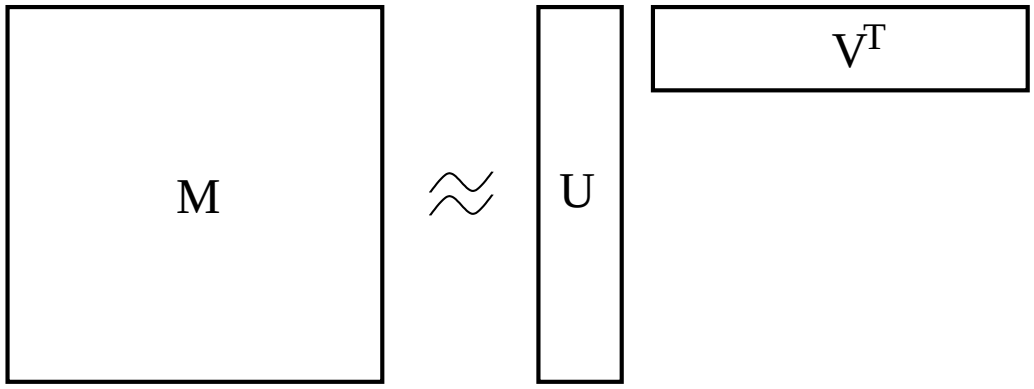
- So if $N < d$ it is faster to compute this eigendecomposition than the original one. Intuition: N points span a space of at most dimension $N - 1$. The eigenvalues are the same; and the eigenvectors are linear transforms of each other. (We say that the two matrices are "similar".)

SVD

- Think of each row as a data vector, do PCA. Now think of each column as a data vector and do PCA again.
- If you had the SVD of your raw data matrix you would be almost there (exept for those troublesome means):

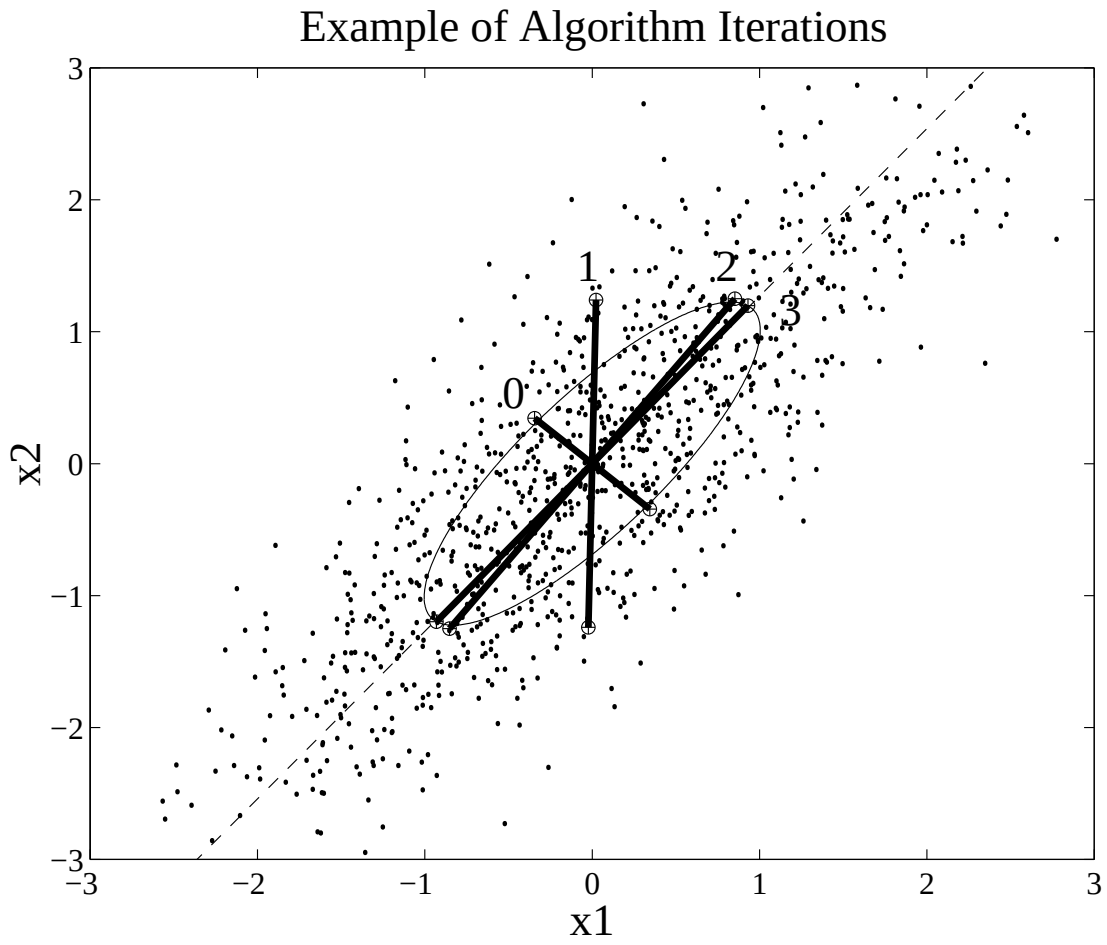
$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$$
$$\mathbf{X}\mathbf{X}^\top = \mathbf{U}\mathbf{S}^2\mathbf{U}^\top$$
$$\mathbf{X}^\top\mathbf{X} = \mathbf{V}\mathbf{S}^2\mathbf{V}^\top$$

- The left/right singular vectors are the eigenvectors of columnwise/rowwise data. The eigenvalues are the squares of the corresponding singular values.



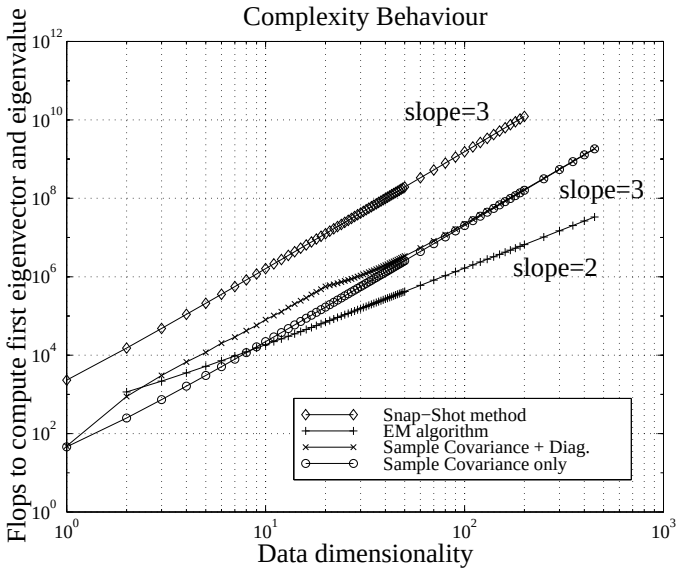
EM Algorithm for PCA

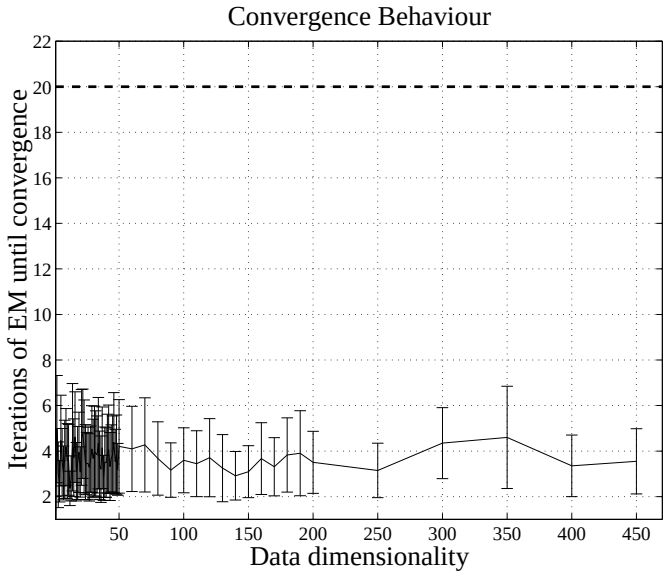
- The zero noise limit (PCA) also has a nice EM algorithm:
  E-step: $\mathbf{Z} = (\mathbf{C}^\top \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{Y}$
  M-step: $\mathbf{C}^{new} = \mathbf{Y}\mathbf{Z}^\top (\mathbf{Z}\mathbf{Z}^\top)^{-1}$



EM Algorithm for PCA

- The zero noise limit (PCA) also has a nice EM algorithm:
  E-step: $\mathbf{Z} = (\mathbf{C}^\top \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{Y}$
  M-step: $\mathbf{C}^{new} = \mathbf{Y}\mathbf{Z}^\top (\mathbf{Z}\mathbf{Z}^\top)^{-1}$
- This scales as $O(dNk)$ for extracting k principal components from N datappoints in d-dimensions. Much faster than either regular PCA or Snap-Shot when both d and N are huge but k is small.





EM Algorithm for PCA

- The zero noise limit (PCA) also has a nice EM algorithm:
  E-step: $\mathbf{Z} = (\mathbf{C}^\top \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{Y}$
  M-step: $\mathbf{C}^{new} = \mathbf{Y}\mathbf{Z}^\top (\mathbf{Z}\mathbf{Z}^\top)^{-1}$
- It can also deal very nicely with missing data.
  Imagine that $\mathbf{y}$ is partitioned into $\mathbf{y} = [\mathbf{a}; \mathbf{b}]$ where $\mathbf{a}$ is the known part of $\mathbf{y}$ and $\mathbf{b}$ is the missing part.
  Similarly, imagine that $\mathbf{C}$ is partitioned into $\mathbf{C} = [\mathbf{D}; \mathbf{E}]$ where $\mathbf{D}$ corresponds to the known portion of $\mathbf{y}$ and $\mathbf{E}$ corresponds to the mising portion.
- Now what you do in the E step is:

$$\mathbf{z} = (D^\top D)^{-1} D^\top \mathbf{a}$$
$$\mathbf{b} = E\mathbf{z}$$

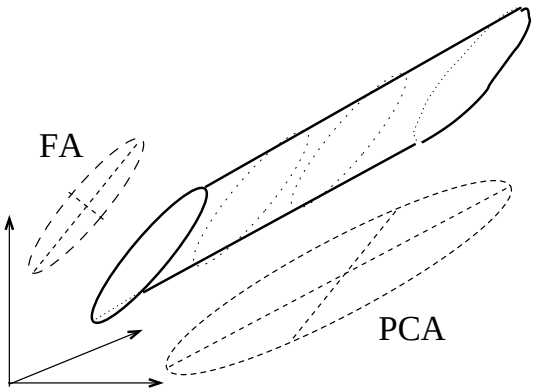
- Finally, in the M-step you use this $\mathbf{z}$ and the vector $[\mathbf{a}, \mathbf{b}]$ as usual.

Scale Invariance in Factor Analysis

- In FA the *scale* of the data is unimportant: we can multiply y_i by α_i without changing anything:

$$\begin{aligned} \mu_i &\leftarrow \alpha_i \mu_i \\ \Lambda_{ij} &\leftarrow \alpha_i \Lambda_{ij} \quad \forall j \\ \Psi_i &\leftarrow \alpha_i^2 \Psi_i \end{aligned}$$

- However, the *rotation* of the data *is* important.
- FA looks for directions of large correlation in the data, so it is not fooled by large variance noise.

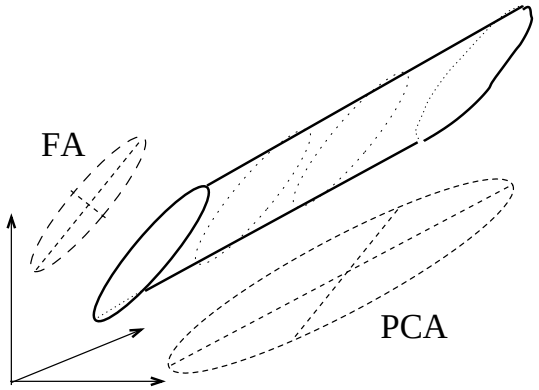


Rotational Invariance in PCA

- In PCA the *rotation* of the data is unimportant: we can multiply the data $\mathbf{y}$ by and rotation $\mathbf{Q}$ without changing anything:

$$\mu \leftarrow \mathbf{Q}\mu$$
$$\Lambda \leftarrow \mathbf{Q}\Lambda$$
$$\Psi \leftarrow \text{unchanged}$$

- However, the *scale* of the data *is* important.
- PCA looks for directions of large variance, so it will chase big noise directions.

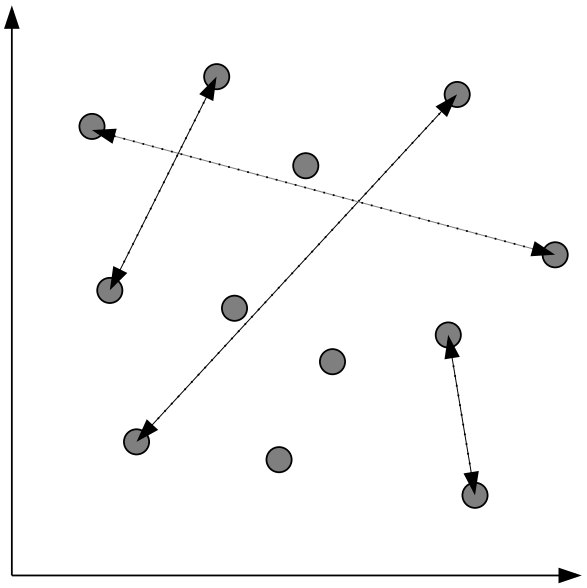


MDS: Multidimensional Scaling

- So far we have considered linear mappings which capture directions of data correlation or variance.
- Another objective is to capture pairwise similarities between the datapoints using distances after the mapping.
- In particular, we can find embeddings $\mathbf{x}_i$ to minimize:

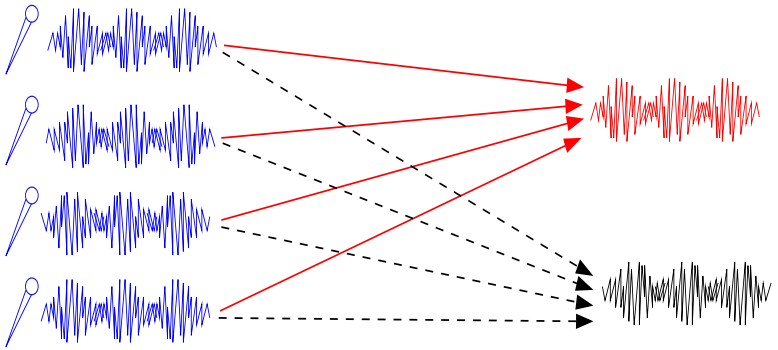
$$\sum_{ij} f(D_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|)$$

- Torgerson ('52) noticed a linear algebra trick to minimize this when $f(\cdot)$ is squared error. (equivalent to PCA)
- Often nonlinear squashing functions are applied to D_{ij} or used for $f(\cdot)$; this leads to non-metric MDS and variants like Sammon mapping.



Independent Components Analysis (ICA)

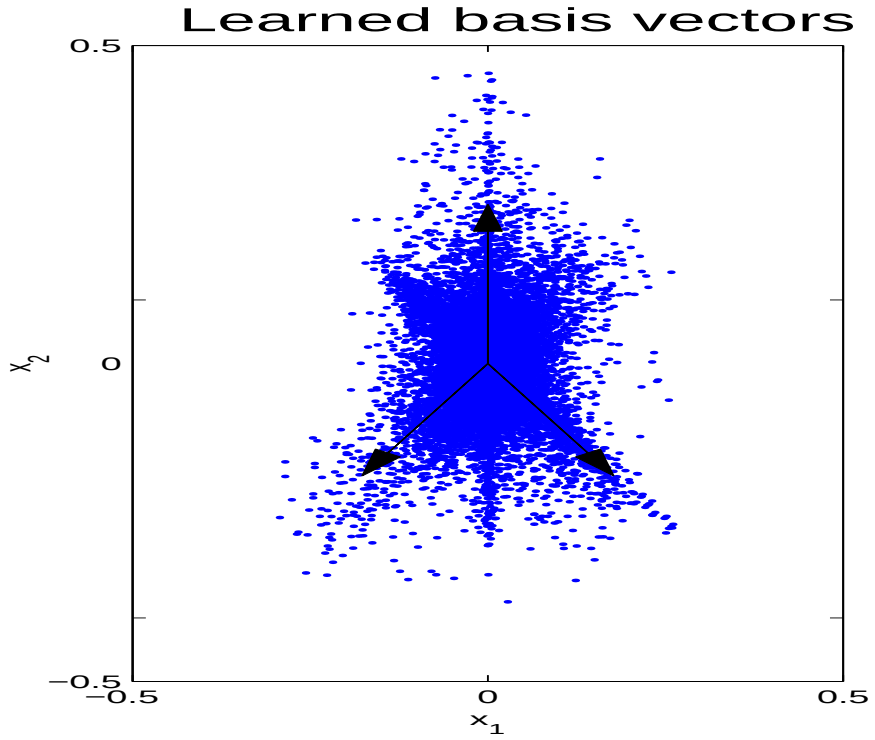
- ICA is another continuous latent variable model, like FA, but it has a *non-Gaussian* and *factorized* prior on the latent variables.
- This is good in situations where most of the factors are very small most of the time and they do not interact with each other. Example: mixtures of speech signals.



- The learning problem is the same: find the weights from the factors to the outputs and infer the unknown factor values. In the case of ICA the factors are sometimes called “sources”, and the learning is sometimes called “unmixing”.

Geometric Intuition

- Since the latent variables are assumed to be independent, we are trying to find a linear transformation of the data that recovers these independent causes.
- Often we use *heavy tailed* source priors, e.g. $p(z_i) \propto 1/\cosh(z_i)$.
- Geometric intuition: finding spikes in histograms.



ICA Model

- The simplest form of ICA has as many outputs as sources (square) and no sensor noise on the outputs:

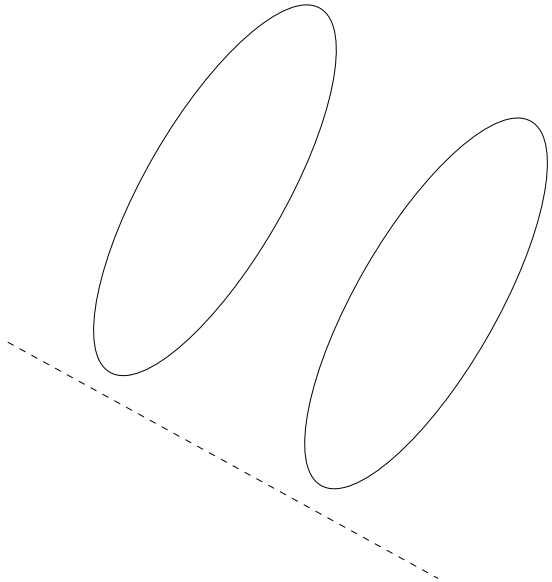
$$p(\mathbf{z}) = \prod_k p(z_k)$$

$$\mathbf{y} = \mathbf{V}\mathbf{z}$$

- Learning in this case can be done with gradient descent (plus some "covariant" tricks to make updates faster/more stable).
- If you keep the square $\mathbf{V}$ and use isotropic Gaussian noise on the outputs there is a simple EM algorithm (Welling & Weber).
- Much more complex cases have been studied also: nonsquare, convolutional, time delays in mixing, etc.
- But for that, we would need to get into time-series...

Projection Pursuit

- Diaconis and Freedman showed that most projections of high dimensional data are approximately Gaussian (normal).



- So search for projections that look non-Gaussian, because these should be "interesting" or at least "unusual" directions.
- What you can't have, you always want...
- Generally done in an iterative fashion, in which next direction is found after projecting out all previously found directions.
- Hence "pursuit". In fact, this was originally done manually.
- The contrast function is very similar to that in ICA.

Moving right along...

- Unsupervised Learning / Density Estimation / Manifolds
- Linear (Projection) Methods

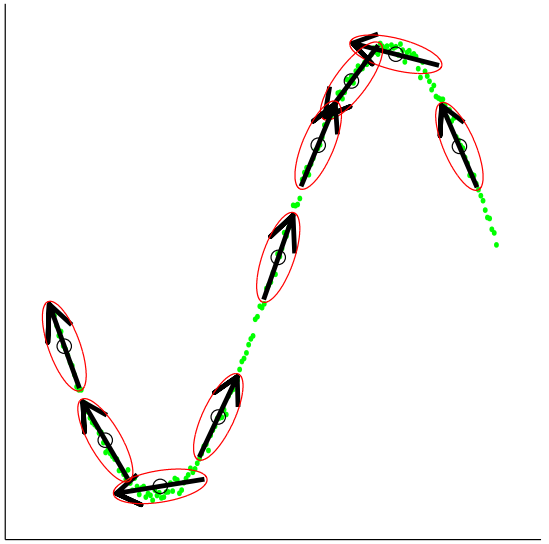
- Locally Linear (Alignment) Methods
  Basic Idea:
  1) Cluster.
  2) Within each cluster do PCA.
  That's almost it, but there is a kink...
  and then a variety of solutions

  Global Coordination / Charting / Automatic Alignment

- Nonlinear (Embedding) Methods

Mixtures of Dimensionality Reducers

- The next logical step: globally nonlinear but locally linear.
- Try a model that has two kinds latent variables: one discrete cluster, and one vector of continuous causes.
- Such models simultaneously do clustering, and within each cluster, dimensionality reduction. Great idea!
- To a good approximation, we can represent smooth manifolds by collections of simpler models, each of which describes a locally linear neighborhood.
- If our data comes from a smooth manifold, then the local coordinates of different models in their region of validity can be related to the global coordinates.



Mixtures of Factor Analyzers

- The simplest version of this is the *mixture of factor analyzers*.

$$
\begin{aligned}
p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}|0, I) \qquad p(k) = \alpha_k \\
p(\mathbf{y}|\mathbf{z}, k, \theta) &= \mathcal{N}(\mathbf{y}|\mu_k + \Lambda_k \mathbf{z}, \Psi) \\
p(\mathbf{y}|\theta) &= \sum_k \int_{\mathbf{z}} p(k)p(\mathbf{z})p(\mathbf{y}|\mathbf{z}, k, \theta)d\mathbf{z} \\
&= \sum_k \mathcal{N}(\mathbf{y}|\mu_k \, , \, \Lambda_k \Lambda_k^\top + \Psi)
\end{aligned}
$$

- Which is a *constrained mixture of Gaussians*.
- This is like a mixture of linear experts, using a logistic regression gate, with missing inputs.
- Fitting procedure?
  Maximum likelihood, using EM.
  (Of course!)

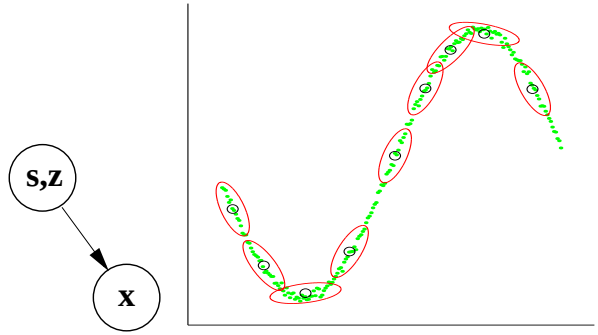
Local Linear Density Models

- **Mixtures of Factor Analysers (MFA)**

  Observed data $\mathbf{x} \in \mathcal{R}^D$

  Latent variables

  $s \in \{1, 2, \ldots, S\}, \boldsymbol{z}_s \in \mathcal{R}^d$



$$P(\mathbf{x}, s, \boldsymbol{z}_s) = P(\mathbf{x}|s, \boldsymbol{z}_s)P(s)P(\boldsymbol{z}_s)$$

$$P(\boldsymbol{z}_s) = \frac{1}{(2\pi)^{d/2}} \exp\left\{-\frac{1}{2}\boldsymbol{z}_s{}^\top \boldsymbol{z}_s\right\}$$

$$P(\mathbf{x}|s, \boldsymbol{z}_s) = \frac{|\Psi_s|^{-1/2}}{(2\pi)^{D/2}} \exp\left\{-\frac{1}{2}\left[\mathbf{x} - \boldsymbol{\mu}_s - \Lambda_s \boldsymbol{z}_s\right]^\top \Psi_s^{-1}\left[\mathbf{x} - \boldsymbol{\mu}_s - \Lambda_s \boldsymbol{z}_s\right]\right\}$$

- **Learning Parameters**

  Estimate means $\boldsymbol{\mu}_s$, loadings Λ_s, noises Ψ_s, priors p_s.

  Maximum likelihood: max. $\log P(\mathbf{x})$ on training set.

  Tractable, efficient EM algorithm.

A Problem w/Maximum Likelihood

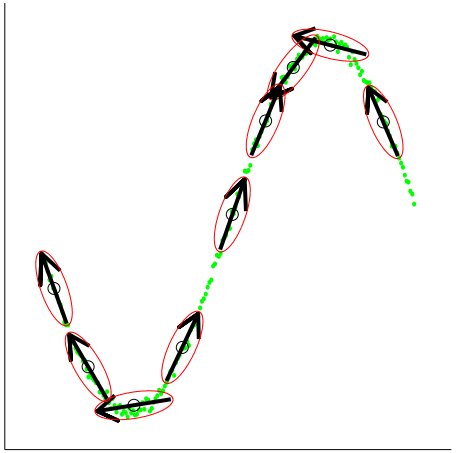
- **Parameter Degeneracy**
  Marginal distribution, $P(\mathbf{x})$ (sum out latent variables):

$$P(\mathbf{x}) = \sum_s p_s \frac{|\Lambda_s \Lambda_s^\top + \Psi_s|^{-1/2}}{(2\pi)^{D/2}} \exp\left\{ -\frac{1}{2} [\mathbf{x} - \boldsymbol{\mu}_s]^\top \left(\Lambda_s \Lambda_s^\top + \Psi_s\right)^{-1} [\mathbf{x} - \boldsymbol{\mu}_s] \right\}$$

  Invariance: $\Lambda_s \rightarrow \Lambda_s R_s$, for $R_s R_s^\top = I$
  R_s allow **arbitrary rotations** of local coordinates.

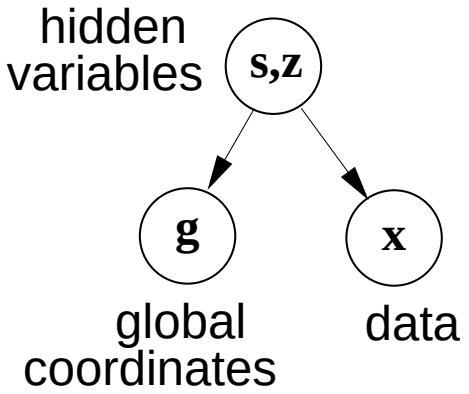
- **Bad Internal Codes**
  Likelihood is invariant to these transformations. Thus, normal EM estimation in MFAs does not favor models whose local coordinate systems are aligned in a consistent way.

- Instead, ML produces models whose internal representations change **unpredictably** as one traverses connected paths on the manifold.



Achieving Local Consistency by Coordination/Alignment

- **Want Local Agreement**
  Idea: at neighbourhood boundaries, **coactivated** local models should agree on global coordinates $\mathbf{g}$. Learn a mapping from local to global coordinates:



$$\mathbf{g}(\boldsymbol{z}_s, s) \;=\; A_s \boldsymbol{z}_s + \boldsymbol{\kappa}_s$$

  Additional "coordination" weights $A, \boldsymbol{\kappa}$ must also be learned from examples.

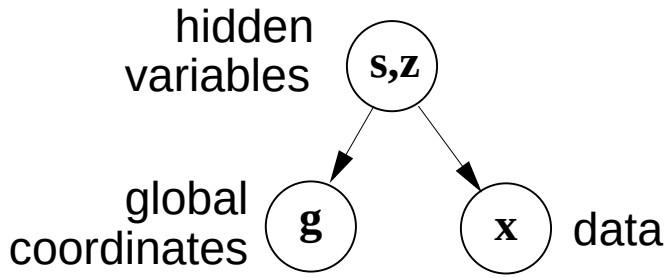
- **Goal: Get Latent Variables to Agree**
  Want to **stitch together** the local coordinate systems so that $\mathbf{g}$ changes smoothly and continuously as one traverses a connected path on the data manifold, even across the domains of many different local models.

  Subtle point: This can affect estimation of model as well.

Idea: Probabilistic Global Coordination

By treating the coordinates $\mathbf{g}$ as latent variables, we can incorporate them into the probabilistic model:

$$P(\mathbf{g}|\boldsymbol{z}_s, s) = \delta(\mathbf{g} - A_s\boldsymbol{z}_s - \boldsymbol{\kappa}_s)$$
$$P(\mathbf{g}|\mathbf{x}) = \sum_s P(s|\mathbf{x})P(\boldsymbol{z}_s|\mathbf{x}, s)P(\mathbf{g}|\boldsymbol{z}_s, s)$$



cost: **KL Q(g) || P(g) − log P(x)**

When several models explain a point $\boldsymbol{x}_n$ with high probability, their posterior distributions for the global coordinates should be similar:

$$P(\mathbf{g}|\boldsymbol{x}_n, s_i) \approx P(\mathbf{g}|\boldsymbol{x}_n, s_j)$$

Global coordinates are independent of the model given the data.

$\Rightarrow$ **Penalize multimodal posteriors** $P(\mathbf{g}|\boldsymbol{x}_n)$.
Unimodal approximating distributions, $Q(\mathbf{g}|\boldsymbol{x}_n)$.

$$Q(\mathbf{g}, s|\boldsymbol{x}_n) = Q(\mathbf{g}|\boldsymbol{x}_n)Q(s|\boldsymbol{x}_n); \quad Q(\mathbf{g}|\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{g}_n, \Sigma_n); \quad Q(s|\boldsymbol{x}_n) = q_{ns}$$

A New Model: Coordinated MFA

Using $P(\mathbf{g}|\boldsymbol{x}_n)$ and $Q(\mathbf{g}|\boldsymbol{x}_n)$ we can do both density estimation and manifold parameterization using the probabilistic machinery.
Consider the objective function:

$$\Phi = \sum_n \log P(\boldsymbol{x}_n) - \lambda \, \mathrm{KL}\left[Q(\mathbf{g}, s|\boldsymbol{x}_n) \| P(\mathbf{g}, s|\boldsymbol{x}_n)\right]$$

Model parameters $\{p_s, \Lambda_s, \boldsymbol{\mu}_s, \Psi_s\}$.
Coordination parameters $\{A_s, \boldsymbol{\kappa}_s\}$.
Regularizing parameters $\{\boldsymbol{g}_n, \Sigma_n, q_{ns}\}$.
First term defines the **log-likelihood** of the data.
Second term, in the form of a Kullback-Liebler divergence, **penalizes** MFAs whose posterior distributions are inconsistent with the global coordinates.
The balancing of these terms leads to models that satisfy **simultaneous goals of modeling**, yielding good likelihood with coordinated factors that match the model's posterior distributions to unimodal approximations.

Variational Methods & Internal Representations

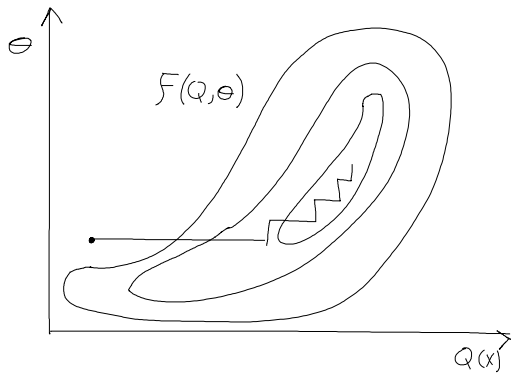
$$\Phi = \sum_n \log P(\boldsymbol{x}_n) - \lambda \, \mathrm{KL}\left[Q(\mathbf{g}, s|\boldsymbol{x}_n) \| P(\mathbf{g}, s|\boldsymbol{x}_n)\right] = \sum_n \mathcal{S}_n - \sum_{ns} q_{ns} \, \mathcal{E}_{ns}$$

$$\mathcal{S}_n = \mathcal{H}(Q_{g|n}) + \mathcal{H}(Q_{s|n}) \qquad \mathcal{E}_{ns} = -\langle \log P(\mathbf{g}, s, \boldsymbol{x}_n) \rangle_{Q_{g|n}}$$

- Our cross entropy regularizer is formally equivalent to the objective function used in **variational penalty methods**.
- It is a lower bound on log likelihood.
- Unexpected application: not to perform **approximate inference in intractable models**. but to compute auxiliary parameters and break a degeneracy in the original model's parameter space in a way that learns more powerful internal representations.
- As the parameter $\lambda \to 0$ we can only make modifications which exploit strict invariances in the parameterization.

Coordinated MFA Learning Algorithm

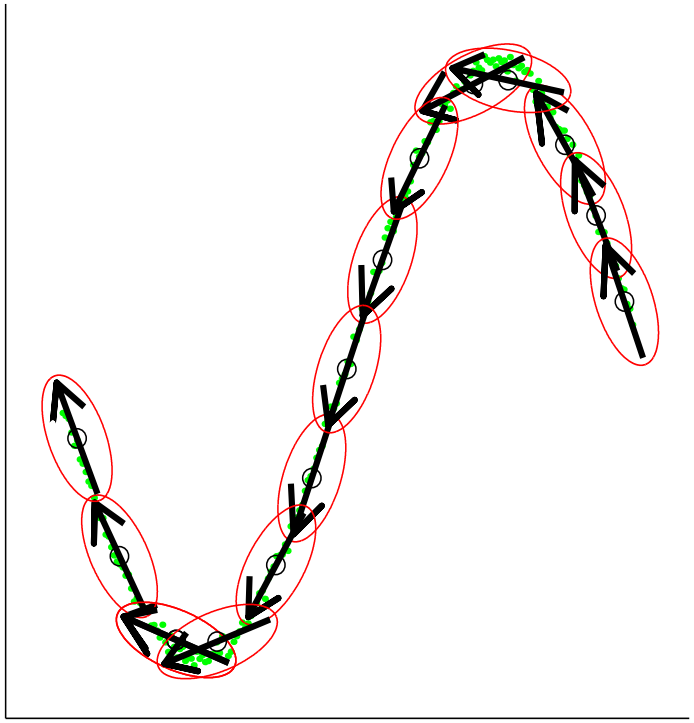
- **E-step**
  Iterate fixed-point equations to optimize the regularization parameters which control the approximate distributions $Q(\mathbf{g}, s|\boldsymbol{x}_n)$.
  Means $\boldsymbol{g}_n$, covariance matrices Σ_n, and mixture weights q_{ns} are determined for each data point.
- **M-step**
  Update the model parameters $\{p_s, \Lambda_s, \boldsymbol{\mu}_s, \Psi_s\}$, and the coordination parameters $\{A_s, \boldsymbol{\kappa}_s\}$ to maximize the objective function, simultaneously trying to model the data well and learn consistent representations.
- **Stable Learning**
  The algorithm is always increasing a well defined objective function. (May lower the log likelihood in exchange for better consistency or vice versa.)

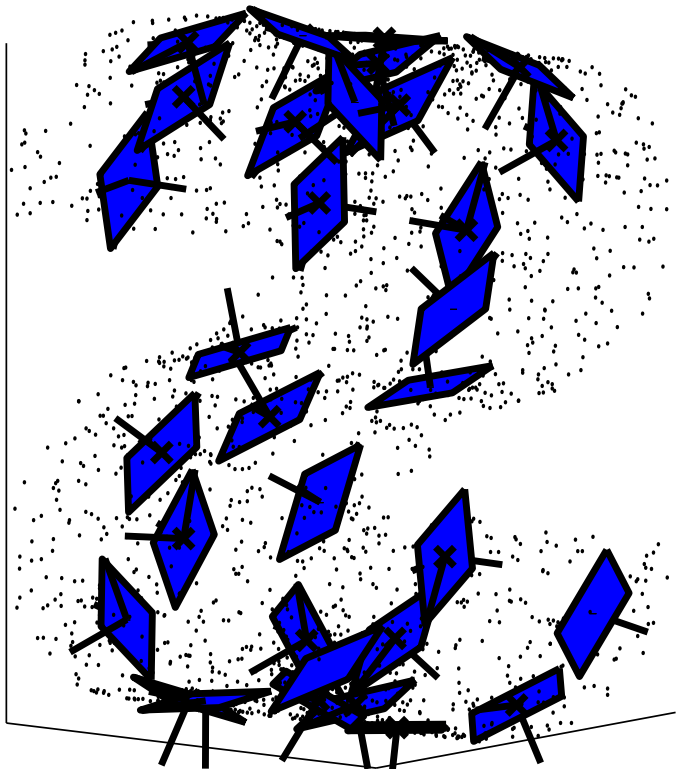


The obligatory demo...

No matter how entertaining you think you are,
it's best to have some demos.

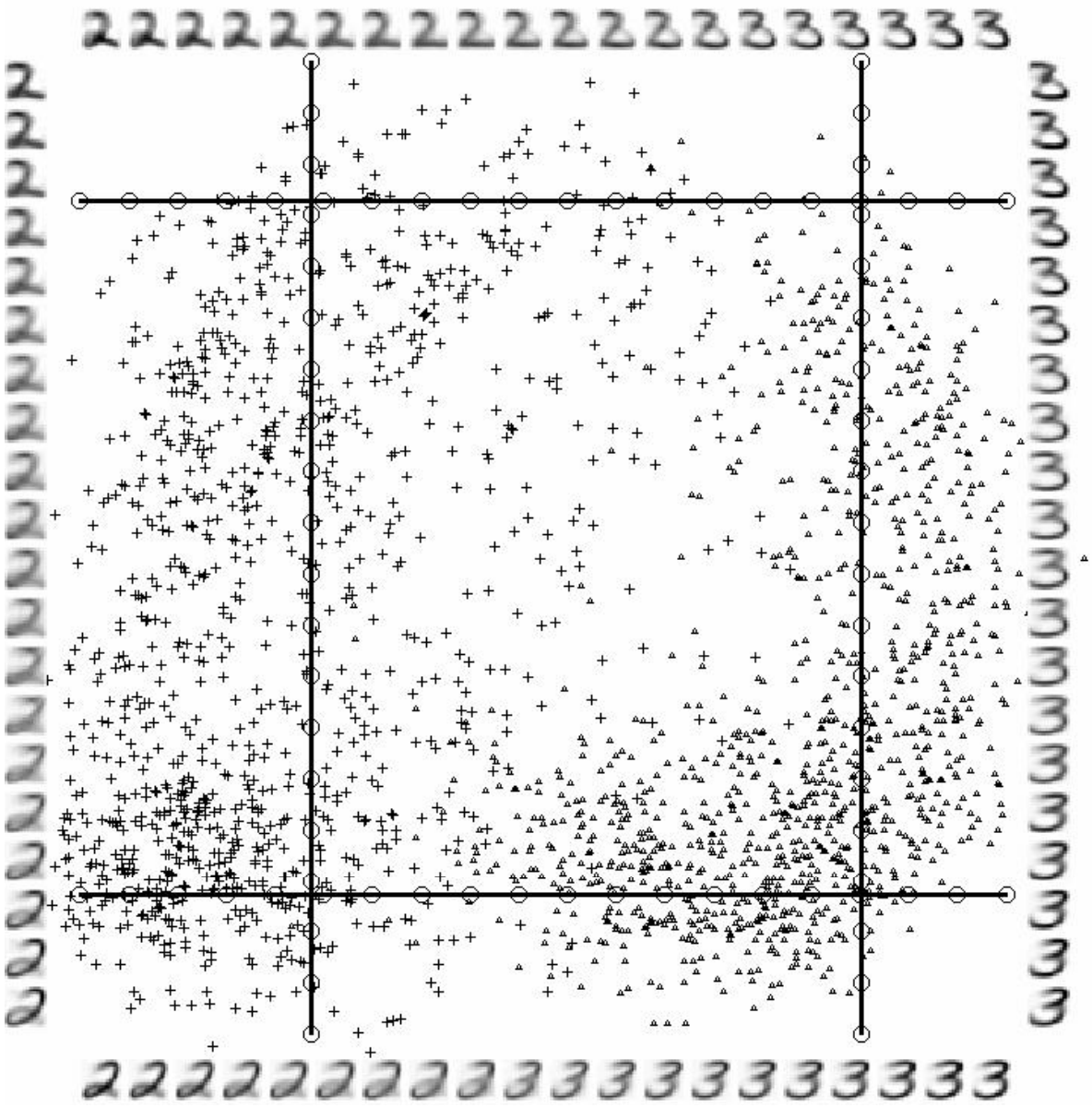
Synthetic Examples





See the movies!

Real Data: Digits



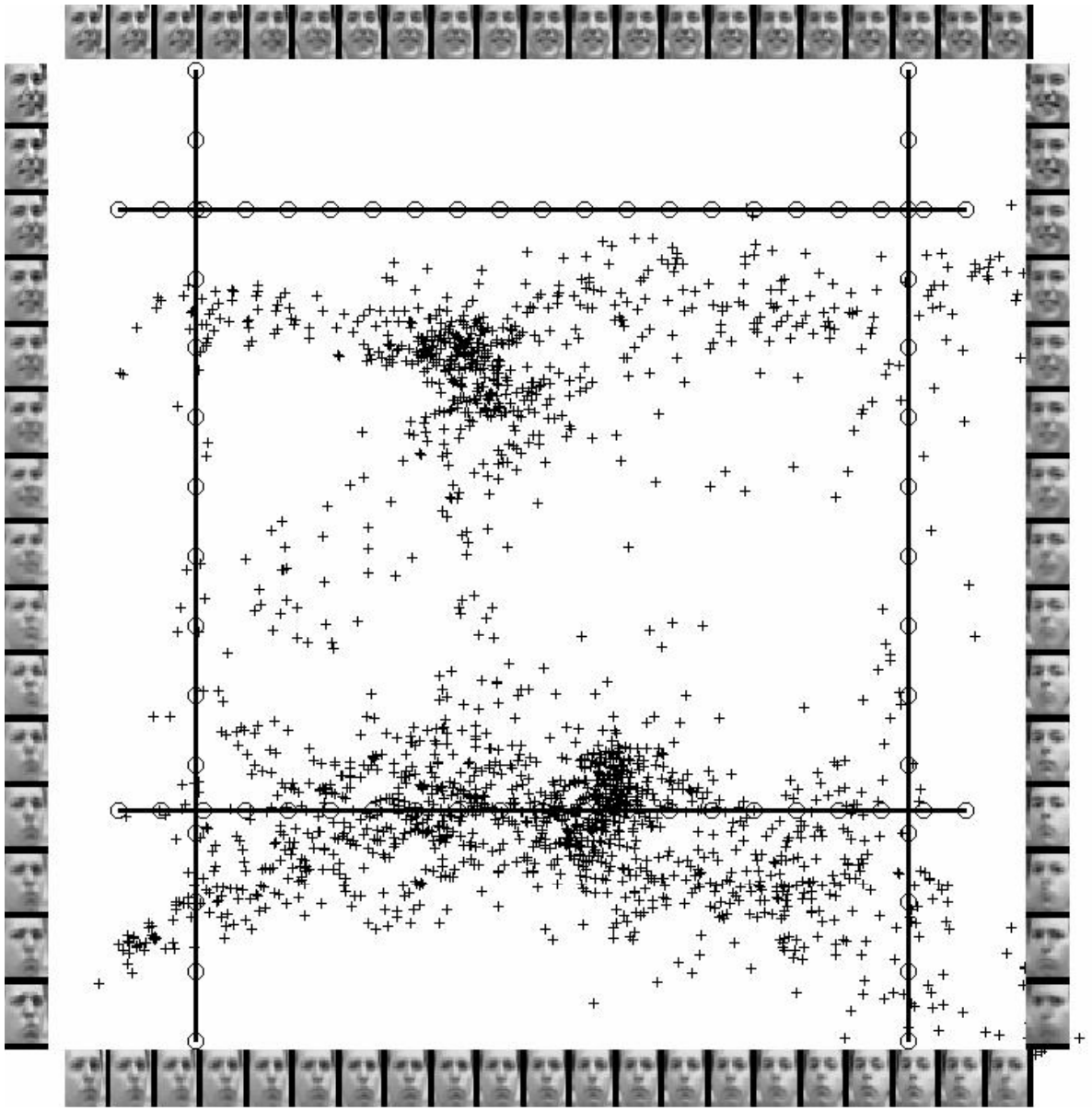
Digit Data

N=2200 grayscale photos

D=16×16 resolution.

K=12 neighbours Euclidean distance.

Real Data: Faces



Face Data

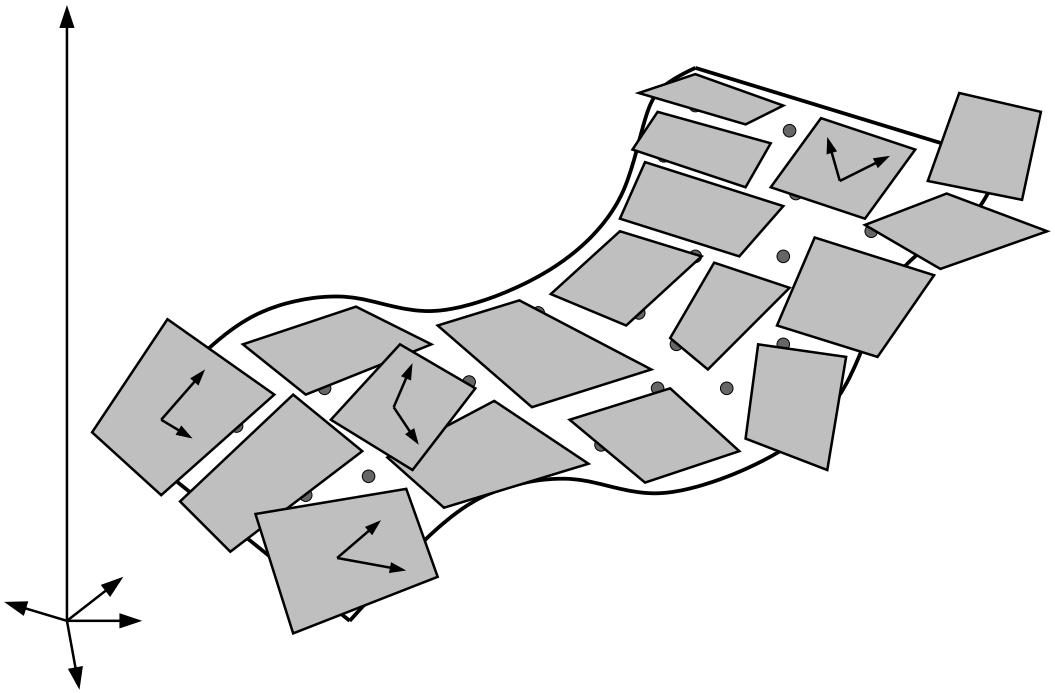
N=2000 grayscale photos

D=20×28 resolution.

K=12 neighbours Euclidean distance.

Other Coordination/Alignment Pointers

- Original credit for coordination – Revow & Hinton
- Global Coordination – Roweis, Saul, Hinton
- Post-Coordination:
  Automatic Alignment – Teh & Roweis; Verbeek
  Charting – Brand
- Correspondences – Ham, Saul, Lee; Verbeek, Roweis, Vlassis



Coffee time again...

- Unsupervised Learning / Density Estimation / Manifolds
- Linear (Projection) Methods
- Locally Linear (Alignment) Methods
- Nonlinear (Embedding) Methods

"Classic" nonlinear dimensionality methods, which try to build an explicit approximation to the manifold using the observations as training data directly:

SOM,GTM,Principal Curves, nonlinear autoencoders

and many newer ones that we won't talk about
(e.g. Latent Variable GP, Stochastic Neighbour Embedding)

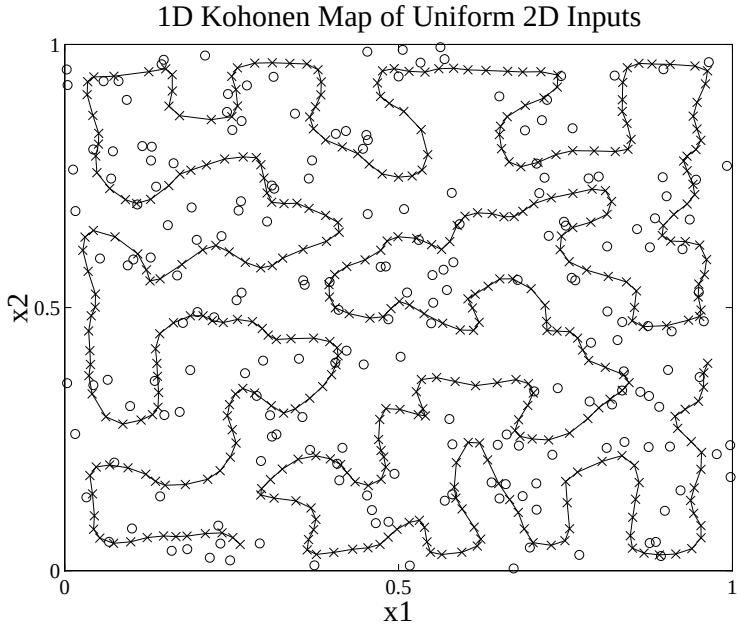
SOM: Self Organizing Maps

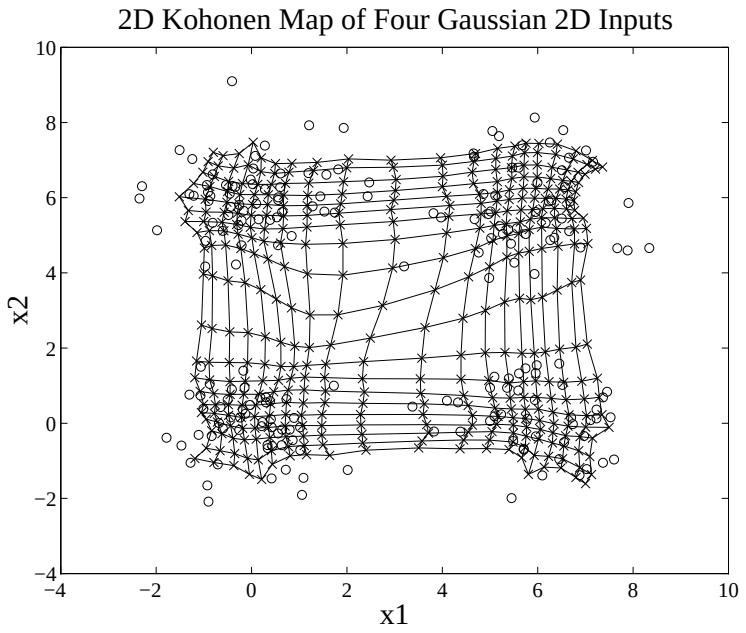
- Kohonen ('82) introduced the idea of building topology into a vector quantizer (k-means) model by associating with each prototype $\mathbf{x}_i$ a location in a low dimensional embedding space.
- Learning now proceeds as usual, except that when a prototype is updated, **its neighbouring prototypes (in the embedding) are also updated slightly**:

  $$\mathbf{x}_i \leftarrow \mathbf{x}_i + \alpha_{ii^*}(\mathbf{x} - \mathbf{x}_i) \qquad \forall \text{ neighbours } i \text{ of } i^*$$

  This makes codewords nearby in the embedding encode similar prototypes in the data space.
- The neighbourhood size α in the embedding space is annealed from large to small, so SOMs can be thought of as a way of finding a particular local minimum of the k-means algorithm with desirable structure in the prototypes.
- Unfortunately there is in general no cost function which SOM learning is always reducing. (But see elastic nets.)

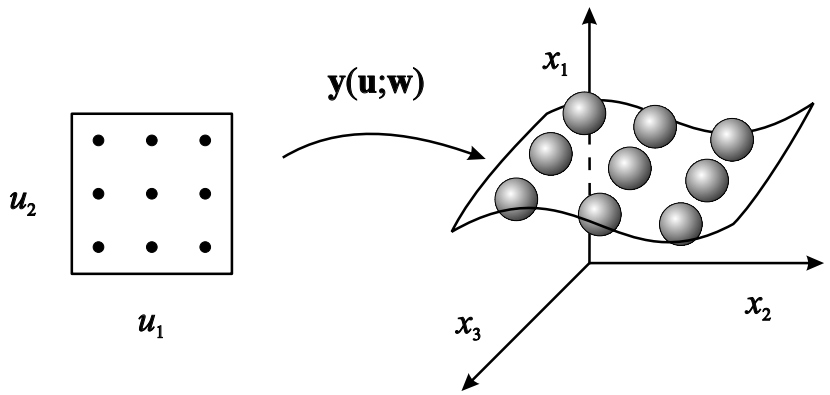
SOM examples





GTM:Generative Topographic Mapping

A probabilistic extension of SOMs which consists of a constrained mixture of Gaussians rather than a constrained k-means model.



- Hidden variables $\mathbf{x}$, observables $\mathbf{y}$, mapping $p(\mathbf{y}|\mathbf{x})$:

$$p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$$

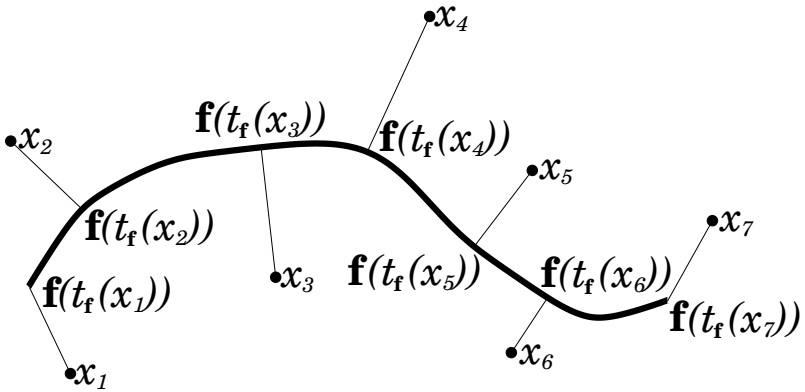
  Prior $p(\mathbf{x})$ & mapping $p(\mathbf{y}|\mathbf{x})$ chosen to make model tractable.

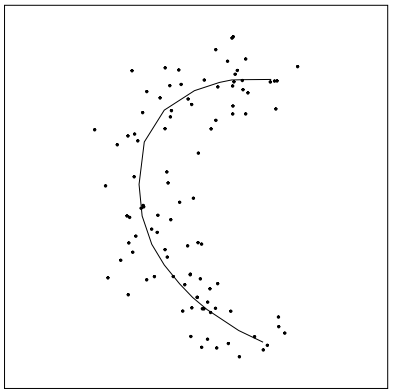
- In particular, $p(\mathbf{x})$ is a sum of delta-functions centred on a regular grid in latent space. The mapping is achieved using a fixed set of radial basis functions plus isotropic Gaussian noise:

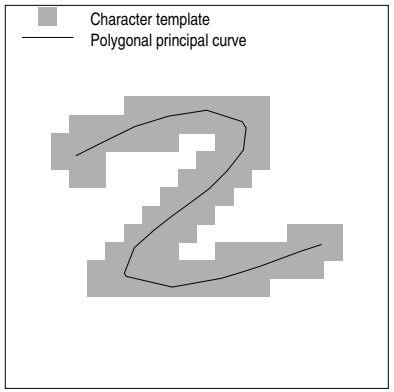
$$p(\mathbf{y}) = \sum_k \frac{1}{K} \left(\frac{\beta}{2\pi}\right) \exp\left[-\frac{\beta}{2} \|W\phi(\mathbf{x}_k) - \mathbf{y}\|^2\right]$$

Principal Curves

- A "self-consistent" curve passing through a data distribution such that the average of data which project to a point on the curve is that point. [Hastie&Stuetzle]
- Algorithm:
  P-step: project the data points $\mathbf{x}_i$ onto the current curve $\mathbf{x}(\lambda)$ giving λ_i.
  C-step: refit the curve by averaging all data points that project "nearby": $\mathbf{x}(\gamma) \leftarrow \langle \mathbf{x}_j \rangle_{j(\gamma)}$
- If curve is a line, it is the first principal component of the data.

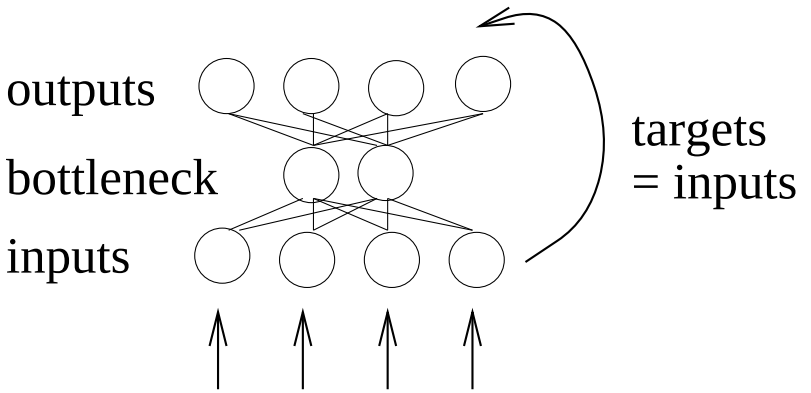






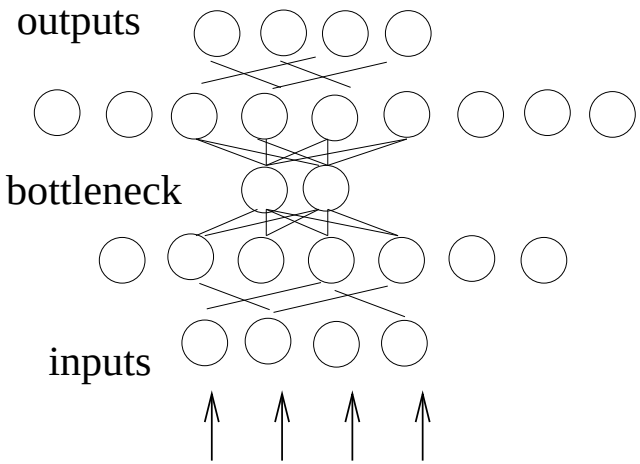
Autoencoder Neural Networks: Linear & Nonlinear

- If squared error is used on the output, a three layer autoencoder with a bottleneck of k units will learn to span the first k principal components of the data.



- This is because the encoder is linear, and given that the best decoder is also linear.
- What if we use a (nonlinear) MLP for encoding? Now the decoder must also be (a larger) MLP.

This leads to five layer autoencoders, which can in principle learn nonlinear manifolds but are difficult to train.



Coming around the final bend, looking strong, but a little behind the clock...

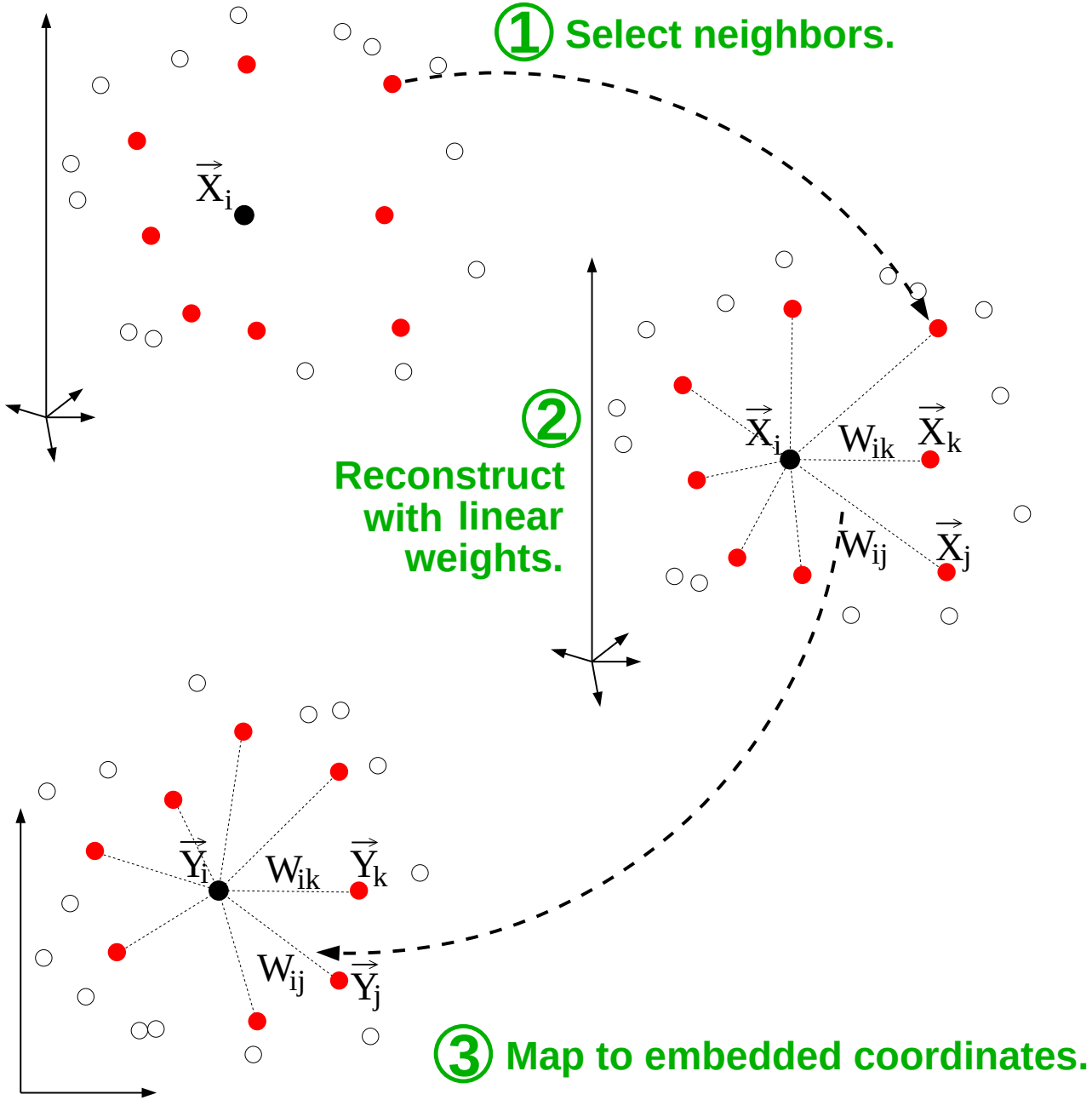
- Unsupervised Learning / Density Estimation / Manifolds
- Linear (Projection) Methods
- Locally Linear (Alignment) Methods
- Nonlinear (Embedding) Methods
  "Spectral" embedding methods, which set up an eigensystem whose solution simultaneously delivers the low-dimensional coordinates of all the training points at once:

  LLE, Isomap, Laplacian Eigenmaps, Kernel PCA, Spectral "Clustering"

Relationship Preserving Embeddings

- Problem formulation: embed high dimensional sensory "objects" into a low dimensional "description" space so that **nearby objects in original space remain nearby in the description (embedding) space**.
- Input: Original data $\boldsymbol{X}_i \in \Re^D$
- Output: New coordinates $\boldsymbol{Y}_i \in \Re^d$ ($d \ll D$) chosen so that some aspect(s) of the original relationships between the points preserved.
- Possibilities:
  Local geometry should stay constant.
  Geodesic distances should not change.
  Nearby points in $\boldsymbol{X}$ should remain nearby in $\boldsymbol{Y}$.
- Key assumptions:
  1) The manifold is **relatively smooth** and **well sampled**.
  2) It is **easy to compare nearby points**.

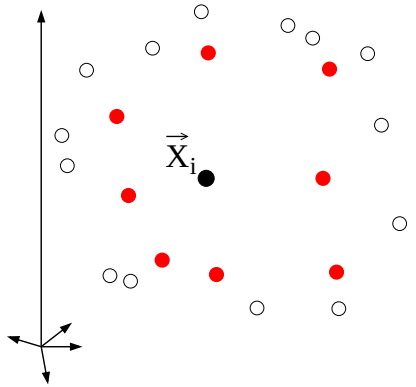
Locally Linear Embedding (LLE)



LLE Step 1: Assign neighbours.

For each point, choose a surrounding neighbourhood over which the manifold is roughly flat.

- Given: raw data consisting of N real-valued vectors $\boldsymbol{X}_i$, each of dimensionality D.
- Pick a few* neighbours for each point.
  Possible neighbourhood rules:
  - K nearest neighbours
  - points within a certain radius (ϵ-ball)
  - more sophisticated metric based on prior knowledge (e.g. local curvature)



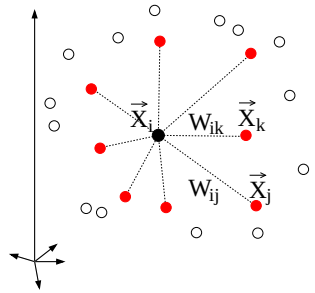
- Must have sufficient data such that underlying manifold is well sampled.

LLE Step 2: Locally linear fits.

Characterize the local geometry of each neighbourhood by weights that linearly reconstruct $\boldsymbol{X}_i$ from its neighbours.

- Local Geometry: Nearby data points lie on locally (almost) linear patches, so a linear reconstruction of $\boldsymbol{X}_i$ from its neighbours should have small error:

$$\min_W \mathcal{E}(W) = \sum_i |\mathbf{x} - \sum_j W_{ij} \boldsymbol{X}_j|^2$$



- Weight Constraints: weights must sum to unity: $\sum_j W_{ij} = 1$. Nonzero weights W_{ij} only for neighbours of $\boldsymbol{X}_i$.
  Optimal weights subject to these constraints can be found by solving a least squares problem.

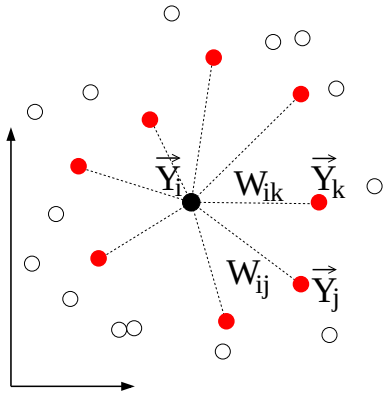
NB: solution is invariant to rotations, translations, and rescalings of $\boldsymbol{X}_i$ and its neighbours.

LLE Step 3: Compute embedding.

Map high-dimensional data observations $\boldsymbol{X}$ into low-dim. embedding vectors $\boldsymbol{Y}$ representing manifold coordinates.

- Embedding cost: $\min_Y \Phi(Y) = \sum_i |\mathbf{y} - \sum_j W_{ij} \boldsymbol{Y}_j|^2$

  This cost also measures linear reconstruction errors.
  But now we optimize $\boldsymbol{Y}_i$ while holding the weights W_{ij} fixed.



- Constraints:
  Centre the coordinates on the origin: $\sum_i \mathbf{y} = 0$.
  Avoid degenerate solutions by requiring unit covariance: $\frac{1}{N} \sum_i \mathbf{y}\mathbf{y}^\top = I$.

Optimal vectors $\boldsymbol{Y}_i$ subject to these constraints are found by solving an eigenvector problem. (Up to a rotation.)

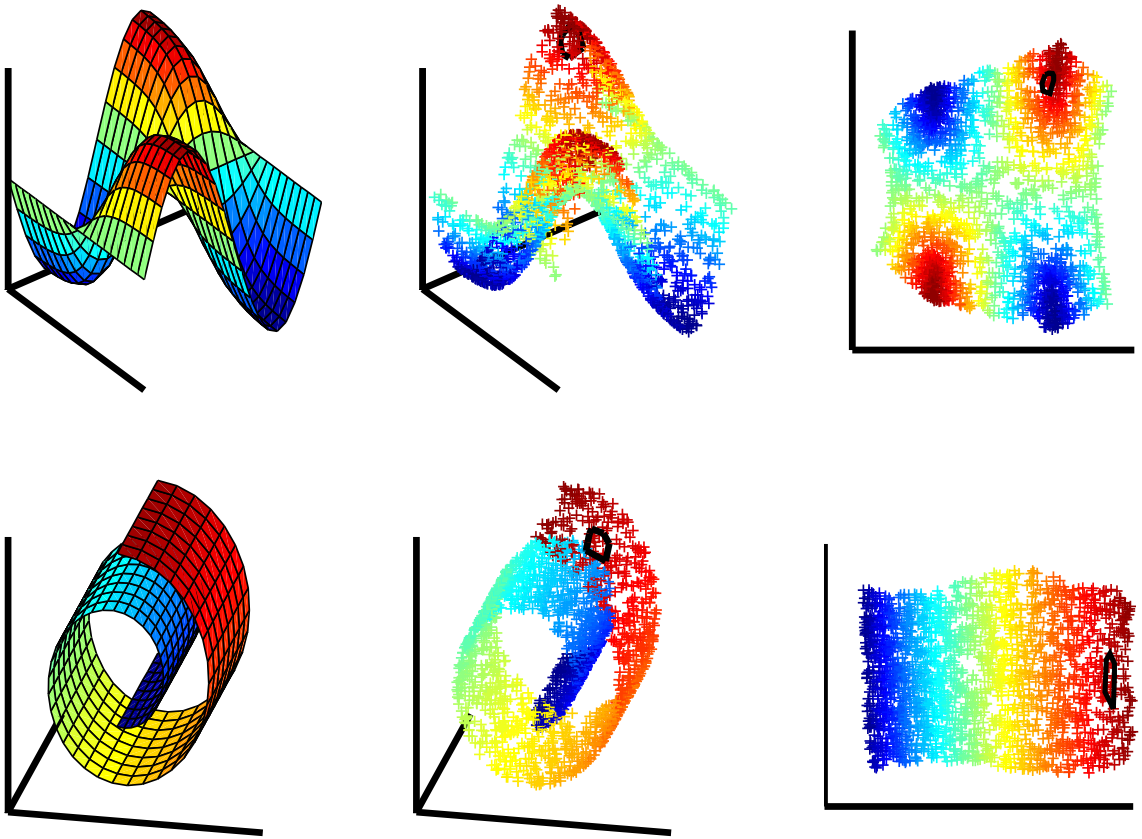
$[\Phi(Y) = \sum_{ij} M_{ij} (\mathbf{y} \cdot \boldsymbol{Y}_j)$ with $M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki} W_{kj}]$

Summary of LLE

- **Choose Neighbours**
  Assign K neighbours to each point.
  This is the algorithm's only free parameter.
  Only ever need to measure local distances.
  How you pick neighbours is crucial.
- **Compute Weights from Neighbours**
  Exploit local linearity to reconstruct each point from its neighbours using constrained least squares.
- **Generate Embedding from Weights**
  Compute low dimensional vectors $\boldsymbol{Y}$ that are well reconstructed by the same weights as the data by finding bottom eigenvectors of a sparse matrix.

Embedding dims are ordered and do not change as more added.
One pass algorithm. Always finds the exact global optimum of cost function. Has no learning rates, annealing schedules, etc.

LLE Examples

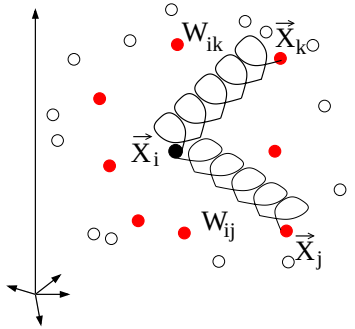


W characterizes local geometry

Weights W_{ij} capture local geometry of manifold near $\boldsymbol{X}_i$.

- Symmetries: optimal W_{ij} are invariant to data transformations
  - **rotations** $|R\mathbf{x} - \sum_j W_{ij} R\boldsymbol{X}_j|^2 = |\mathbf{x} - \sum_j W_{ij}\boldsymbol{X}_j|^2$ since $RR^T = I$
  - **translations** $|(\mathbf{x} + \boldsymbol{z}) - \sum_j W_{ij}(\boldsymbol{X}_j + \boldsymbol{z})|^2 = |\mathbf{x} - \sum_j W_{ij}\boldsymbol{X}_j|^2$ since $\sum_j W_{ij} = 1$
  - **scalings** $|\alpha\mathbf{x} - \sum_j W_{ij}\alpha\boldsymbol{X}_j|^2 = \alpha^2|\mathbf{x} - \sum_j W_{ij}\boldsymbol{X}_j|^2$ same W_{opt}
- Springs: Think of W_{ij} as setting the strength of a spring connecting $\boldsymbol{X}_i$ to $\boldsymbol{X}_j$. Why are W_{ij} also valid in Y-space?

Because the coordinate map $\boldsymbol{X} \leftrightarrow \boldsymbol{Y}$ is locally linear, i.e. a rotation, translation and scaling.



The springs are invariant to exactly these operations!

NB: Because of these weights, LLE achieves a more powerful embedding than one that preserves local distances to neighb.

LLE Eigenvector Calculation

The computational bottleneck in LLE is the last step in which we find the coordinate which minimize the embedding cost.

- **Eigenvectors minimize quadratic form**

  Rayleigh-Ritz theorem: $e(Q) = \frac{\sum_{ij} q_i q_j M_{ij}}{\sum_{ij} q_i q_j L_{ij}}$ is extremized by $\vec{q}$ which are largest/smallest eigenvectors of $L^{-1}M$

- **Structure of M is key**

  We seek smallest eigenvectors of the sparse matrix

  $$M = (I - W)^\top (I - W)$$

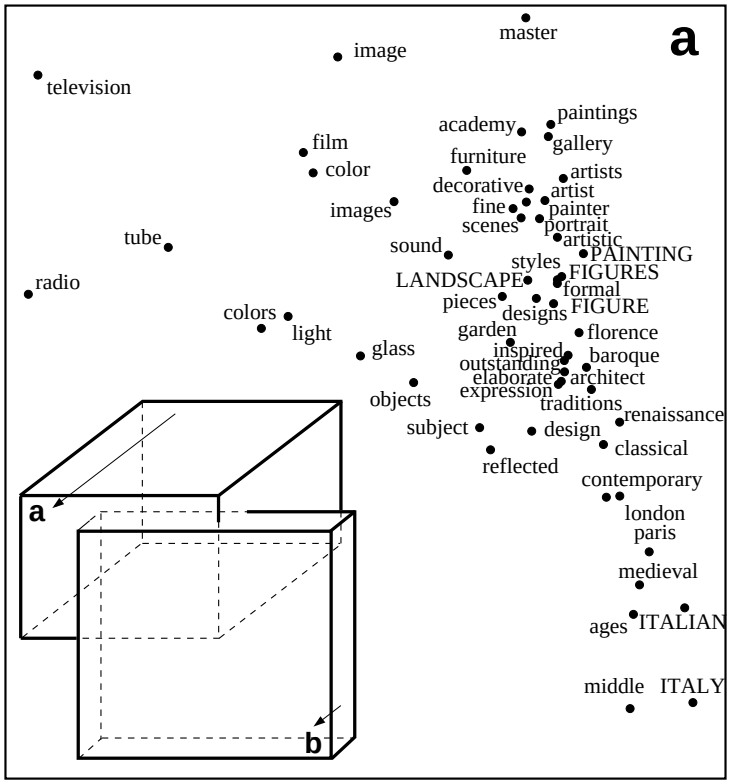
  1. Don't store M, only store W.
  2. Easy to multiply by M using $Mx = (x - Wx) - W^\top (x - Wx)$.
  3. Find eigenvectors using power methods with spectral shifting or gradient descent on RR quotient.
  4. Discard bottom eigenvector [1,1,1...] (has zero eigenvalue).

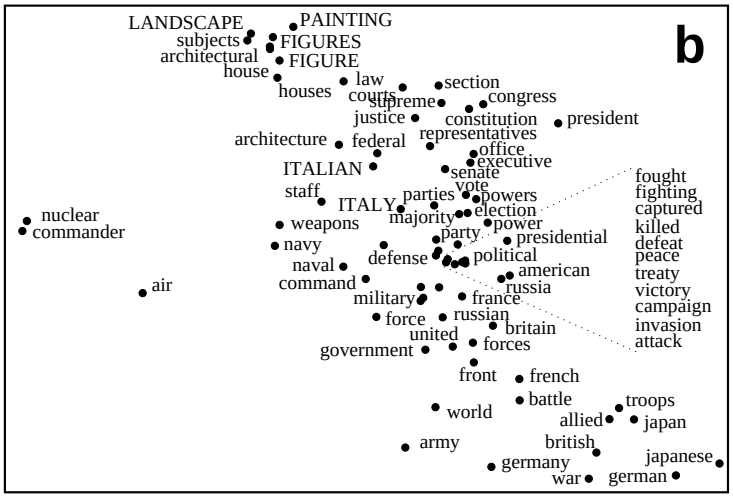
LLE Example: Text

Data: word-document counts of N=5000 words from D = 31000 encyclopedia entries.

(ability, able, academy, …, yellow, york, zone)

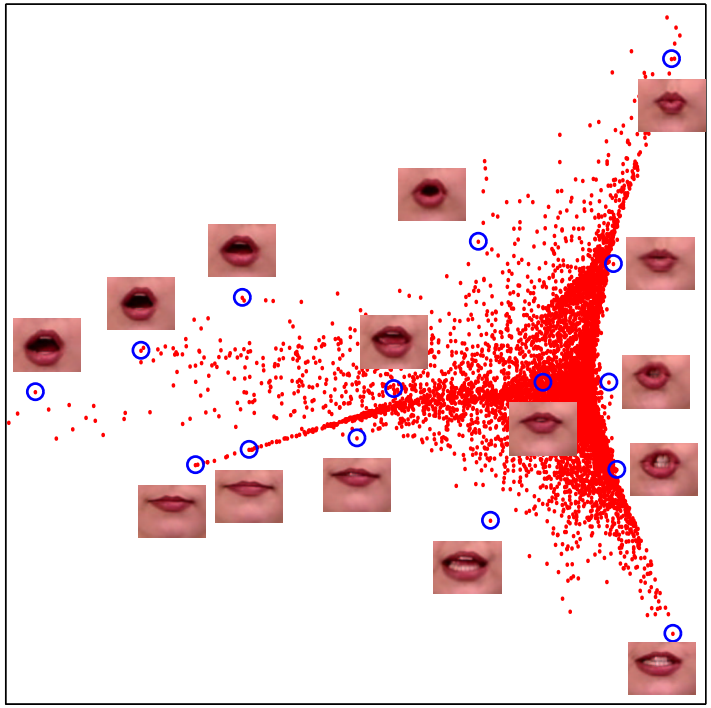
Neighbours: K=20 nearest using normalized dot products.

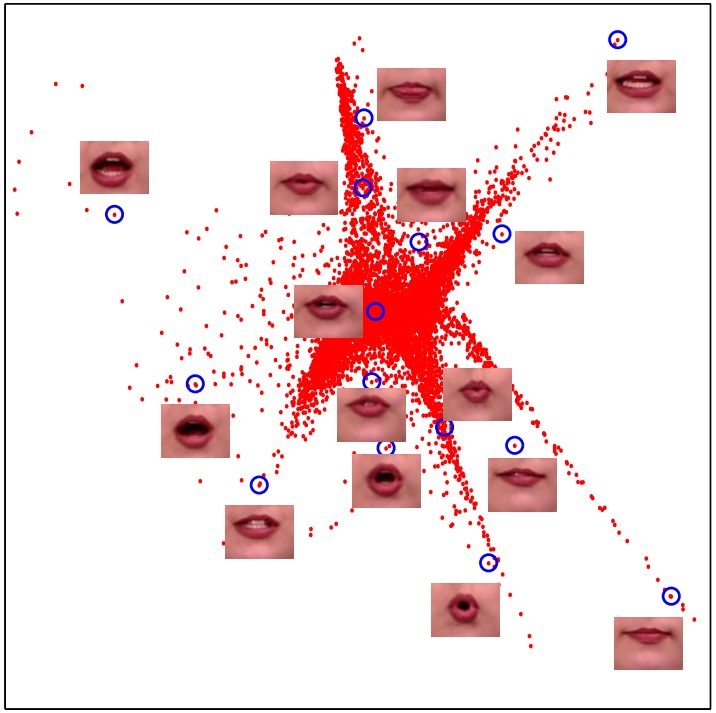




LLE Example: Lips

Data: many (N=9000) high resolution (D=27000) images of one speaker's mouth during speech.
Neighbours: K=16 nearest using Euclidean distance.





Isomap

- Josh Tenenbaum (’98) came up with a very clever way to apply MDS to achieve nonlinear embeddings.
- Build a graph on the data that is only locally connected, and then measure pairwise distance by path length on that graph.
- Now use those graph-derived distances as the target distances for MDS.

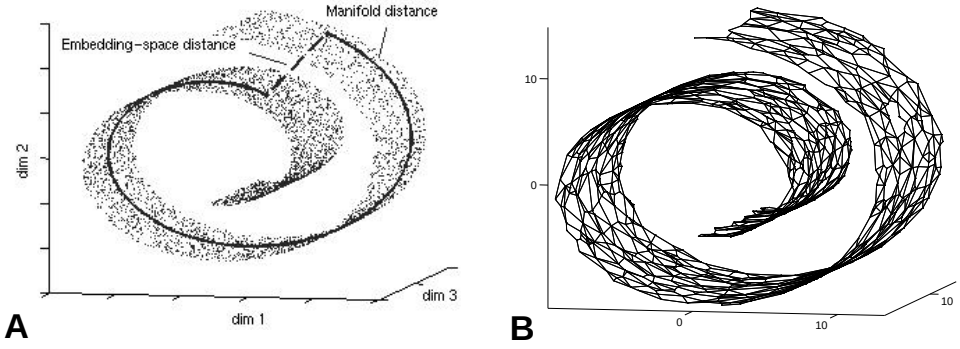


Figure 1: (A) Intrinsically two−dimensional manifold embedded nonlinearly in three−dimensional space, as a "swiss roll". Points nearby in the embedding space may be far apart on the underlying manifold. (B) ISOMAP efficiently estimates the true geodesic distances by computing shortest paths in a discrete network representation of the manifold.

- Originally Isomap generated a dense eigenproblem and could not deal with local scaling, but now there are sparse (“landmark”) and non-isometric (“conformal”) versions.

Laplacian Eigenmaps & Spectral Clustering

- Rather than asking to maintain local geometry (as in LLE) or to maintain all pairwise distance – emphasizing the large ones (as in Isomap) we can ask that nearby points not be moved too far away in the embedding.
- Define an undirected graph on neighbours, with weights:

$$W_{ij} = W_{ji} = \exp\left[-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2\right]$$

- Now look for a set of embeddings that minimize:

$$\sum_{ij} W_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|^2$$

  while still keeping $\mathbf{Y}$ of rank at least d.
  Can be found by solving a generalized eigenvector problem.
- Amazingly, this is exactly the internal representation used by spectral clustering before it normalizes the affinity matrix and begins the actual clustering phase.

General structure of Spectral Embedding Algorithms

- Optional: find neighbours to define a sparse graph.
- Build a (possibly sparse) affinity matrix/tensor between points.
- Munge the affinities/weights/etc. into a quadratic form involving the unknown embedding coordinates.
- Drop the hammer: compute biggest/smallest eigenvectors.
- Read out embedding coordinates from the eigenvectors.

- Many more recent inventions/variations (e.g. Semidefinite Embedding, Hessian LLE)

Picking neighbours is crucial

- **Neighbourhood scale**
  Should be small enough so manifold is roughly flat. but...
  Should be large enough to include a reasonable number of neighbours at your data sampling density.
  In particular, if we ultimately want up to d coordinates, each point must have $> d$ neighbours on average.

- **Nearby Distances**
  If you want to define the neighbours using distance, (e.g. k-NN or ϵ-ball), you need to know how to measure distances between nearby datapoints.

- **Relative positions**
  A single global coordinate system for the original data is not needed. Each datapoint only needs to know the location of its neighbours relative to itself.
  In fact, most of these algorithms can work based on distances alone. (But not the way you might think at first!)

Kernel PCA

- Since you've seen the Snap-Shot method, you know that PCA can be performed either using the sample covariance matrix or the Gram matrix.
- What's that you say? An algorithm that depends only on the Gram matrix? Bring on the Kernel Machine!
- Using Gaussian or other kernels we can do nonlinear "embedding" of some original data, but often we view this as feature extraction rather than dimensionality reduction.
- Plus, there is the preimage problem...
- However, if you insist, you can unify all spectral embedding algorithms as a special kind of kernel PCA by defining whatever you do before you take the eigenvectors as a "data-dependent kernel" and whatever you do afterwards as "computing the embedding from the kernel PCA functions". [Han et. al]

Comments on Spectral Embedding

- The algorithms I discussed provide an embedding of the training data only, but they do not provide a full mapping as did the coordination procedures.
  Recently, Bengio et. al have worked out a way of generalizing these algorithms to new data by viewing them all as special cases of the problem of learning the eigenfunctions of an unknown operator.
- **Manifolds which are not locally flat**
  Some data (e.g. fractals) have different intrinsic dimensionality in different parts of the space.
  Some data has a closed topology (e.g. sphere).

Wrapping Up

- Dimensionality reduction is a slippery business, focussing on the geometric aspect of structure discovery.
- Linear methods are a good baseline, and very important to know about. Often they get you quite far, and have nice linear algebraic closed form solutions.
- Beyond that, you can try to embed each training example into a low-dimensional space so as to preserve some stuff you like.
- If done properly even this can sometimes fall out as a nice spectral problem with an SVD-like solution.
- Or, more ambitiously, you can try to build the "magic box", i.e. learn a full mapping from the high-dimensional space to the low-dimensional space and back again, possibly by coordinating some local models or by generalizing a spectral embedding.
- Tons of stuff we couldn't cover. I owe you a pile of references...

Always leave them wanting more...

There are just two rules for success:

1. Never tell all you know.

- Roger H. Lincoln