ECE 521

Lecture 11 *(not on midterm material)*

13 February 2017

K-means clustering, Dimensionality reduction

With thanks to Ruslan Salakhutdinov for an earlier version of the slides

Overview

- ***K*-means clustering**
- Dimensionality reduction
  - Autoencoders
  - PCA
  - Extensions of PCA

Preface to *K*-means Clustering

- Recall the unknown mixing coefficients in Gaussian mixture models.
- These coefficients are an example of latent variables, which allow complicated distributions to be formed from simpler distributions.
- In general, ‘mixture models’ can be interpreted in terms of having discrete latent variables
- Later, we will also look at continuous latent variables.

K-Means Clustering

• Consider the following problem: Identify clusters, or groups, of data points in a multidimensional space.

• We observe the dataset $\{\mathbf{x_1}, ..., \mathbf{x}_N\}$ consisting of *N* observations, each of *D* dimensions

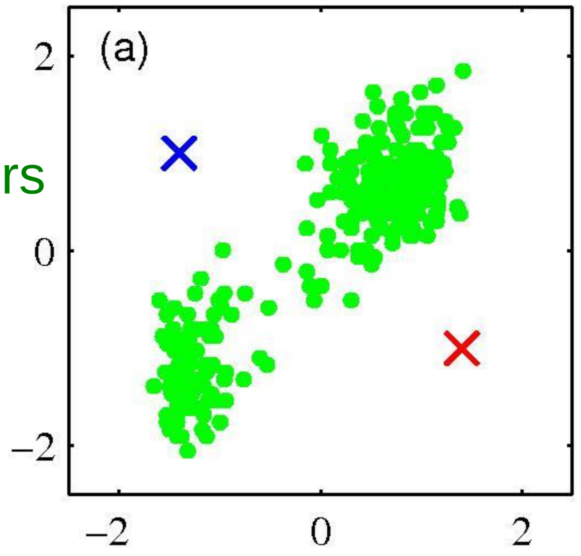
• We would like to partition the data into *K* clusters, where *K* is given

• We next introduce *D*-dimensional vectors, prototypes $\boldsymbol{\mu}_k, k = 1, ..., K.$

• We can think of $\boldsymbol{\mu}_k$ as representing cluster centres

• Our goal:

- Find an assignment of data points to clusters
- Sum of squared distances of each data point to its closest prototype is to be minimized



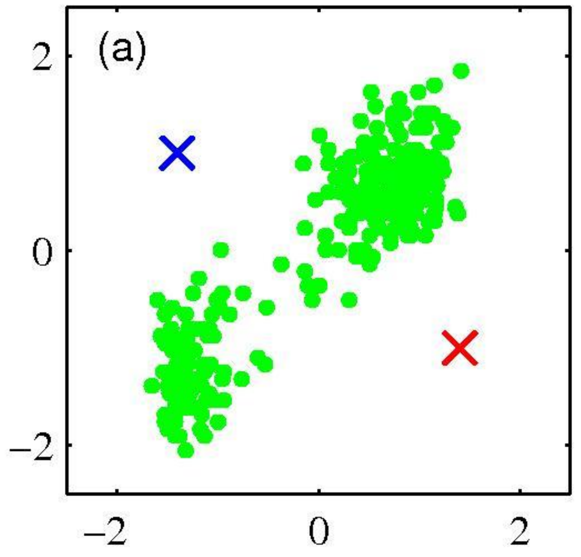
K-Means Clustering

- For each data point $\mathbf{x}_n$ we introduce a binary vector $\mathbf{r}_n$ of length *K* (1-of-*K* encoding), which indicates which of the *K* clusters the data point $\mathbf{x}_n$ is assigned to.
- Define an objective function (distortion measure):

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2.$$

- It represents the sum of squares of the distances of each data point to its assigned prototype $\boldsymbol{\mu}_k$.

- Our goal is to find the values of r_{nk} and the cluster centres $\boldsymbol{\mu}_k$ so as to minimize the objective function *J*.



Iterative Algorithm

- Define an iterative procedure to minimize:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2.$$

- Given $\boldsymbol{\mu}_k$, minimize J with respect to r_{nk} (akin to the **E-step** in EM):

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j ||\mathbf{x}_n - \boldsymbol{\mu}_j||^2 \\ 0 & \text{otherwise} \end{cases} \longleftarrow$$

Hard assignments of points to clusters

which simply says assign the nth data point $\mathbf{x_n}$ to its closest cluster centre

- Given r_{nk}, minimize J with respect to $\boldsymbol{\mu}_k$ (akin to the **M-step**):

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}}. \longleftarrow$$

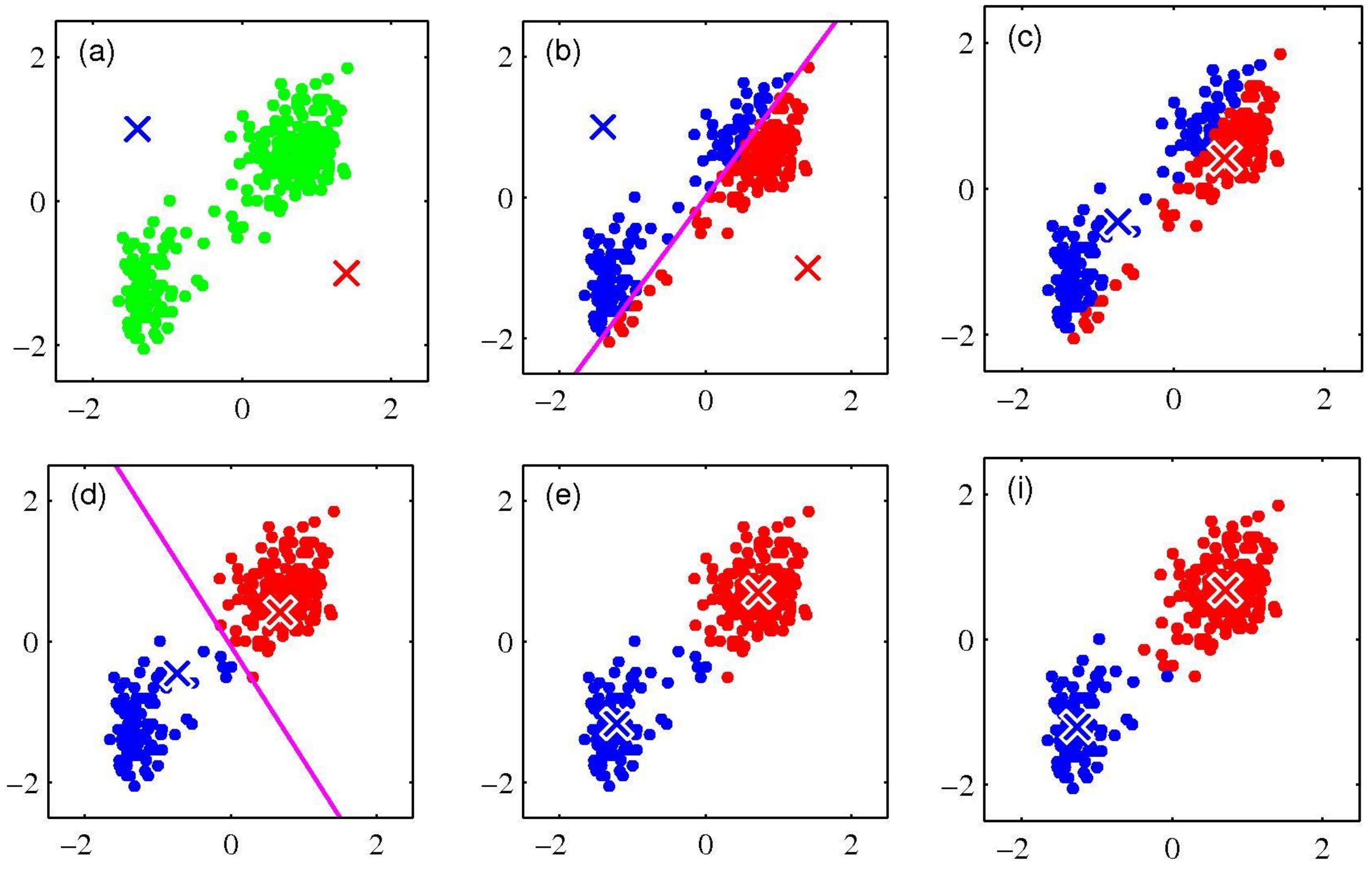
Number of points assigned to cluster *k*.

Set $\boldsymbol{\mu}_k$ equal to the mean of all the data points assigned to cluster *k*

- Guaranteed convergence to a local minimum (not global minimum)

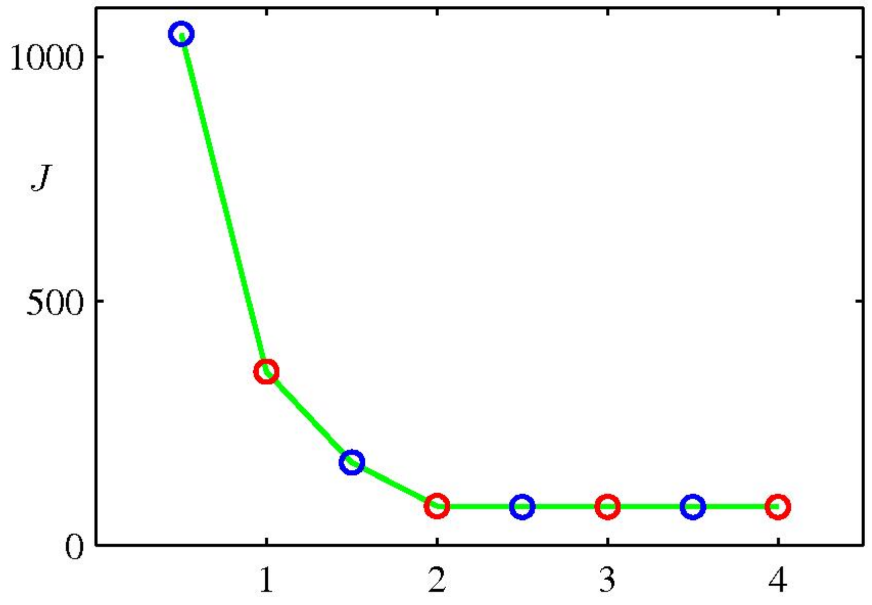
Example

- Example of using *K*-means (*K*=2) on the Old Faithful dataset.



Convergence

• Plot of the cost function after each E-step (blue points) and M-step (red points)



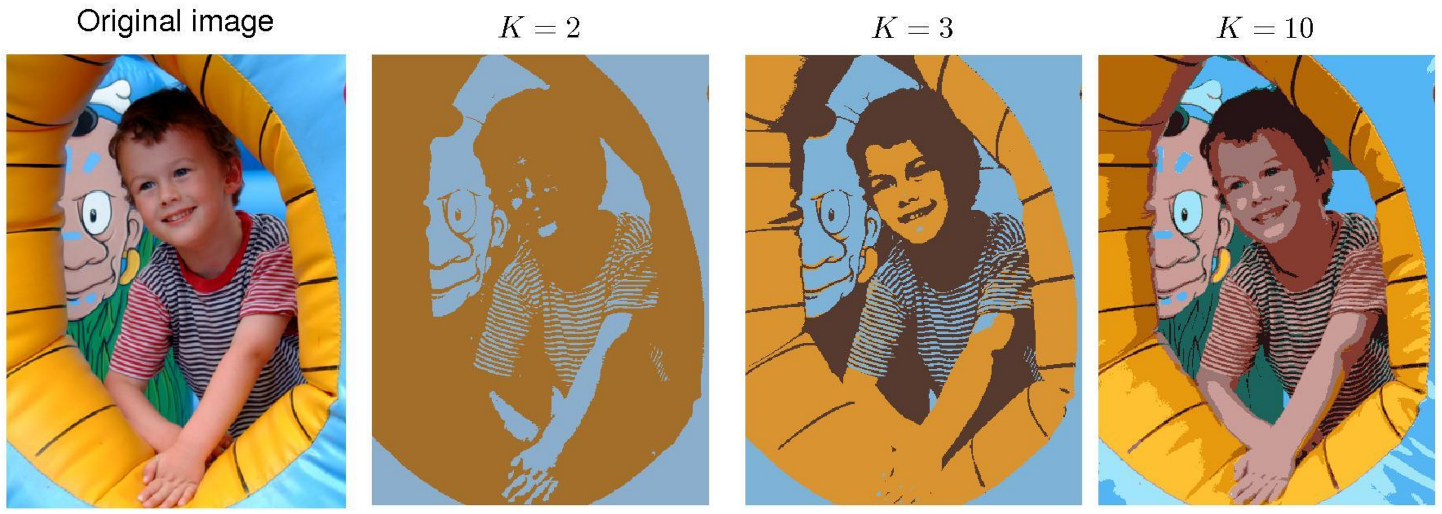
The algorithm has converged after three iterations.

• *K*-means can be generalized by introducing a more general dissimilarity measure:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} K(\mathbf{x}_n, \boldsymbol{\mu}_k).$$

Image Segmentation

- Another application of the *K*-means algorithm.
- Partition an image into regions corresponding, for example, to object parts.
- Each pixel in an image is a point in 3-D space, corresponding to R,G,B channels.

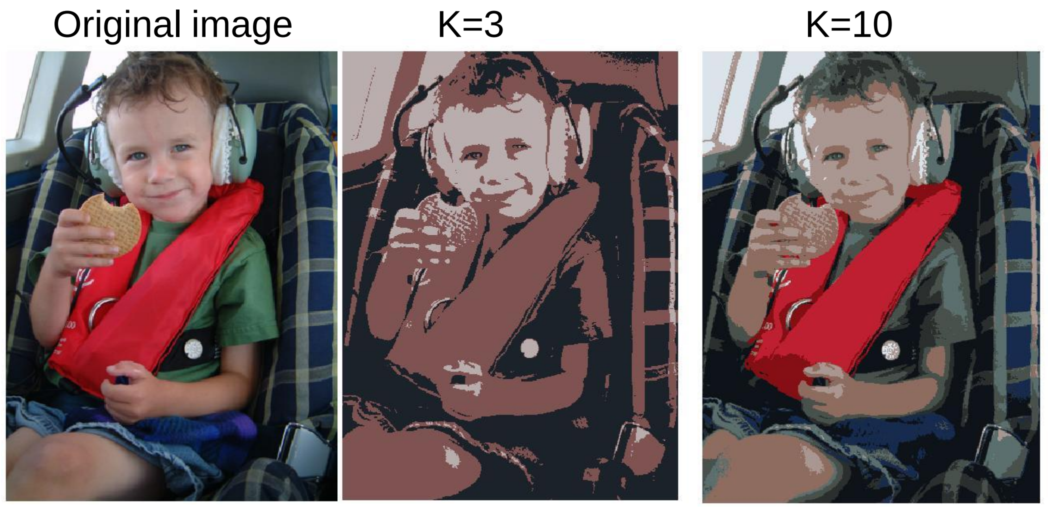


- For a given value of *K*, the algorithm represents an image using *K* colours.

- Another application is image compression.

Image Compression

- For each data point, we store only the identity k of the assigned cluster.
- We also store the values of the cluster centers $\boldsymbol{\mu}_k$.
- Provided $K << N$, we require significantly less data.

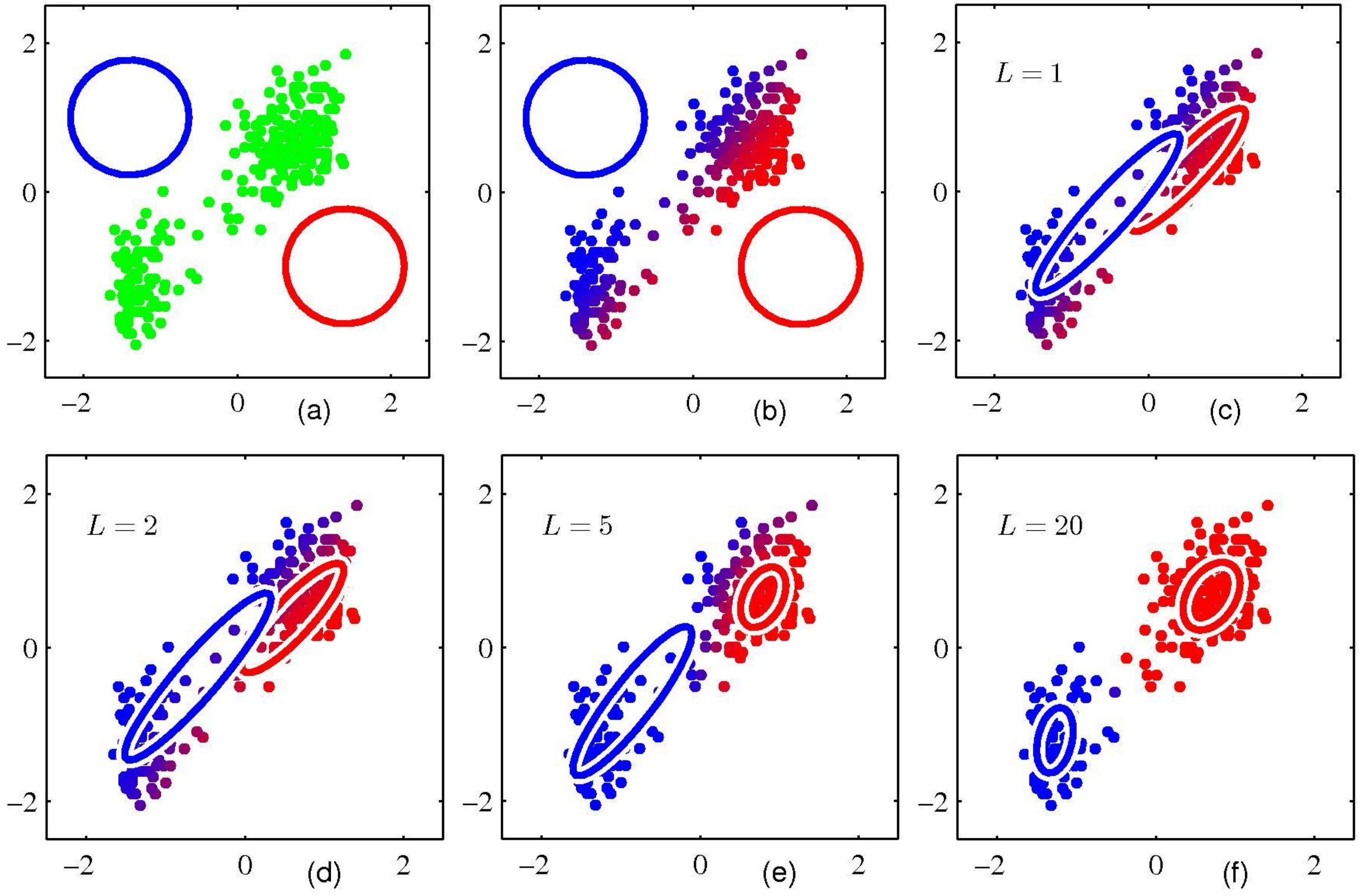


- The original image has 240 x 180 = 43,200 pixels.
- Each pixel contains {R,G,B} values, each of which requires 8 bits.

- Requires 43,200 x 24 = 1,036,800 bits to transmit directly.
- With K-means, we need to transmit K code-book vectors $\boldsymbol{\mu}_k$ -- 24K bits.
- For each pixel we need to transmit $\log_2 K$ bits (as there are K vectors).
- Compressed image requires 43,248 (K=2), 86,472 (K=3), and 173,040 (K=10) bits, which amounts to compression ratios of 4.2%, 8.3%, and 16.7%.

Extension: the real EM Algorithm

- Much slower convergence compared to *K*-means



Overview

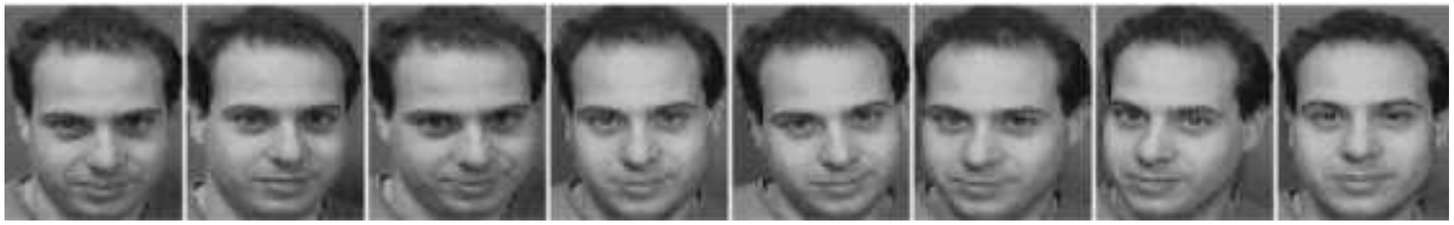
- *K*-means clustering
- **Dimensionality reduction**
  - Autoencoders
  - PCA
  - Extensions of PCA

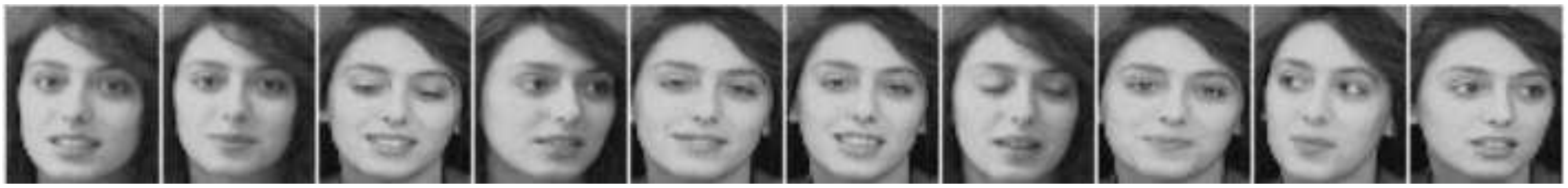
Continuous Latent Variable Models

- Often there are some unknown underlying causes of the data.
- So far we have looked at models with discrete latent variables, such as the mixture of Gaussians.
- Sometimes, it is more appropriate to think in terms of continuous factors which control the data we observe.
- Motivation: for many datasets, data points lie close to a manifold of much lower dimensionality compared to that of the original data space.
- Training continuous latent variable models is often called dimensionality reduction, since there are typically fewer latent dimensions.
- Examples: Principal Component Analysis, Factor Analysis, Independent Component Analysis

Intrinsic Latent Dimensions

- What are the intrinsic latent dimensions in these two datasets?

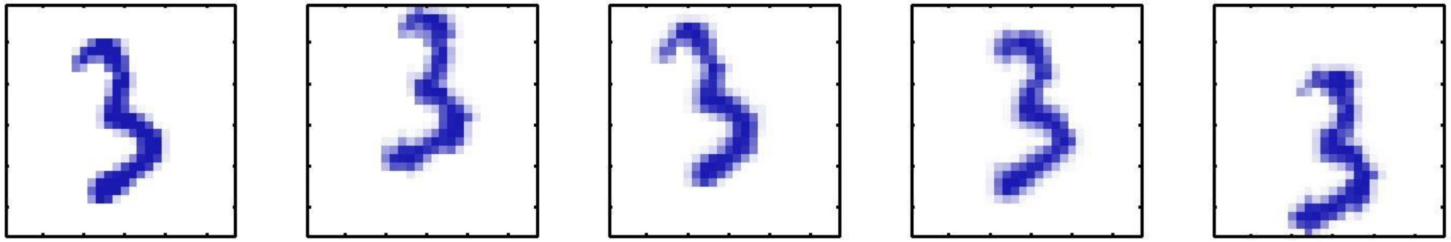




- How can we find the latent dimensions from this high-dimensional data?

Intrinsic Latent Dimensions

• In this dataset, there are only 3 degrees of freedom of variability, corresponding to vertical- and horizontal translations, and the rotations.



• Each image undergoes a random displacement and rotation within some larger image field.

• The resulting images have 100 x 100 = 10,000 pixels.

Autoencoders

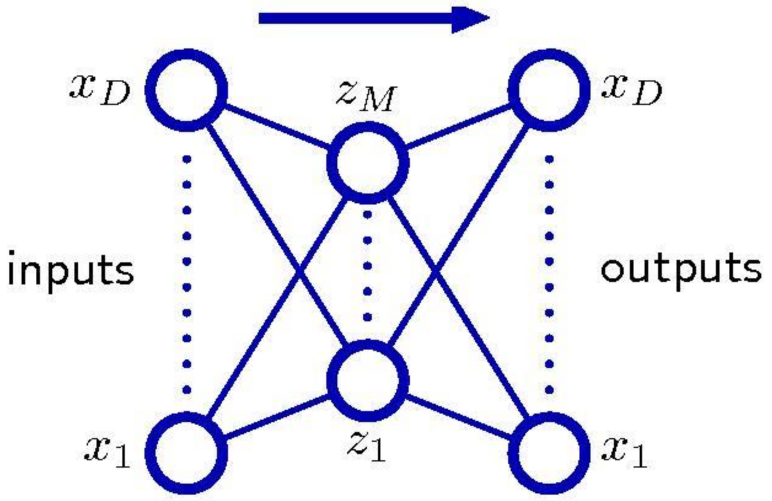
• Neural networks can be used for nonlinear dimensionality reduction.

• This is achieved by having the same number of outputs as inputs. These models are called autoencoders, or autoassociative networks

• Consider a multilayer perceptron that has *D* inputs, *D* outputs, and *M* hidden units such that *M* < *D*.

• It is useful if we can squeeze the information through some kind of bottleneck.

- If we use a linear network this is very similar to Principal Component Analysis.



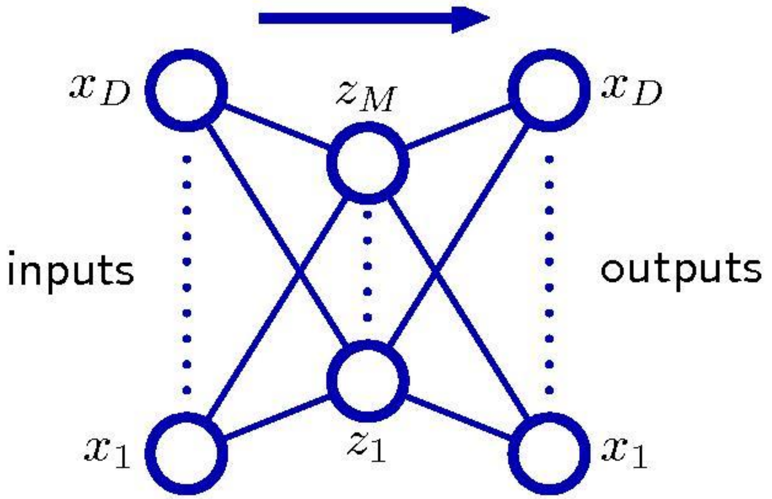
Autoencoders and PCA

- Given an input **x**, its corresponding reconstruction is given by:

$$y_k(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{M} w_{kj}^{(2)} \sigma \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i \right), \quad k = 1, .., D.$$

- We can determine the network parameters **w** by minimizing the reconstruction error:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} ||y(\mathbf{x}_n, \mathbf{w}) - \mathbf{x}_n||^2.$$



- If the hidden and output layers are linear, we will learn hidden units that are a linear function of the data and minimize the squared error.

- The *M* hidden units will span the same space as the first *m* principal components. The weight vectors may not be orthogonal.

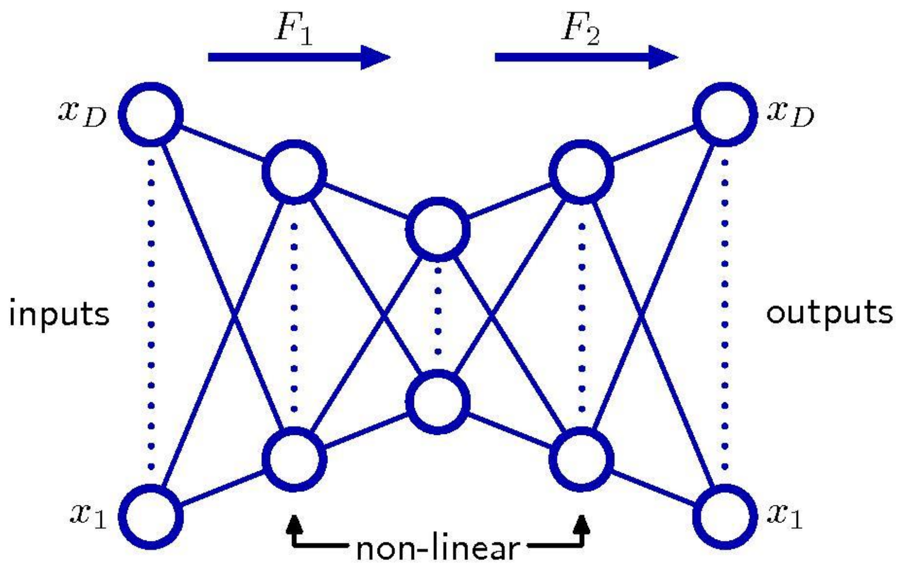
Deep Autoencoders

- We can put extra nonlinear hidden layers between the input and the bottleneck and between the bottleneck and the output.

- This gives a nonlinear generalization of PCA.

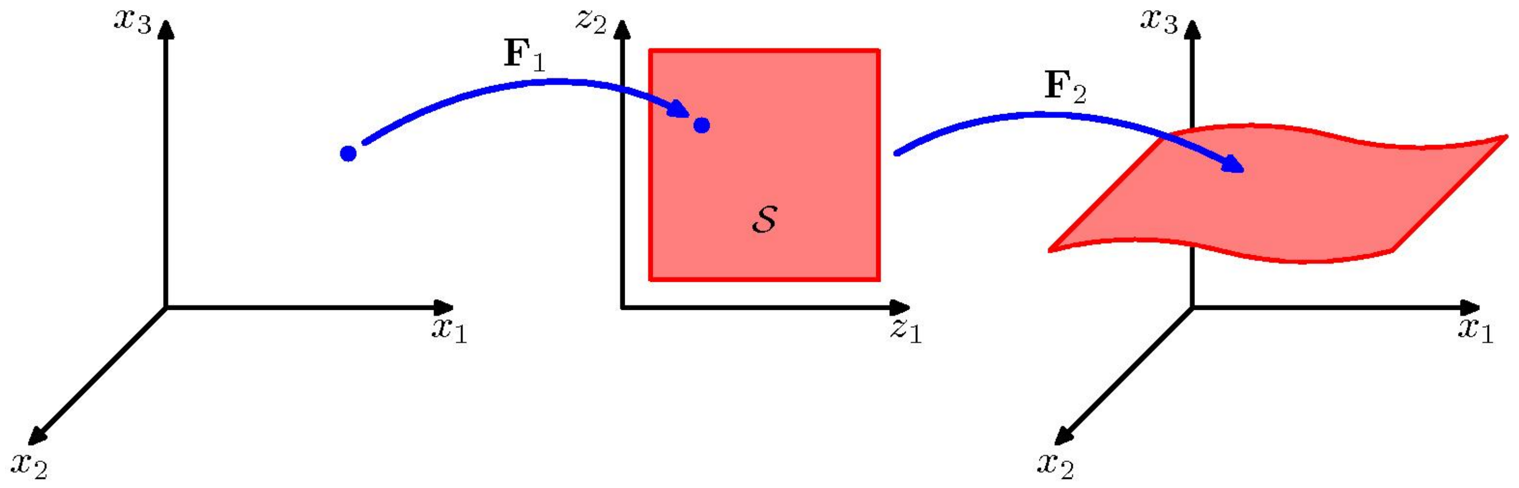
- The network can be trained by the minimization of the reconstruction error function.

- Much harder to train.



Geometrical Interpretation

- Geometrical interpretation of the mappings performed by the network with 2 hidden layers, for the case of *D*=3 and *M*=2 units in the middle layer



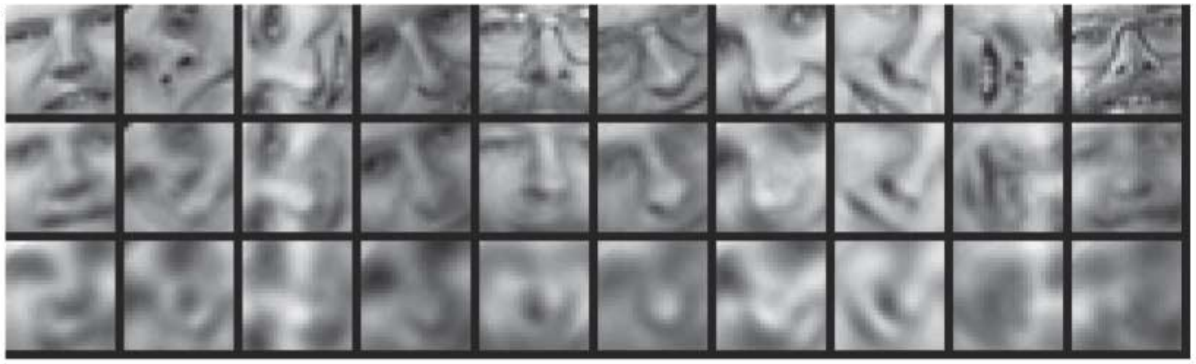
- The mapping $\mathbf{F_1}$ defines a nonlinear projection of points in the original *D*-space into the *M*-dimensional subspace

- The mapping $\mathbf{F_2}$ maps from an *M*-dimensional space into *D*-dimensional space

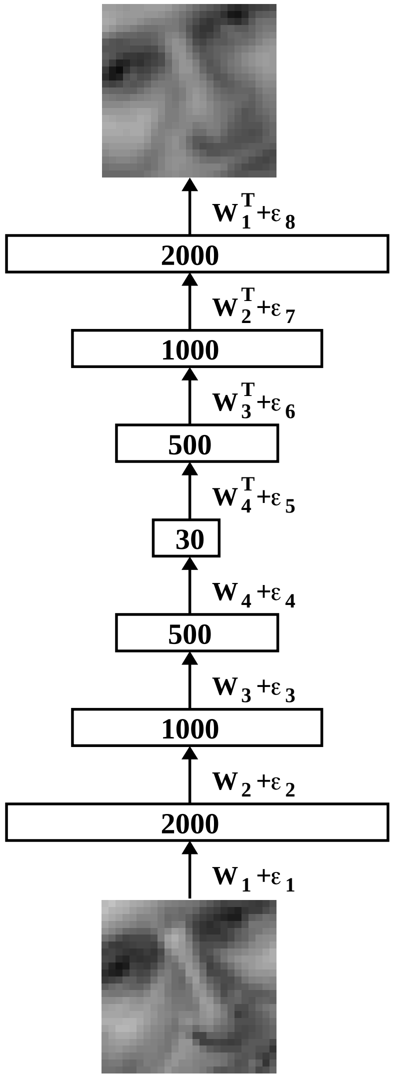
Deep Autoencoders

• We can consider very deep autoencoders.

• There is an efficient way to learn these deep autoencoders

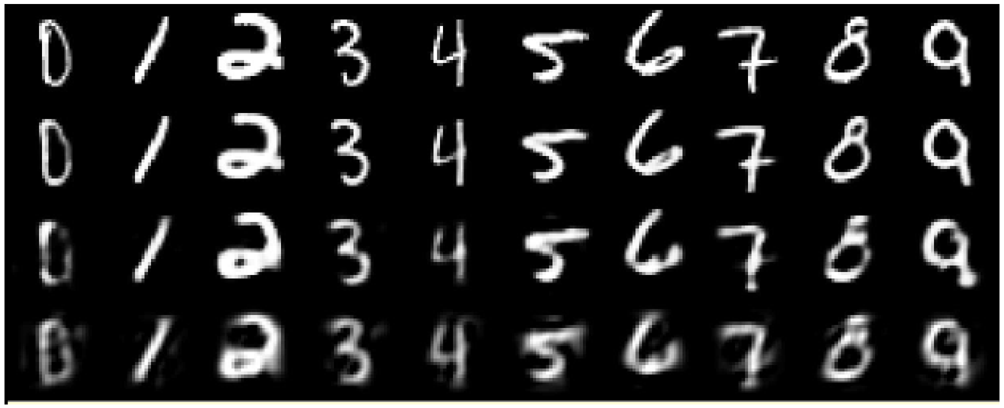


• By row: Real data, Deep autoencoder with a bottleneck of 30 linear units, and 30-d PCA.



Deep Autoencoders

- We can consider very deep autoencoders.
- Similar model for MNIST handwritten digits:



Real data

30-d deep autoencoder

30-d logistic PCA

30-d PCA

- The Deep autoencoder produces much better reconstructions.

Class Structure of the Data

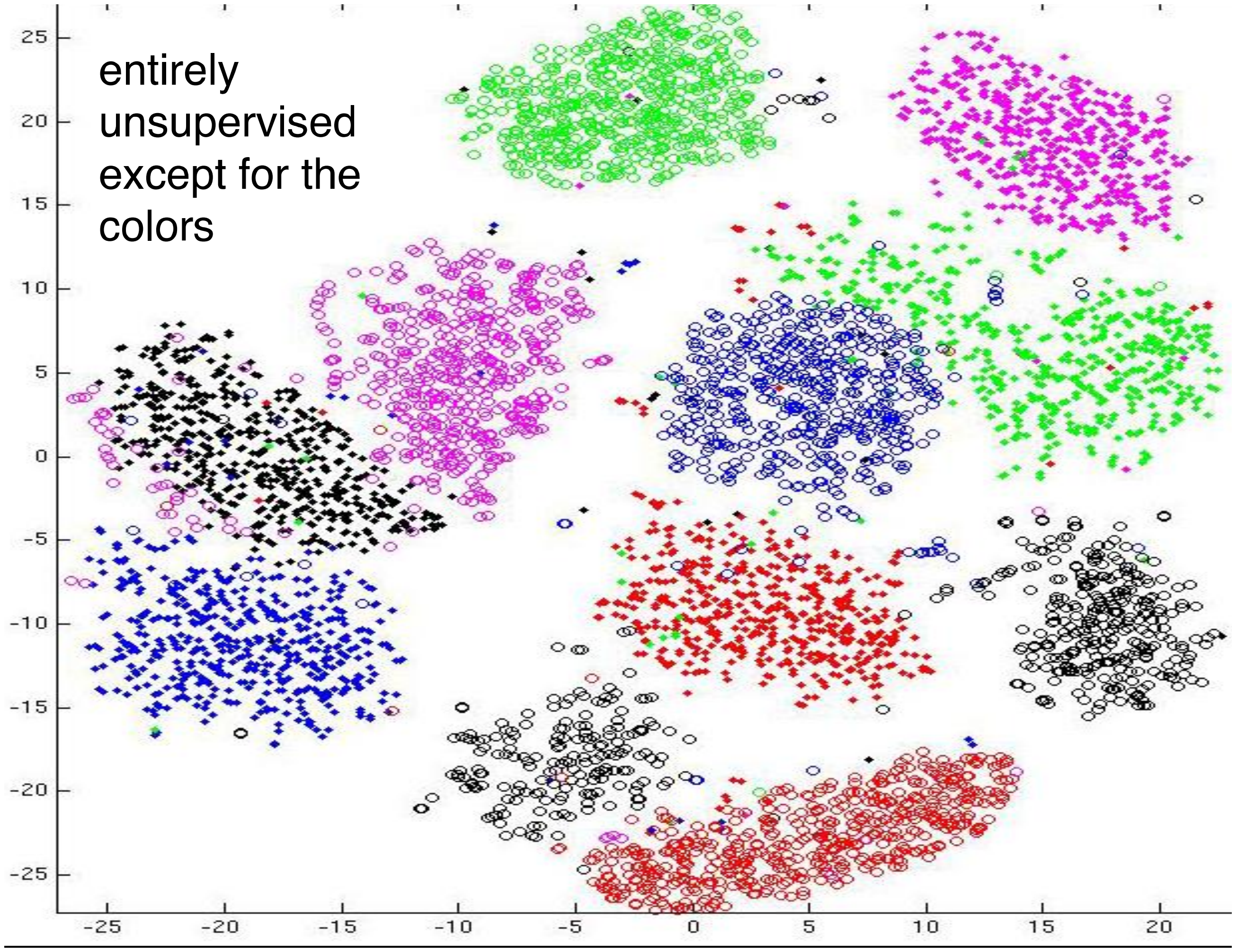
- Do the 30-D codes found by the deep autoencoder preserve the class structure of the data?

- Take the 30-D activity patterns in the code layer and display them in 2-D using a form of non-linear multi-dimensional scaling

- Will the learning find the natural classes?

Class Structure of the Data

• Do the 30-D codes found by the deep autoencoder preserve the class structure of the data?

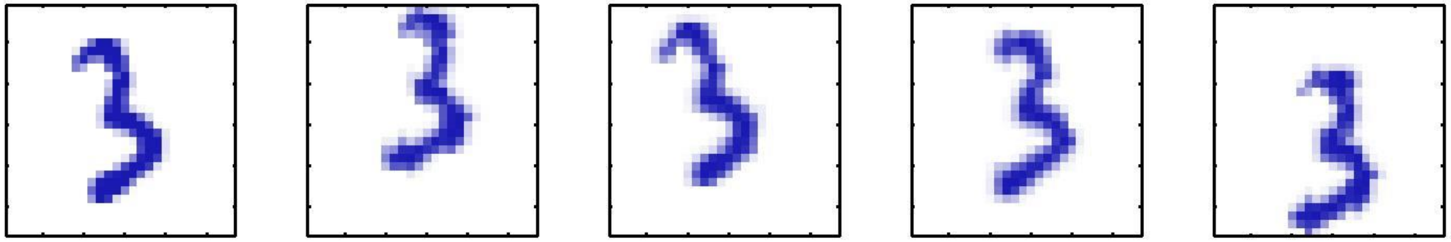


Overview

- *K*-means clustering
- Dimensionality reduction
    - Autoencoders
    - **PCA**
    - Extensions of PCA

Recall: Intrinsic Latent Dimensions

• In this dataset, there are only 3 degrees of freedom of variability, corresponding to vertical- and horizontal translations, and the rotations.



• Each image undergoes a random displacement and rotation within some larger image field.

• The resulting images have 100 x 100 = 10,000 pixels.

Generative View

- Each data example was generated by first selecting a point from a distribution in the latent space, then generating a point from the conditional distribution in the input space

- Simplest latent variable models: Assume a Gaussian distribution for both the latent and observed variables.

- This leads to a probabilistic formulation of Principal Component Analysis and Factor Analysis.

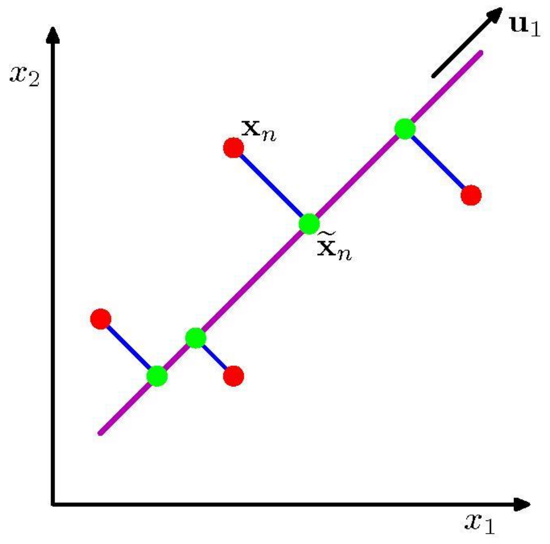
- We will look at standard PCA, and note its extensions

- Advantages of probabilistic formulations: use of EM for parameter estimation, mixture of PCAs, Bayesian PCA.

Principal Component Analysis

- Used for data compression, visualization, feature extraction, dimensionality reduction.

- The goal is find *M* principal components underlying the *D*-dimensional data
  - select the top *M* eigenvectors of **S** (data covariance matrix): $\{\mathbf{u}_1, ..., \mathbf{u}_M\}$.
  - project each input vector $\mathbf{x}_n$ into this subspace, e.g. $z_{n1} = \mathbf{x}_n^T \mathbf{u}_1$.



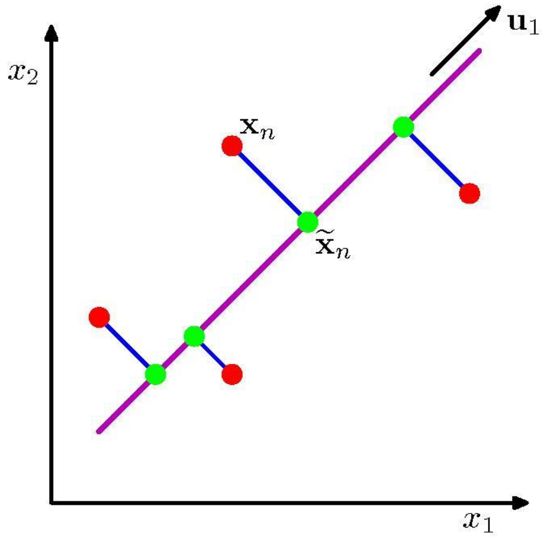
- Full projection into *M* dimensions takes the form:

$$\begin{bmatrix} \mathbf{u}_1^\top \\ \cdots \\ \mathbf{u}_M^\top \end{bmatrix} [\mathbf{x}_1 \cdots \mathbf{x}_N] = [\mathbf{z}_1 \cdots \mathbf{z}_N]$$

- Two views/derivations:
  - Maximize variance (scatter of green points).
  - Minimize error (red-green distance per data point).

Maximum Variance Formulation

- Consider a dataset $\{x_1,\ldots,x_N\}$, $x_n \in R^D$. Our goal is to project data onto a space having dimensionality $M < D$.
- Consider the projection into $M=1$ dimensional space.
- Define the direction of this space using a D-dimensional unit vector $\mathbf{u_1}$, so that $\mathbf{u}_1^T\mathbf{u}_1 = 1.$
- Objective: maximize the variance of the projected data with respect to $\mathbf{u_1}$.



$$\frac{1}{N}\sum_{n=1}^{N}\{\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\overline{\mathbf{x}}\}^2 = \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1$$

where sample mean and data covariance is given by:

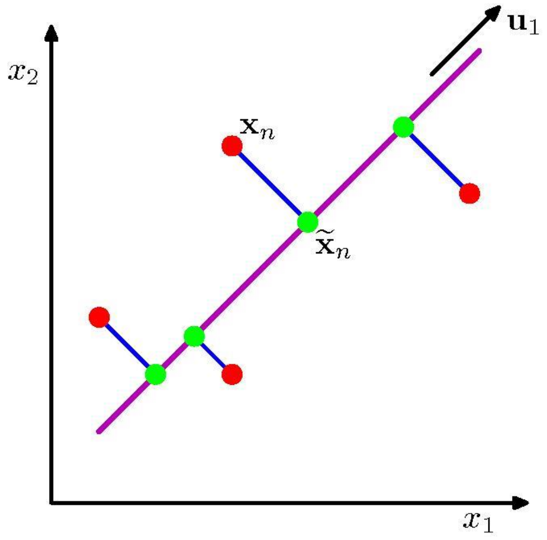
$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$$

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^T$$

Maximum Variance Formulation

• Maximize the variance of the projected data:

$$\frac{1}{N}\sum_{n=1}^{N}\{\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\overline{\mathbf{x}}\}^2 = \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1$$



• Must constrain ||**u₁**|| = 1. Using a Langrage multiplier, maximize:

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 + \lambda(1 - \mathbf{u}_1^T\mathbf{u}_1)$$

• Setting the derivative with respect to $\mathbf{u}_1$ to zero:

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1.$$

• Hence $\mathbf{u}_1$ must be an eigenvector of **S**.

• The maximum variance of the projected data is given by:

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 = \lambda_1.$$

• Optimal $\mathbf{u}_1$ is the principal component (eigenvector with maximal eigenvalue).

Minimum Error Formulation

- Introduce a complete orthonormal set of *D*-dimensional basis vectors: $\{\mathbf{u}_1, ..., \mathbf{u}_D\}$:

$$\mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}.$$

- Without loss of generality, we can write:

$$\mathbf{x}_n = \sum_{i=1}^{D} \alpha_{ni} \mathbf{u}_i, \quad \alpha_{ni} = \mathbf{x}_n^T \mathbf{u}_i.$$

Rotation of the coordinate system to a new system defined by $\mathbf{u}_i$.

- Our goal is to represent data points by the projection into an *M*-dimensional subspace, plus some distortion
- Represent the *M*-dim linear subspace by the first *M* basis vectors

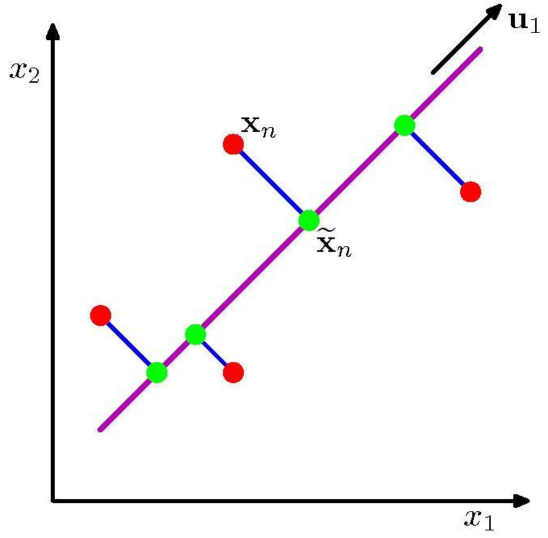
$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} z_{ni} \mathbf{u}_i + \sum_{i=M+1}^{D} b_i \mathbf{u}_i.$$

Minimum Error Formulation

- Represent *M*-dim linear subspace by the first *M* basis vectors:

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} z_{ni}\mathbf{u}_i + \sum_{i=M+1}^{D} b_i\mathbf{u}_i.$$

where ***z***



- Objective: minimize distortion with respect to $\mathbf{u}_i$, z_{ni}, and b_i.

$$J = \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{x}_n - \tilde{\mathbf{x}}_n||^2.$$

- Minimizing with respect to z_{nj}, b_j:

$$z_{nj} = x_n^T u_j$$
$$b_j = \bar{x}^T u_j$$

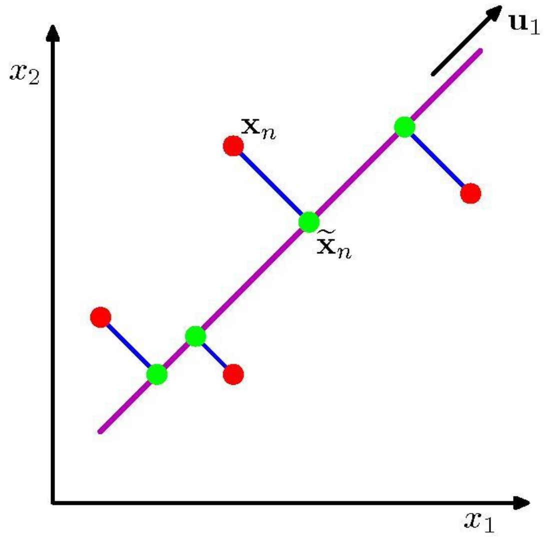
- Hence, the objective reduces to:

$$J = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}(\mathbf{x}_n^T\mathbf{u}_i - \bar{\mathbf{x}}^T\mathbf{u}_i)^2 = \sum_{i=M+1}^{D}\mathbf{u}_i^T\mathbf{S}\mathbf{u}_i.$$

Minimum Error Formulation

• Minimize distortion with respect to **u**$_i$: constraint minimization problem:

$$J = \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{x}_n - \tilde{\mathbf{x}}_n||^2 = \sum_{i=M+1}^{D} \mathbf{u}_i^T \mathbf{S} \mathbf{u}_i.$$



• The general solution is obtained by choosing **u**$_i$ to be eigenvectors of the covariance matrix:

$$\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i.$$

• The distortion is then given by: $J = \sum_{i=M+1}^{D} \lambda_i.$

• The objective is minimized when the remaining *D* – *M* components are the eigenvectors of **S** with *lowest eigenvalues* → same result.

• We will later see a generalization: deep autoencoders.

Applications of PCA

• Run PCA on 2429 19x19 grayscale images (CBCL database)

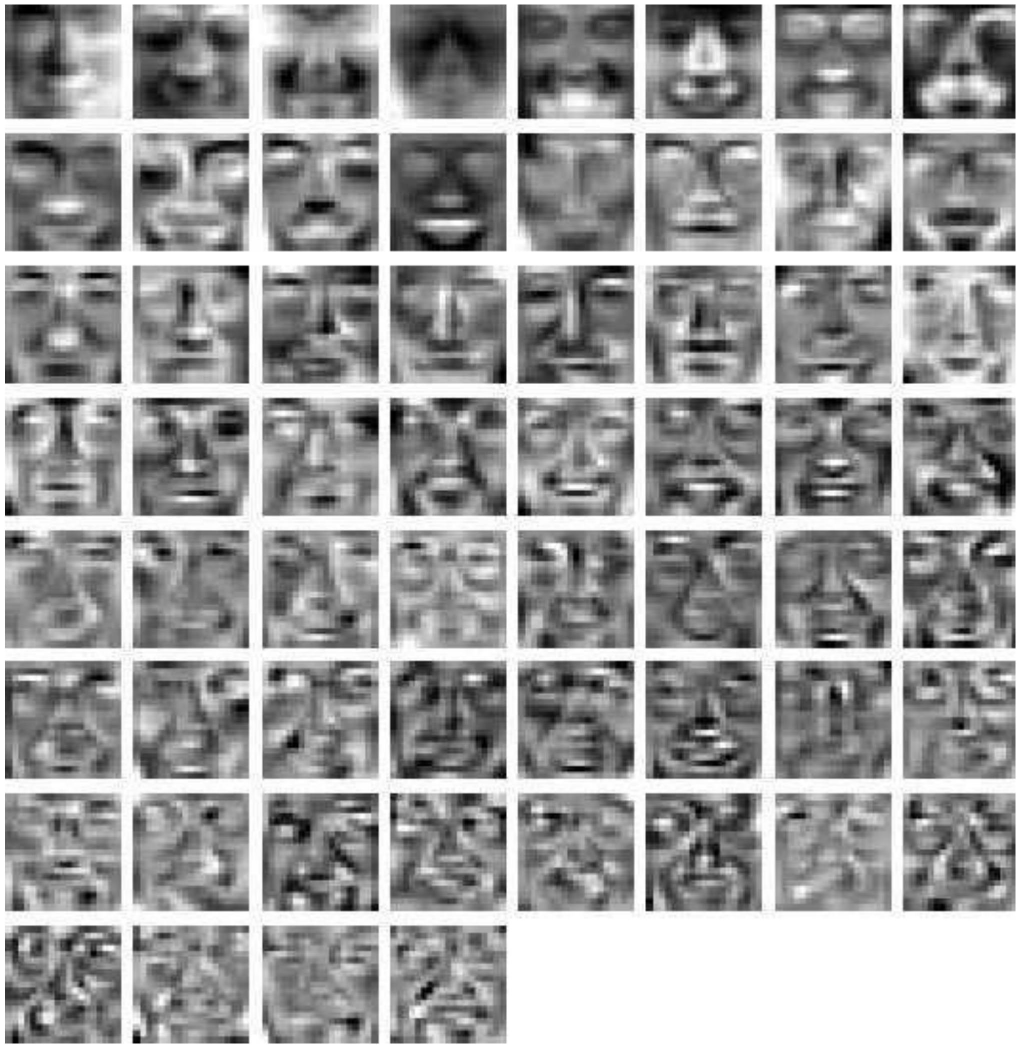
• Data compression: We can get good reconstructions with only 3 components.

• Pre-processing: We can apply a standard classifier to latent representation – PCA with 3 components obtains 79% accuracy on face/non-face discrimination in test data, vs. 77% for a mixture of Gaussians with 84 components.

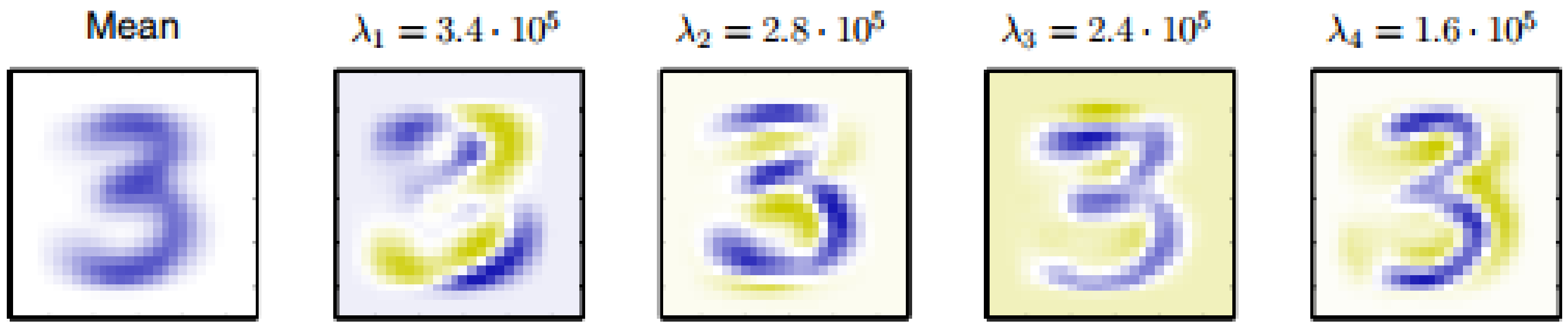
• Data visualization: by projecting the data onto the first two principal components.

Learned Basis

- Run PCA on 2429 19x19 grayscale images (CBCL database)



Eigenvectors for 3's

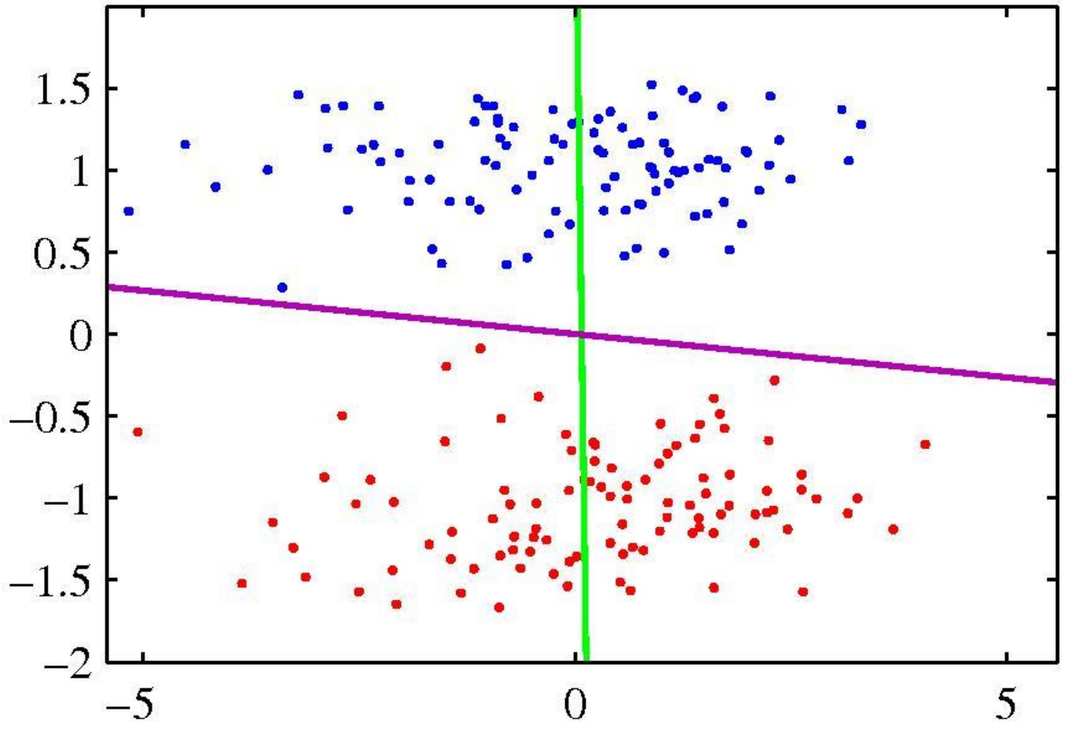


Mean
$\lambda_1 = 3.4 \cdot 10^5$
$\lambda_2 = 2.8 \cdot 10^5$
$\lambda_3 = 2.4 \cdot 10^5$
$\lambda_4 = 1.6 \cdot 10^5$

The mean vector $\overline{\mathbf{x}}$ along with the first four PCA eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_4$ for the off-line digits data set, together with the corresponding eigenvalues.

Brief intro to Fisher's LDA

• Fisher's LDA, like PCA, can be used for dimensionality reduction.



• PCA chooses the direction of maximum variance (magenta curve) leading to strong class overlap (unsupervised).

• LDA takes into account the class labels (supervised), leading to a projection into the green curve.

PCA for High-Dimensional Data

- In some applications of PCA, the number of data points is smaller than the dimensionality of the data space, i.e. $N<D$.
- So far, we have needed to find the eigenvectors of the D x D data covariance matrix **S**, which scales as $\mathcal{O}(D^3)$.
- Direct application of PCA will often be computationally infeasible.
- Solution: Let **X** be the N x D centred data matrix. The corresponding eigenvector equation becomes:

$$\frac{1}{N}\mathbf{X}^T\mathbf{X}\mathbf{u}_i = \lambda_i \mathbf{u}_i.$$

- Pre-multiply by **X**:

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{u}_i) = \lambda_i(\mathbf{X}\mathbf{u}_i).$$

PCA for High-Dimensional Data

- Define $\mathbf{v}_i = \mathbf{X}\mathbf{u}_i$, and hence we have:

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T\mathbf{v}_i = \lambda_i \mathbf{v}_i.$$

- This is an eigenvector equation for the N x N matrix
- It has the same N – 1 eigenvalues as the original data covariance matrix S (which itself has an additional D – N + 1 zero eigenvalues).
- Computational cost scales as $\mathcal{O}(N^3)$ rather than $\mathcal{O}(D^3)$.
- To determine eigenvectors, we multiply by $\mathbf{X}^T$:

$$\left(\frac{1}{N}\mathbf{X}^T\mathbf{X}\right)(\mathbf{X}^T\mathbf{v}_i) = \lambda_i \mathbf{X}^T\mathbf{v}_i.$$

- Hence $\mathbf{X}^T$ $\mathbf{v}_i$ is an eigenvector of **S** with eigenvalue λ_i.
- These eigenvectors may not be normalized.

Probabilistic PCA

- We briefly mention a 1990s probabilistic extension of PCA
- Advantages of probabilistic PCA (PPCA):
  - We can derive an EM algorithm for PCA, which is computationally efficient.
  - PPCA allows us to deal with missing values in the data set.
  - We can formulate a mixture of PPCAs in a principled way.
  - PPCA forms the basis for Bayesian PCA, in which the dimensionality of the principal subspace can be determined from the data.
  - The existence of a likelihood function allows direct comparisons with other probabilistic density models
  - And more

First five weeks

1. Intro to ML: types of learning, evaluating models, probability theory, loss functions
2. kNN, optimization, MLE, regularization
3. Linear basis function models, decision theory, classification
4. Logistic regression, neural networks
5. Neural networks and deep learning