Unsupervised Learning

with Latent Variable Models

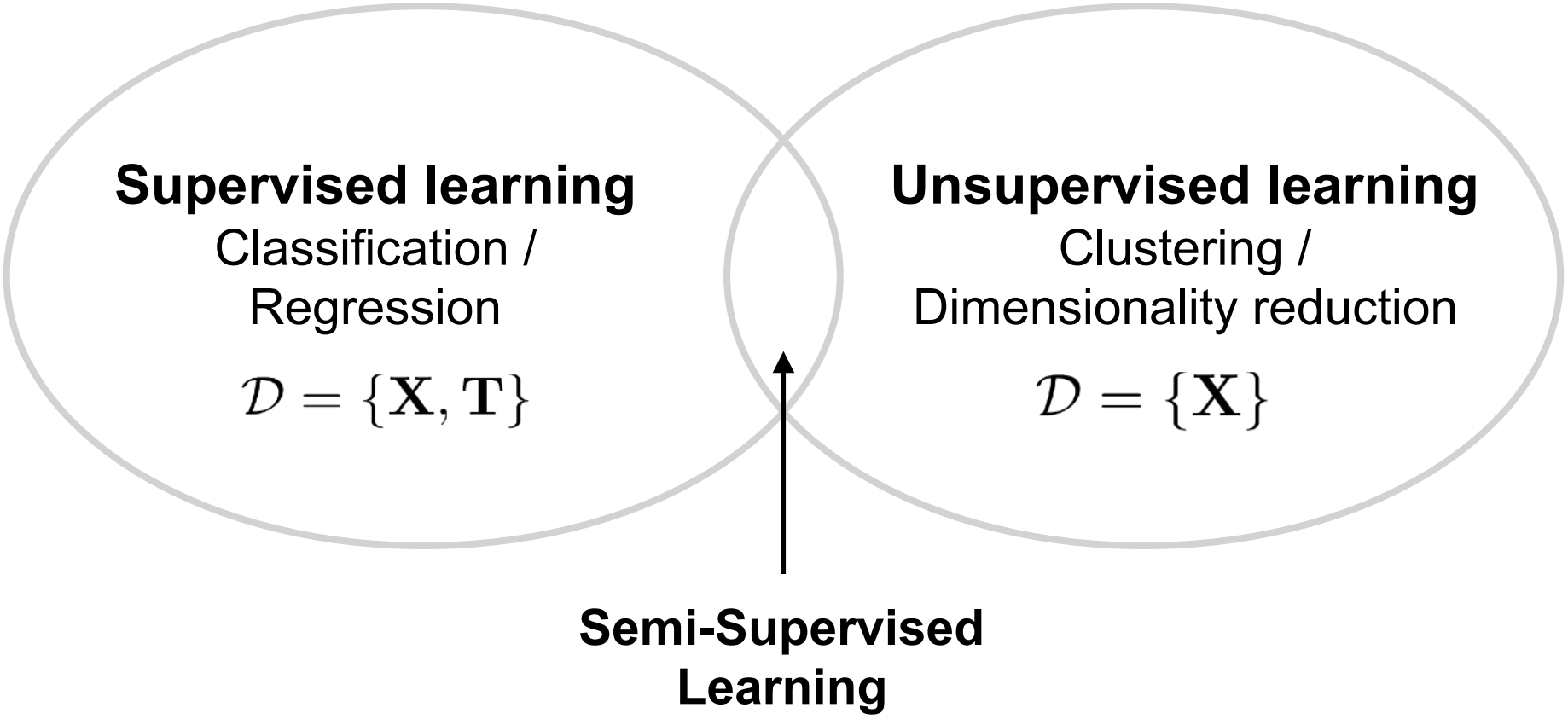
Thomas Kipf

10 October 2016

Machine Learning I

UNIVERSITEIT VAN AMSTERDAM

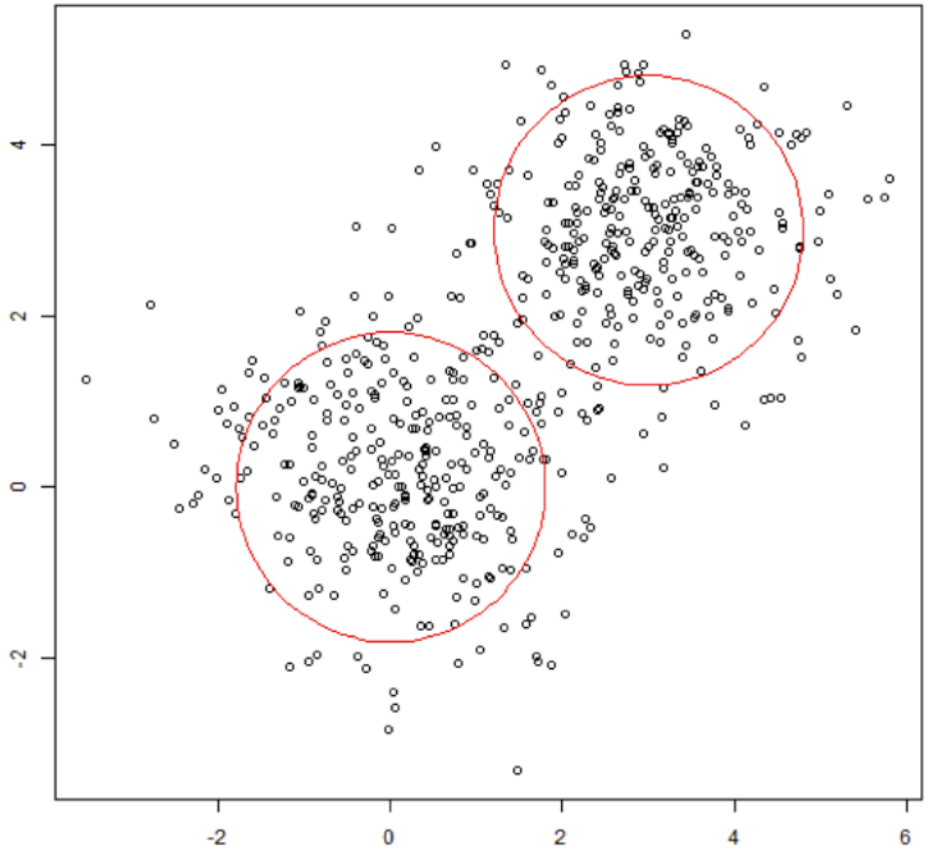
Machine Learning in a nutshell



(Reinforcement learning)

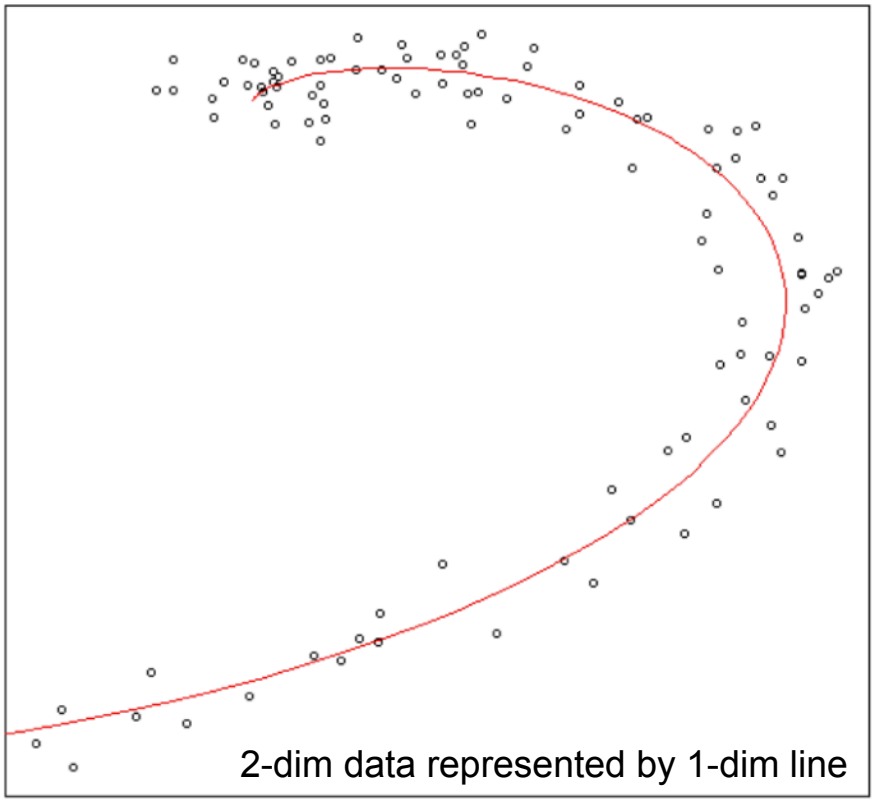
Recap: Unsupervised Learning

Clustering



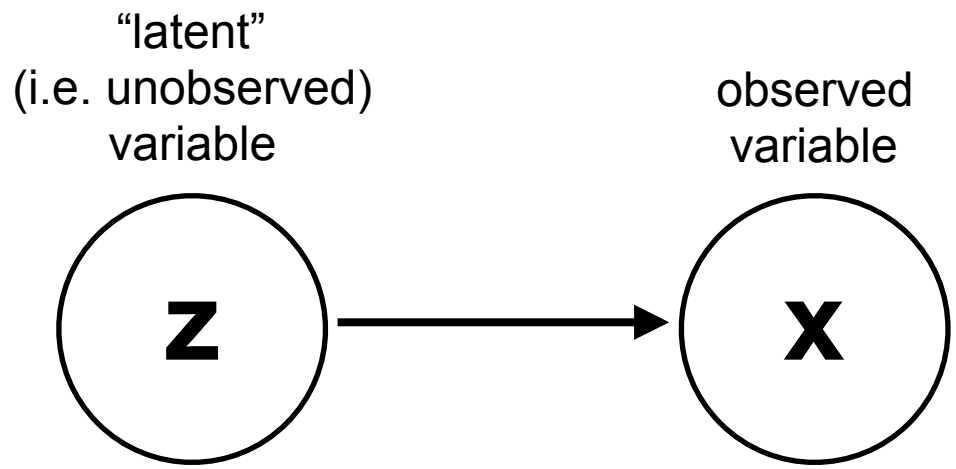
Discrete latent variables

Dimensionality reduction



Continuous latent variables

Latent Variable Models



We say that "**z** generates **x**", i.e. $p(\mathbf{x}|\mathbf{z})$

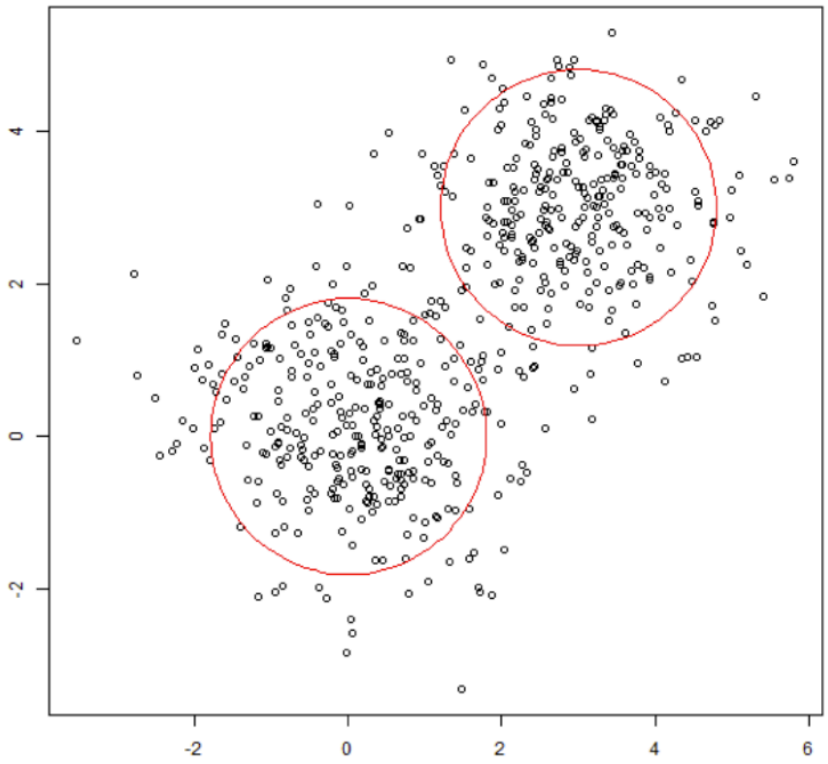
This allows us to express fairly complicated (marginal) distributions $p(\mathbf{x})$
in terms of more tractable joint distributions $p(\mathbf{x}, \mathbf{z})$

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} = \int p(\mathbf{x}, \mathbf{z})d\mathbf{z}$$

z can be either *continuous* or *discrete*

for discrete **z**: replace with summation

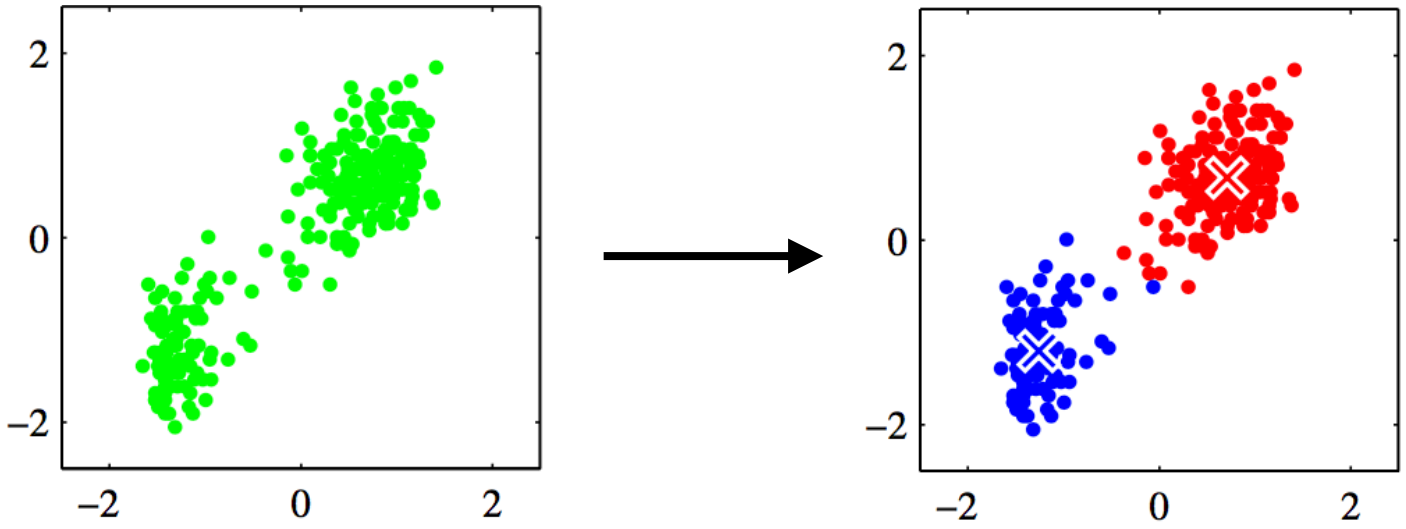
Discrete Latent Variable Models



see **Chapter 9.1** in C.M. Bishop, Pattern Recognition and Machine Learning (2007)

Example: Clustering with K-Means

K-Means: simple non-probabilistic clustering algorithm



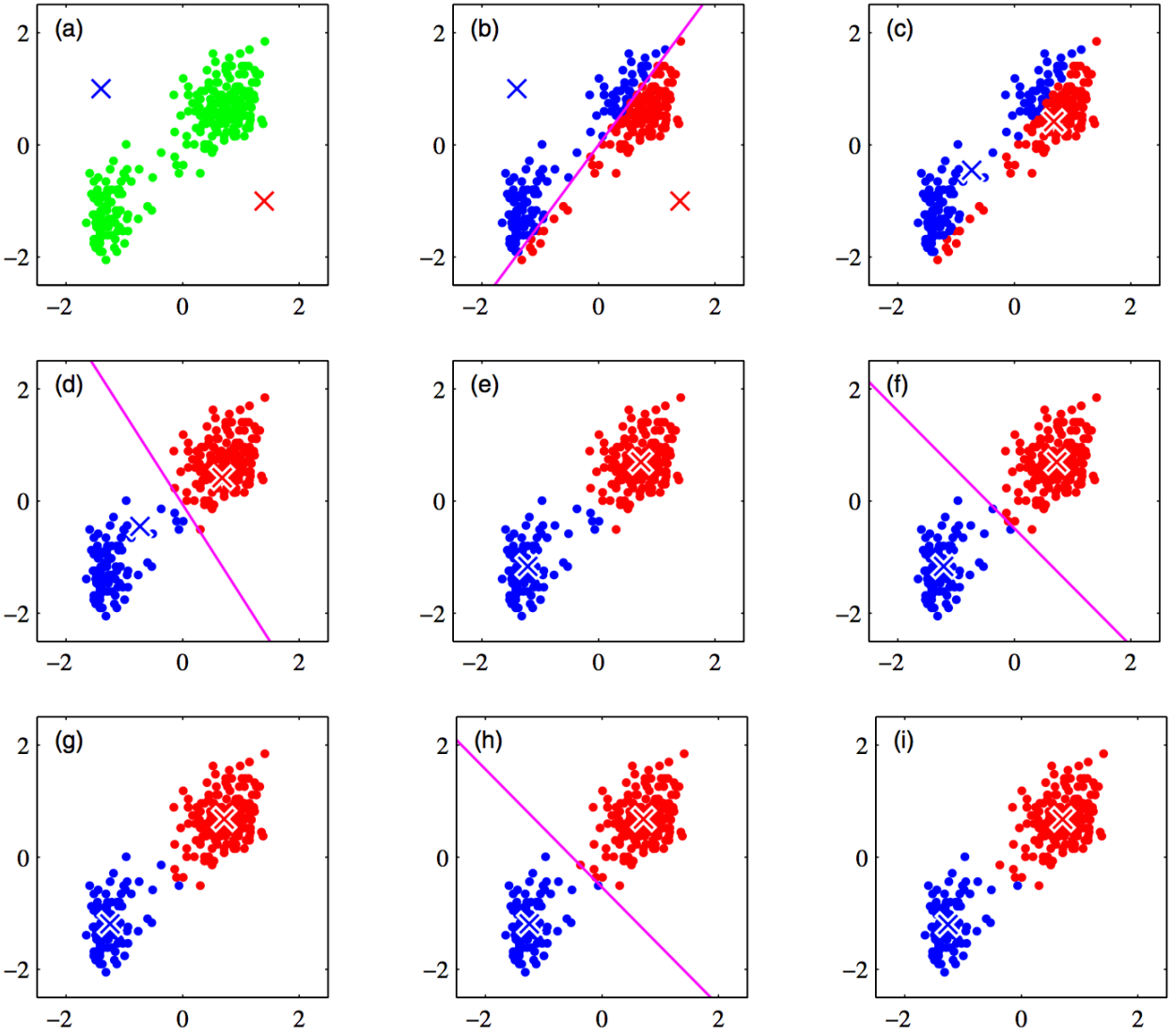
Dataset

Final clustering

Every single data point is modeled by a discrete (latent) variable

(here: the identity/color of the cluster)

K-Means in pictures



K-Means algorithm

Dataset $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with $\mathbf{x}_n \in \mathbb{R}^D$ — **Goal**: Partition data into K clusters

Formally: Find K vectors $\boldsymbol{\mu}_k \in \mathbb{R}^D$ and assignment indicators $r_{nk} \in \{0, 1\}$ so that the following error function is minimized:

"latent variable" (**z**)

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \quad \text{with} \quad \frac{\partial J}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k) = 0$$

1. Initialize $\boldsymbol{\mu}_k \in \mathbb{R}^D$

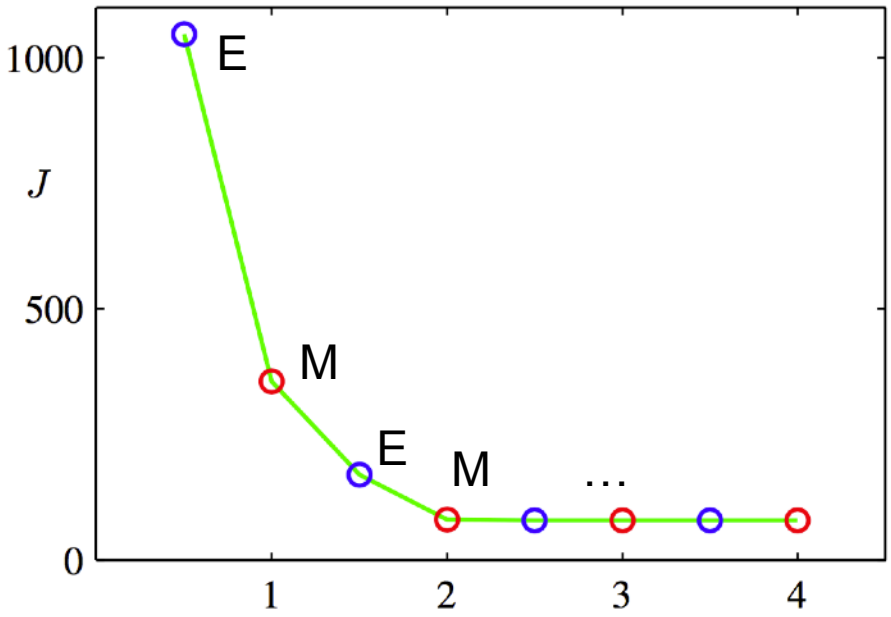
2. Repeat (fix one variable while optimizing with respect to the other):

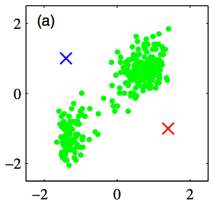
"E"-Step: $r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$ *Re-assign clusters*

"M"-Step: $\boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}}$ *Re-compute centers*

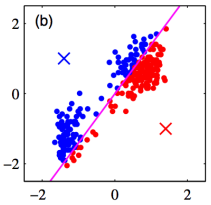
… until convergence

K-Means: Convergence

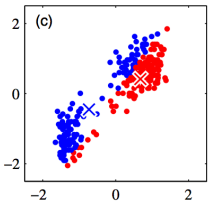




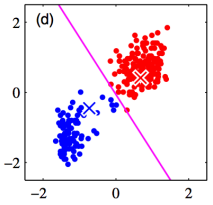
Initialize



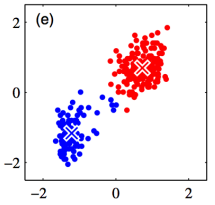
E-step



M-step



E-step



M-step

…

Application: K-Means image compression

K-Means: Discussion

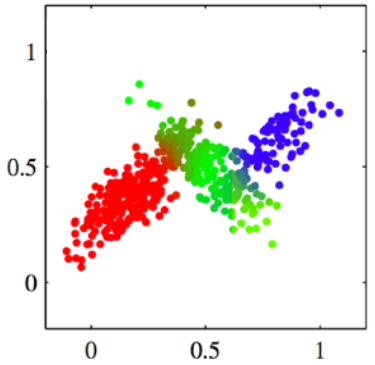
Generalization: $\widetilde{J} = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \mathcal{V}(\mathbf{x}_n, \boldsymbol{\mu}_k)$ with dissimilarity measure $\mathcal{V}(\mathbf{x}, \mathbf{x}')$

Things to keep in mind:

- Number of clusters has to be chosen in advance
- Convergence guaranteed (can be local minimum -> restarts required!)
- Cluster assignments are “hard” (no “half-way” assignments)
- $\mathcal{O}(NK)$ complexity. Online SGD-like variant exists (see Bishop)
- Sensitive to scale of features

Beyond K-Means - probabilistic view (next lecture):

- (Gaussian) Mixture Models (“soft” cluster assignment)
- Optimal number of clusters can be inferred from data
- General Expectation-Maximization (EM) algorithm



Issues with discrete latent variables

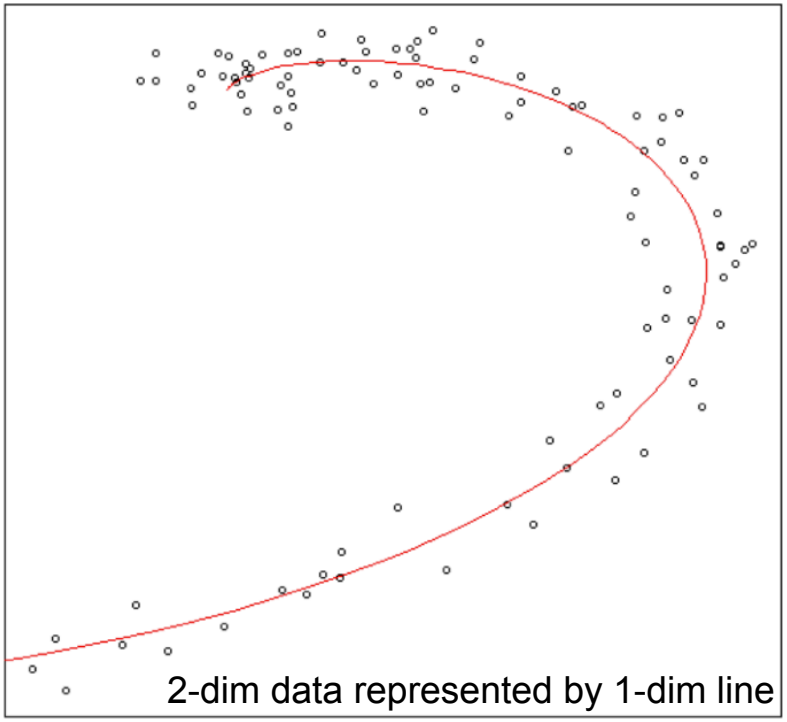
Discrete latent variables are often *less efficient* at representing information than continuous latent variables

Why?

Consider the following case:

- there are two factors (which describe the data) with 256 settings each
- we can describe the latent causes with two 8-bit numbers (we can model this with continuous latent variables)
- for clustering (discrete latent variables), we need 2^{16} ~= 10^5 numbers!

Continuous Latent Variable Models



see **Chapter 12.1** in C.M. Bishop, Pattern Recognition and Machine Learning (2007)

Example: Dimensionality reduction

MNIST single digit; 100x100 pixels; rotated and translated

- rotation and translation: only 3 degrees of freedom
- nonetheless: in pixel space: 100x100=10,000 degrees of freedom!
- knowing these 3 continuous latent variables (1x rotation, 2x translation) alone suffices to generate the the 100x100 pixel image

Question:

- How can we build a model that learns such a latent representation given the data (the images) alone?
- Example: Model that uncovers the “smile degree of freedom” in faces

[Tom White, https://twitter.com/dribnet]

PCA (Principal Component Analysis)

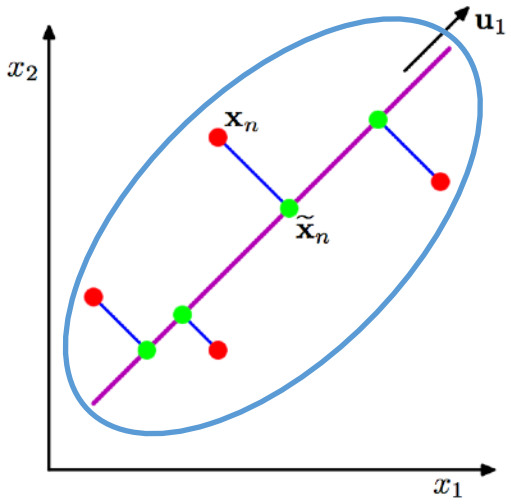
Core unsupervised learning method, simplest continuous latent variable model

Applications: Dimensionality reduction, sphering, compression, …

Dataset $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with $\mathbf{x}_n \in \mathbb{R}^D$

High-level idea:

- Fit a D-dimensional ellipsoid to the data (e.g. for D=2: ellipse)
- **For dimensionality reduction / compression**: project data to an M-dimensional (M<D) manifold spanned by the M longest principal axes
- **For sphering / whitening**: center and rotate ellipsoid, so that principal axes align with basis vectors; scale rotated data to have equal (unity) variance



PCA via variance maximization

Goal: Find M-dim hyperplane with maximum variance projection*

First: Consider the case of a 1-dim projection (M-dim case later)

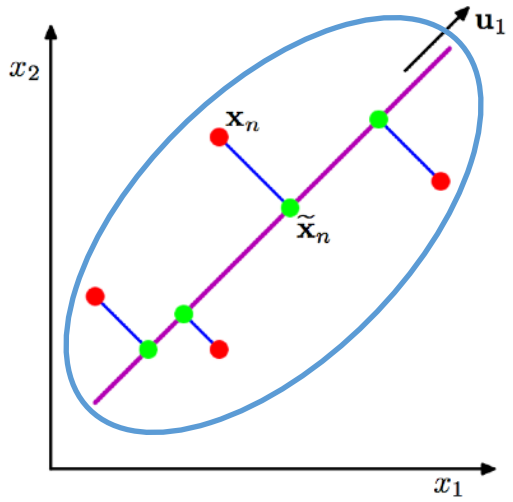
Let $\mathbf{u}_1$ be the first projection.

$$z_{n1} = \tilde{x}_n^1 = \mathbf{u}_1^T \mathbf{x}_n$$

single data vector

$$\bar{z}_1 = \frac{1}{N} \sum_n \mathbf{u}_1^T \mathbf{x}_n = \mathbf{u}_1^T \bar{\mathbf{x}}$$

mean projection



*Important assumption: variation contains information

PCA via variance maximization

$$\text{var}(z_1) = \frac{1}{N}\sum_{n=1}^{N}(z_{n1} - \bar{z}_1)^2$$

variance of projection

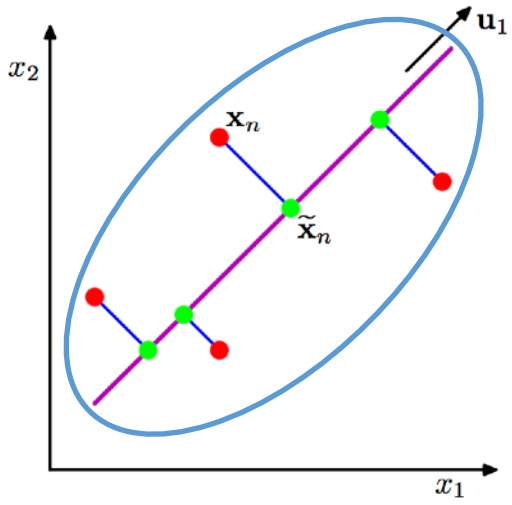
$$= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\bar{\mathbf{x}})^2$$

$$= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T(\mathbf{x}_n - \bar{\mathbf{x}}))(\mathbf{u}_1^T(\mathbf{x}_n - \bar{\mathbf{x}}))$$

$$= \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T\mathbf{u}_1$$

$$= \mathbf{u}_1^T\left(\underbrace{\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T/N}_{\mathbf{S}\text{: covariance matrix}}\right)\mathbf{u}_1$$

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



PCA via variance maximization

The (constrained) optimization problem is therefore:

$$\mathbf{u}_1 = \arg\max_{\mathbf{u}_1} \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 \qquad \text{subject to } \mathbf{u}_1^T \mathbf{u}_1 = 1$$

without the constraint, the objective can be made arbitrarily large.

Using Lagrange multipliers, the optimization function becomes:

$$f(\mathbf{u}_1) = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 - \lambda_1 (\mathbf{u}_1^T \mathbf{u}_1 - 1)$$

Therefore we have to solve:

$$\frac{\partial f(\mathbf{u}_1)}{\partial \mathbf{u}_1} = 2\mathbf{S}\mathbf{u}_1 - 2\lambda_1 \mathbf{u}_1 = 0$$

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

This is an **eigenvector/eigenvalue problem** with the solution λ_1 and $\mathbf{u}_1$

PCA via variance maximization

We can substitute $\mathbf{S}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$ into the variance:

$$\begin{aligned} \text{var}(z_1) &= \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 \\ &= \mathbf{u}_1^T \lambda_1 \mathbf{u}_1 \\ &= \lambda_1 \mathbf{u}_1^T \mathbf{u}_1 \\ &= \lambda_1 \end{aligned}$$

Therefore we can maximize our objective:

$$f(\mathbf{u}_1) = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 - \lambda_1 (\mathbf{u}_1^T \mathbf{u}_1 - 1)$$

By choosing $\mathbf{u}_1$ to be the (normalized) eigenvector of the largest eigenvalue λ_1

(We call $\mathbf{u}_1$ the principal component)

PCA: Optimizing M projections

In general, we want projections $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_M$

We define:

$$\mathbf{U} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \ldots \ \mathbf{u}_M] \quad \text{and} \quad \boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_M \end{bmatrix}$$

We want to maximize the following objective:

$$\begin{aligned} J(\mathbf{U}) &= \sum_{i=1}^{M} \mathbf{u}_i^T \mathbf{S} \mathbf{u}_i - \sum_{i=1}^{M} \lambda_i (\mathbf{u}_i^T \mathbf{u}_i - 1) \\ &= \mathrm{tr}(\mathbf{U}^T \mathbf{S} \mathbf{U}) - \mathrm{tr}(\mathbf{U} \boldsymbol{\Lambda} \mathbf{U}^T) + \mathrm{tr}(\boldsymbol{\Lambda}) \end{aligned}$$

(Notice the transposes within the trace)

PCA: Optimizing M projections

Taking the derivative with respect to $\mathbf{U}$ (using the Matrix Cookbook)

[https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf]

$$\frac{\partial J(\mathbf{U})}{\partial \mathbf{U}} = 2\mathbf{S}\mathbf{U} - 2\mathbf{U}\boldsymbol{\Lambda} = 0$$

$$\mathbf{S}\mathbf{U} = \mathbf{U}\boldsymbol{\Lambda}$$

Which is again an eigenvalue/eigenvector problem. We solve for $\mathbf{U}$ and $\boldsymbol{\Lambda}$.

Inserting this into our objective function yields:

$$\begin{aligned} J(\mathbf{U}) &= \mathrm{tr}(\mathbf{U}^T\mathbf{U}\boldsymbol{\Lambda}) - \mathrm{tr}(\boldsymbol{\Lambda}\mathbf{U}^T\mathbf{U}) + \mathrm{tr}(\boldsymbol{\Lambda}) \\ &= \mathrm{tr}(\mathbf{I}\boldsymbol{\Lambda}) - \mathrm{tr}(\boldsymbol{\Lambda}\mathbf{I}) + \mathrm{tr}(\boldsymbol{\Lambda}) \\ &= \mathrm{tr}(\boldsymbol{\Lambda}) \\ &= \sum_{i=1}^{M} \lambda_m \end{aligned}$$

Where we used (cyclicity of trace):

$$\mathrm{tr}(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T) = \mathrm{tr}(\boldsymbol{\Lambda}\mathbf{U}^T\mathbf{U})$$

and $\mathbf{U}^T\mathbf{U} = \mathbf{I}$

PCA: Optimizing M projections

So the final objective function is: $J(\mathbf{U}) = \sum_{i=1}^{M} \lambda_m$

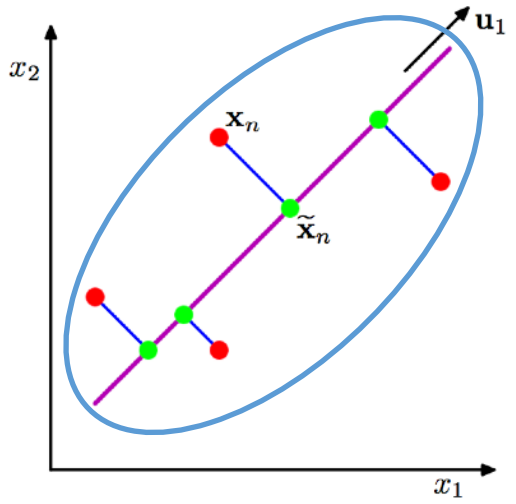
i.e. a sum over *M* eigenvalues of the covariance matrix **S**

This is maximized by taking the *M* largest eigenvalues!

The projection is given by the corresponding eigenvectors: $\mathbf{U} = [\mathbf{u}_1 \ \mathbf{u}_1 \ \ldots \ \mathbf{u}_M]$

Reminder (for single projection):

$$\begin{aligned} \mathrm{var}(z_1) &= \mathbf{u}_1^T \left(\underbrace{\sum_{n=1}^{N} (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T / N}_{\mathbf{S}\text{: covariance matrix}} \right) \mathbf{u}_1 \\ &= \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 \end{aligned}$$



PCA in practice

In Python, we can solve for the projections by:

1. `S = np.cov(X.T)` compute covariance
2. `U, L, V = np.linalg.svd(S)` singular value decomposition*

We can perform *dimensionality reduction* with the following projection:

$$\mathbf{z}_n = \mathbf{U}^T(\mathbf{x}_n - \bar{\mathbf{x}})$$

Note that we first have to subtract the mean.

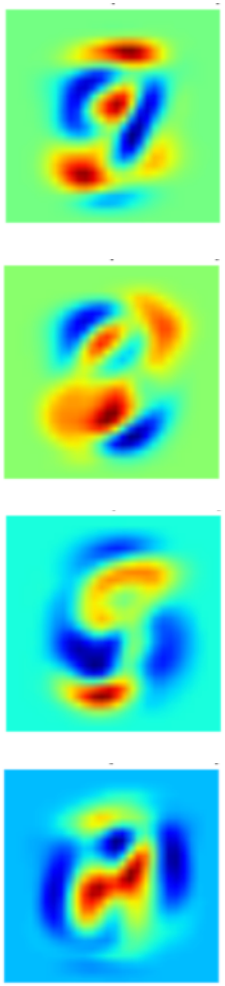
Reconstruct the original data from the *latent variables* alone:

$$\tilde{\mathbf{x}}_n = \mathbf{U}\mathbf{z}_n + \bar{\mathbf{x}}$$

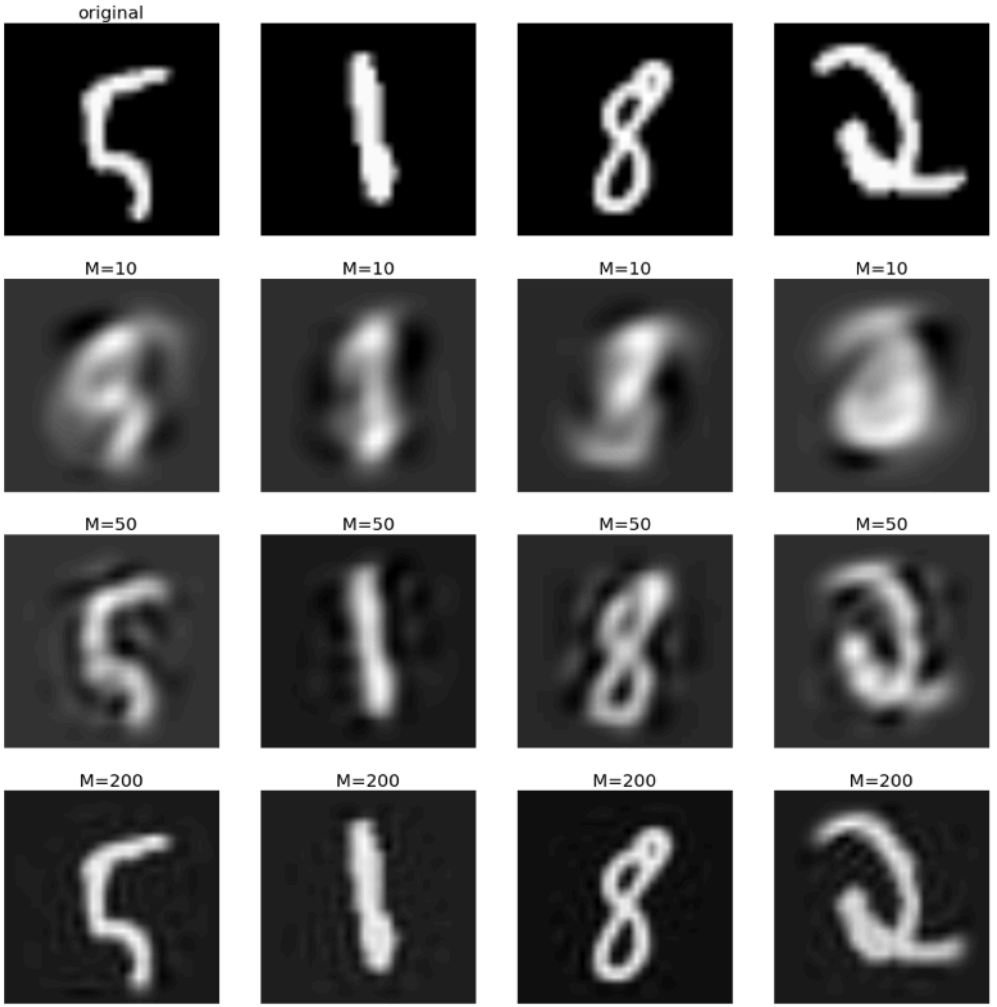
* equal to eigenvalue decomposition for positive semi-definite matrices, covariance matrices are positive semi-definite

Example: PCA on digits (MNIST)

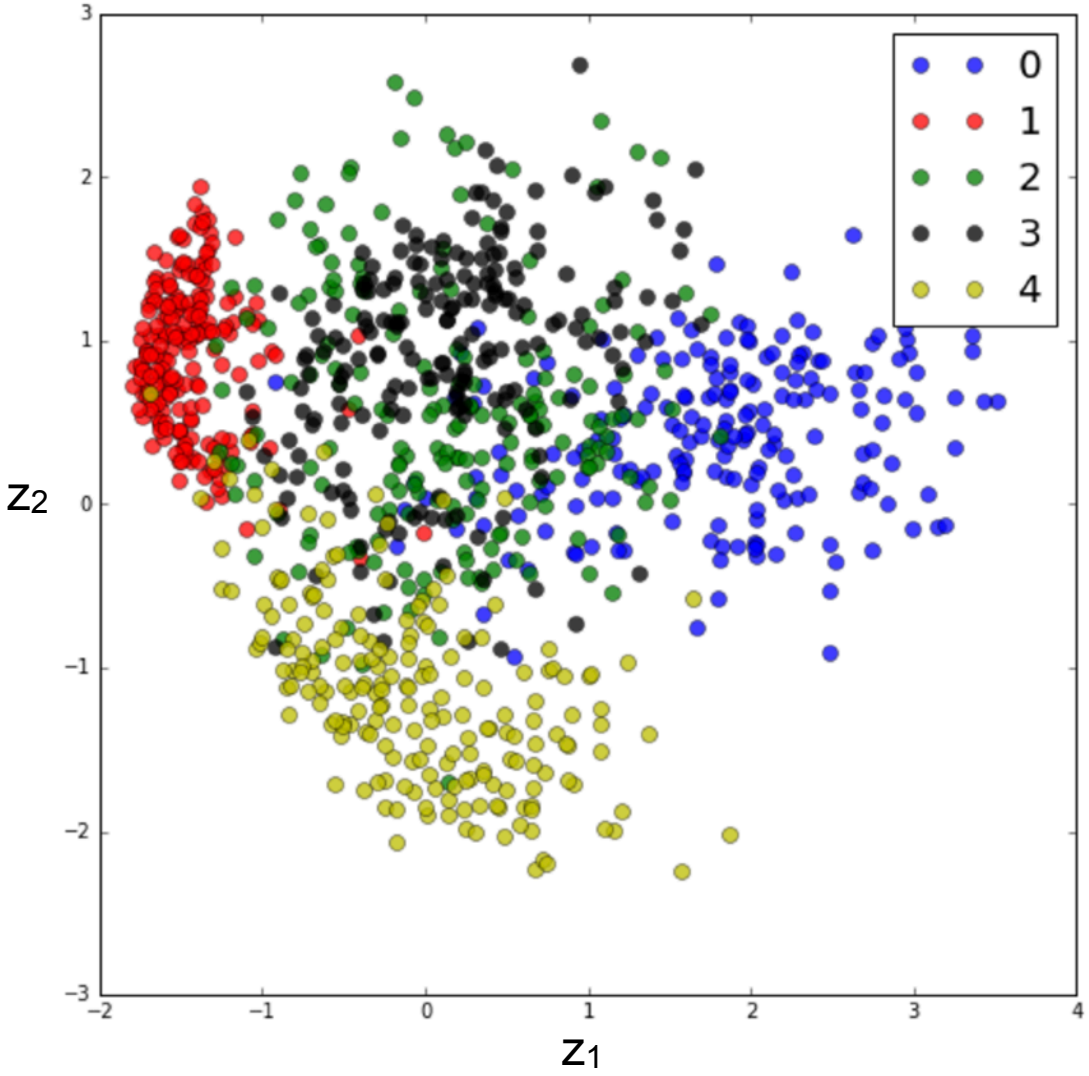
Eigenvectors:



Reconstructions:



Visualizing the latent space (MNIST)



Example: Eigenfaces with PCA

Idea: Express faces as linear combination of a small set of eigenfaces.

Original faces

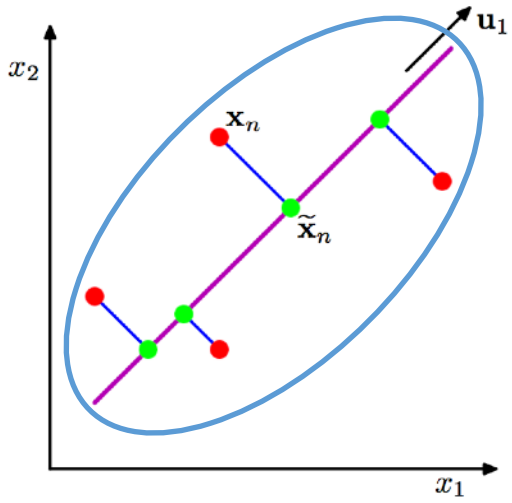
Eigenfaces

(first M=36 eigenvectors)

Recovered faces

[Labeled Faces in the Wild (cropped) dataset, LFWcrop, http://conradsanderson.id.au/lfwcrop/]

PCA via error minimization



Idea: Minimize the length of projection errors (blue lines in the figure)

Starting point: Full set of basis vectors $\{\mathbf{u}_i\}_{i=1}^{D}$ which are orthonormal, i.e. $\mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}$

We can express any vector as: $\mathbf{x}_n = \sum_{i=1}^{D} \alpha_{ni} \mathbf{u}_i$

with: $\alpha_{nj} = \mathbf{x}_n^T \mathbf{u}_j$

Now: Consider reconstruction using only the first *M* basis vectors and shared offsets for the remaining dimensions, i.e.:

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

PCA via error minimization

Goal: We want to minimize the *reconstruction error*:

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

We have to solve for optimal: $\{\mathbf{u}_i\}_{i=1}^{M}$, $\{b_i\}_{i=M+1}^{D}$, and $\{z_{ni}\}_{i=1}^{M}$

$$\{z_{ni}\}: \quad \frac{\partial C}{\partial z_{nj}} = \frac{1}{N} 2(\mathbf{x}_n - \tilde{\mathbf{x}}_n)^T \mathbf{u}_j = 0$$

$$\begin{aligned} \mathbf{x}_n^T \mathbf{u}_j &= \tilde{\mathbf{x}}_n^T \mathbf{u}_j \\ &= \left(\sum_{i=1}^{M} z_{ni} \mathbf{u}_i + \sum_{i=M+1}^{D} b_i \mathbf{u}_i \right)^T \mathbf{u}_j \\ &= \sum_{i=1}^{M} z_{ni} \mathbf{u}_i^T \mathbf{u}_j \quad = z_{nj} \end{aligned}$$

Therefore: $\boxed{z_{nj} = \mathbf{x}_n^T \mathbf{u}_j}$

PCA via error minimization

$\{b_i\}_{i=M+1}^{D}$:

$$\frac{\partial C}{\partial b_j} = \frac{1}{N}\sum_n 2(\mathbf{x}_n - \tilde{\mathbf{x}}_n)^T \mathbf{u}_j = 0$$

$$\underbrace{\frac{1}{N}\sum_n \mathbf{x}_n^T}_{\bar{\mathbf{x}}^T}\mathbf{u}_j = \frac{1}{N}\sum_n \tilde{\mathbf{x}}_n^T \mathbf{u}_j$$

$$= \frac{1}{N}\sum_n \left(\sum_{i=M+1}^{D} b_i \mathbf{u}_i\right)^T \mathbf{u}_j$$

$$= \frac{1}{N}\sum_n b_j = b_j$$

Therefore: $\boxed{b_j = \bar{\mathbf{x}}^T \mathbf{u}_j}$

PCA via error minimization

$\{\mathbf{u}_i\}_{i=1}^{M}$: This will be a bit tricky… Let us first rewrite the cost function.

We can simplify:

$$\begin{aligned}
\mathbf{x}_n - \tilde{\mathbf{x}}_n &= \sum_{i=1}^{D} \alpha_{ni}\mathbf{u}_i - \sum_{i=1}^{M} z_{ni}\mathbf{u}_i - \sum_{i=M+1}^{D} b_i\mathbf{u}_i \\
&= \underbrace{\sum_{i=1}^{D} (\mathbf{x}_n^T\mathbf{u}_i)\mathbf{u}_i - \sum_{i=1}^{M} (\mathbf{x}_n^T\mathbf{u}_i)\mathbf{u}_i}_{\text{first M terms cancel}} - \sum_{i=M+1}^{D} (\bar{\mathbf{x}}^T\mathbf{u}_i)\mathbf{u}_i \\
&= \sum_{i=M+1}^{D} (\mathbf{x}_n^T\mathbf{u}_i)\mathbf{u}_i - (\bar{\mathbf{x}}^T\mathbf{u}_i)\mathbf{u}_i \\
&= \sum_{i=M+1}^{D} \left((\mathbf{x}_n - \bar{\mathbf{x}})^T\mathbf{u}_i\right)\mathbf{u}_i
\end{aligned}$$

PCA via error minimization

$\{\mathbf{u}_i\}_{i=1}^{M}$ (continued) - The cost function now becomes:

$$
\begin{aligned}
C &= \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{x}_n - \tilde{\mathbf{x}}_n||^2 \\
&= \frac{1}{N}\sum_{n=1}^{N} || \sum_{i=M+1}^{D} \left((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i\right) \mathbf{u}_i||^2 \\
&= \frac{1}{N}\sum_{n=1}^{N} \left(\sum_{i=M+1}^{D} \left((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i\right) \mathbf{u}_i \right)^T \left(\sum_{i=M+1}^{D} \left((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i\right) \mathbf{u}_i \right) \\
&= \frac{1}{N}\sum_{n=1}^{N} \sum_{i=M+1}^{D} \left(\left((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i\right) \mathbf{u}_i\right)^T \left(\left((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i\right) \mathbf{u}_i\right) \\
&= \frac{1}{N}\sum_{n=1}^{N} \sum_{i=M+1}^{D} \mathbf{u}_i^T(\mathbf{x}_n - \bar{\mathbf{x}})\mathbf{u}_i^T\mathbf{u}_i(\mathbf{x}_n - \bar{\mathbf{x}})^T\mathbf{u}_i \qquad \ldots
\end{aligned}
$$

PCA via error minimization

$$C = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D} \mathbf{u}_i^T(\mathbf{x}_n - \bar{\mathbf{x}})\mathbf{u}_i^T\mathbf{u}_i(\mathbf{x}_n - \bar{\mathbf{x}})^T\mathbf{u}_i$$ (from last slide)

$$= \sum_{i=M+1}^{D} \mathbf{u}_i^T\mathbf{S}\mathbf{u}_i$$

$$= \mathrm{tr}(\mathbf{U}^T\mathbf{S}\mathbf{U})$$ We want to minimize this objective!

Remember: $\{\mathbf{u}_i\}_{i=1}^{M}$ are constrained to be orthonormal

We include this constraint in the objective with Lagrange multipliers (as before):

$$C = \mathrm{tr}(\mathbf{U}^T\mathbf{S}\mathbf{U}) - \mathrm{tr}(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T) + \mathrm{tr}(\boldsymbol{\Lambda})$$

$$\frac{\partial C}{\partial \mathbf{U}} = 2\mathbf{S}\mathbf{U} - 2\mathbf{U}\boldsymbol{\Lambda} = 0$$

$$\mathbf{S}\mathbf{U} = \mathbf{U}\boldsymbol{\Lambda}$$ **Again an eigenvalue problem!**

PCA via error minimization

We plug the solution of the eigenvalue problem back into the cost function:

$$\begin{aligned} C &= \mathrm{tr}(\mathbf{U}^T\mathbf{U}\boldsymbol{\Lambda}) - \mathrm{tr}(\boldsymbol{\Lambda}\mathbf{U}^T\mathbf{U}) + \mathrm{tr}(\boldsymbol{\Lambda}) \\ &= \mathrm{tr}(\mathbf{I}\boldsymbol{\Lambda}) - \mathrm{tr}(\boldsymbol{\Lambda}\mathbf{I}) + \mathrm{tr}(\boldsymbol{\Lambda}) \\ &= \mathrm{tr}(\boldsymbol{\Lambda}) \\ &= \sum_{i=M+1}^{D} \lambda_i \end{aligned}$$

i.e. *C* is minimized by choosing the $(D-M)$ eigenvectors $\{\mathbf{u}_i\}_{i=M+1}^{D}$ with the smallest eigenvalues.

$\{\mathbf{u}_i\}_{i=1}^{M}$ is therefore the set of eigenvectors with the M largest eigenvalues!

Summary: $\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} z_{ni}\mathbf{u}_i + \sum_{i=M+1}^{D} b_i\mathbf{u}_i$ $\quad z_{nj} = \mathbf{x}_n^T\mathbf{u}_j$ $\quad b_j = \bar{\mathbf{x}}^T\mathbf{u}_j$

PCA: Discussion

Things to keep in mind:

- Standard PCA is non-probabilistic
  (can be seen as ML solution of Probabilistic PCA)
- See Bishop for probabilistic extensions
- PCA is linear (therefore limited modeling capacity)
- PCA can be solved exactly
  (take M eigenvectors with largest eigenvalues)
- Solving for eigenvectors is expensive ($\mathcal{O}(D^3)$)

Applications:

- Dimensionality reduction
- (Lossy) compression
- Whitening/sphering
- Data visualization
- Data exploration
- …

PCA in high dimensions

The curse of dimensionality:

"A drunk man will find his way home,
but a drunk bird may get lost forever."

- Shizuo Kakutani

What to do if *D* is very large? PCA is $\mathcal{O}(D^3)$!

First: Let's assume that data is centered (i.e. mean subtracted).

Covariance matrix: $\mathbf{S} = \frac{1}{N}\mathbf{X}^T\mathbf{X}$ $(D \text{ by } D)$

We have to solve the eigenvalue problem: $\mathbf{SU} = \mathbf{U}\boldsymbol{\Lambda}$

Idea: Compute eigenvectors of $\mathbf{X}\mathbf{X}^T$ (*N* by *N*) instead

(this can be a lot cheaper if *N* is smaller than *D*)

PCA in high dimensions

Relate $\mathbf{X}^T\mathbf{X}$ to $\mathbf{X}\mathbf{X}^T$:

$$\mathbf{S}\mathbf{U} = \mathbf{U}\boldsymbol{\Lambda}$$

$$\frac{1}{N}\mathbf{X}^T\mathbf{X}\mathbf{U} = \mathbf{U}\boldsymbol{\Lambda}$$

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T\mathbf{X}\mathbf{U} = \mathbf{X}\mathbf{U}\boldsymbol{\Lambda} \qquad \text{left multiply by } \mathbf{X}$$

$$\frac{\mathbf{X}\mathbf{X}^T}{N}\underbrace{(\mathbf{X}\mathbf{U})}_{\mathbf{V}} = (\mathbf{X}\mathbf{U})\boldsymbol{\Lambda}$$

$$\frac{\mathbf{X}\mathbf{X}^T}{N}\mathbf{V} = \mathbf{V}\boldsymbol{\Lambda} \qquad \text{with} \quad \mathbf{V} = \mathbf{X}\mathbf{U}$$

PCA in high dimensions

Working back from $\mathbf{V}$ to $\mathbf{U}$:

$$\frac{\mathbf{X}\mathbf{X}^T}{N}\mathbf{V} = \mathbf{V}\boldsymbol{\Lambda}$$

$$\mathbf{X}^T\frac{\mathbf{X}\mathbf{X}^T}{N}\mathbf{V} = \mathbf{X}^T\mathbf{V}\boldsymbol{\Lambda} \qquad \text{left-multiply by } \mathbf{X}^T$$

$$\frac{\mathbf{X}^T\mathbf{X}}{N}(\mathbf{X}^T\mathbf{V}) = (\mathbf{X}^T\mathbf{V})\boldsymbol{\Lambda}$$

$$\frac{\mathbf{X}^T\mathbf{X}}{N}\mathbf{U} = \mathbf{U}\boldsymbol{\Lambda}$$

So we have: $\mathbf{U} = \mathbf{X}^T\mathbf{V}$

PCA in high dimensions

Recipe for PCA in high dimensions:

1) Solve for eigenvectors $\mathbf{V}$ of $\frac{\mathbf{X}\mathbf{X}^T}{N}$

2) Get eigenvectors $\mathbf{U}$ of covariance matrix via:

$$\mathbf{U} = N^{-1/2}\boldsymbol{\Lambda}^{-1/2}\mathbf{X}^T\mathbf{V}$$

Note: The normalization terms arise when we require **U** and **V** to be matrices of normalized eigenvectors

This is now $\mathcal{O}(N^3)$

Sphering with PCA

Typical pre-processing step: Normalize data to have *zero mean* and *unit variance*

With PCA, we can go one step further: *decorrelate features*

In other words: express features in terms of orthogonal basis vectors

This *rotated* dataset has a *diagonal covariance matrix.*

Sphering refers to the process of

1. Centering each data vector: $\mathbf{x}_n - \bar{\mathbf{x}}$
2. Rotating (decorrelating features): $\mathbf{y}_n = \mathbf{U}^T(\mathbf{x}_n - \bar{\mathbf{x}})$
3. Rescaling the projected values to have unit variance

$$\tilde{\mathbf{y}}_n = \mathbf{\Lambda}^{-1/2}\mathbf{U}^T(\mathbf{x}_n - \bar{\mathbf{x}})$$

Sphering with PCA

Single feature: $\tilde{y}_{in} = \lambda_i^{-1/2} \mathbf{u}_i^T (\mathbf{x}_n - \bar{\mathbf{x}})$

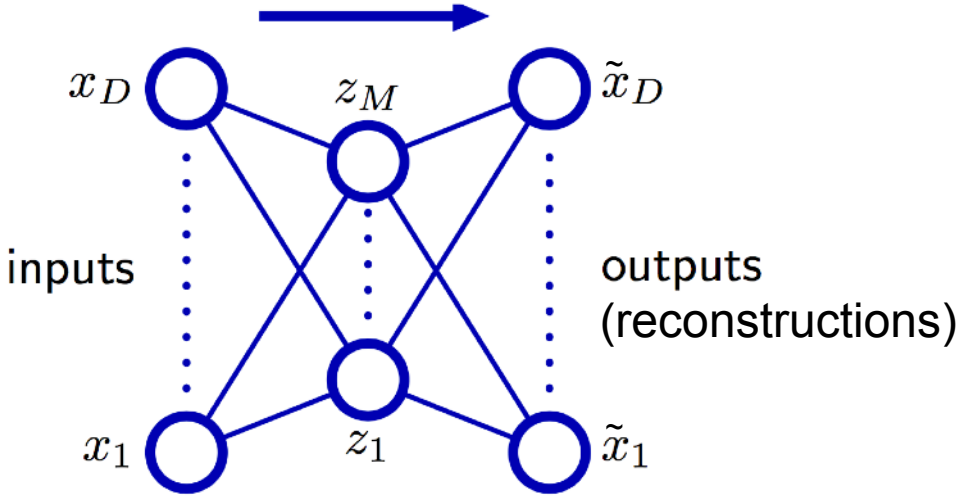
Covariance:

$$\begin{aligned}
\sigma_{ij} &= \frac{1}{N} \sum_n \tilde{y}_{in} \tilde{y}_{jn} \\
&= \frac{1}{N} \sum_n \left(\lambda_i^{-1/2} \mathbf{u}_i^T (\mathbf{x}_n - \bar{\mathbf{x}}) \right) \left((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_j \lambda_j^{-1/2} \right) \\
&= \lambda_i^{-1/2} \lambda_j^{-1/2} \mathbf{u}_i^T \left(\frac{1}{N} \sum_n (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \right) \mathbf{u}_j \\
&= \lambda_i^{-1/2} \lambda_j^{-1/2} \underbrace{\mathbf{u}_i^T \mathbf{S} \mathbf{u}_j}_{=0 \text{ if } i \neq j \quad \lambda_i \text{ o.w.}} \\
&= \lambda_i^{-1/2} \lambda_j^{-1/2} \lambda_i \delta_{ij} \\
&= \lambda_i^{-1/2} \lambda_i^{-1/2} \lambda_i \delta_{ij} \\
&= \delta_{ij}
\end{aligned}$$

Autoencoders

PCA can be seen as a linear version of an *autoencoder.*

An *autoencoder* is a neural network that produces its own inputs as output.



The mappings $\mathbf{z} = f(\mathbf{x})$ and $\tilde{\mathbf{x}} = g(\mathbf{z})$ both are neural networks.

The goal is then to minimize the *reconstruction error* between inputs and outputs.

PCA as special case: $f(\mathbf{x}) = \mathbf{U}^T\mathbf{x}$ and $g(\mathbf{z}) = \mathbf{U}\mathbf{z}$

(We assume that data is centered)

Kernel PCA

PCA can be "kernelized"

PCA in feature space: $\mathbf{C} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}, \quad \mathbf{C}\mathbf{v}_i = \lambda_i\mathbf{v}_i$

Eigenvalue problem in kernel space: $\mathbf{K}\mathbf{a}_i = \lambda_i N\mathbf{a}_i$

with: $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$ (Gram matrix)

Dimensionality reduction:

$$z_i(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{v}_i = \sum_{n=1}^{N} a_{in}\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) = \sum_{n=1}^{N} a_{in}k(\mathbf{x}, \mathbf{x}_n)$$

Strictly speaking, we have to solve the eigenvalue problem for

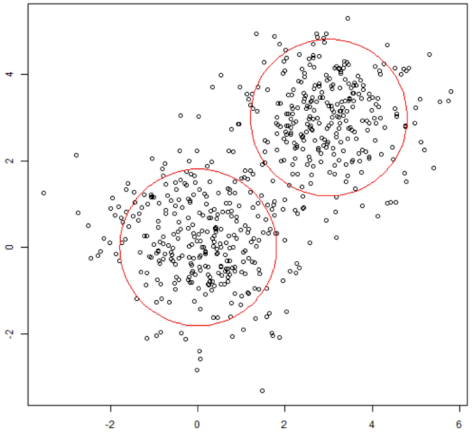
$$\widetilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N\mathbf{K} - \mathbf{K}\mathbf{1}_N + \mathbf{1}_N\mathbf{K}\mathbf{1}_N$$

as we cannot assume that the feature vectors are zero-centered.

Summary

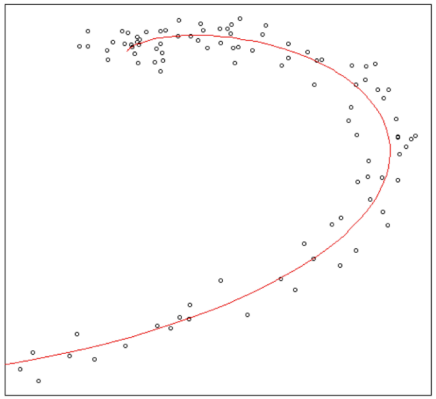
Clustering (discrete latent variables):

- K-Means (introduction, derivation, full algorithm)
- Image compression with K-Means
- Issues with discrete latent variables



Dimensionality reduction (continuous latent variables):

- Standard PCA algorithm
- Derivation via variance maximization
- Derivation via error minimization
- Examples: MNIST compression, eigenfaces
- Latent space visualization
- Sphering with PCA
- PCA in high dimensions
- Autoencoders & Kernel PCA



Further reading

Blog post on eigenfaces:
http://mikedusenberry.com/on-eigenfaces/

Blog post on dimensionality reduction:
http://colah.github.io/posts/2014-10-Visualizing-MNIST/

Questions? You can get in touch with me via:

- E-Mail: T.N.Kipf@uva.nl
- Twitter: @thomaskipf
- Web: http://tkipf.github.io

The lecture slides are based on Ted Meeds' lecture notes (ML1, UvA, 2015) and on chapters 9.1, 12.1 & 12.3 in C.M. Bishop, Pattern Recognition and Machine Learning (2007)