ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Introduction to Machine Learning

Unsupervised Learning: Dimension Reduction

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)

Typical approaches

- Assume $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$
- Obtain mapping $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^k$ where $k \ll d$
- Can distinguish
  - Linear dimension reduction: $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$
  - Nonlinear dimension reduction (parametric or non-parametric)

$A \in \mathbb{R}^{k \times d}$

- **Key question**: Which mappings should we prefer?

Principal component analysis (PCA)

- Given centered data $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$, $1 \leq k \leq d$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T \qquad \mu = \frac{1}{n}\sum_{i} \mathbf{x}_i = 0$$

- The solution to the PCA problem

$$(\mathbf{W}, \mathbf{z}_1, \ldots, \mathbf{z}_n) = \arg\min \sum_{i=1}^{n} ||\mathbf{W}\mathbf{z}_i - \mathbf{x}_i||_2^2$$

where $\mathbf{W} \in \mathbb{R}^{d \times k}$ is orthogonal, $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$

is given by $\mathbf{W} = (\mathbf{v}_1 | \ldots | \mathbf{v}_k)$ and $\mathbf{z}_i = \mathbf{W}^T \mathbf{x}_i$

where

$$\Sigma = \sum_{i=1}^{d} \lambda_i \mathbf{v}_i \mathbf{v}_i^T \qquad \lambda_1 \geq \cdots \geq \lambda_d \geq 0$$

Side note: PCA vs. K-Means

PCA Problem:

$$(\mathbf{W}, \mathbf{z}_1, \ldots, \mathbf{z}_n) = \arg\min \sum_{i=1}^{n} ||\mathbf{W}\mathbf{z}_i - \mathbf{x}_i||_2^2$$

where $\mathbf{W} \in \mathbb{R}^{d \times k}$ is orthogonal, $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$

k-Means problem: (equivalent formulation)

$W = (\mu_1 | \ldots | \mu_k)$

$$(\mathbf{W}, \mathbf{z}_1, \ldots, \mathbf{z}_n) = \arg\min \sum_{i=1}^{n} ||\mathbf{W}\mathbf{z}_i - \mathbf{x}_i||_2^2$$

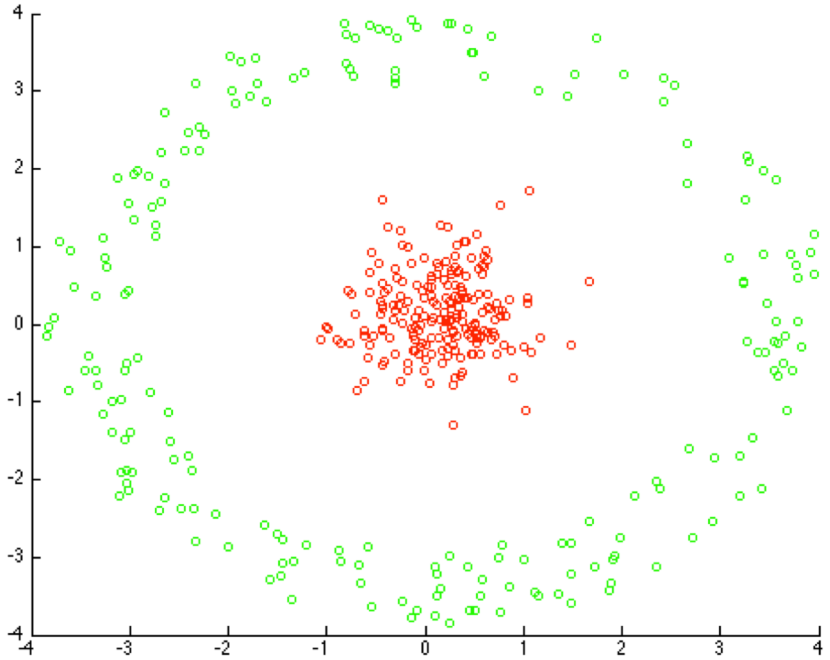
$W \cdot e_j = \mu_j$

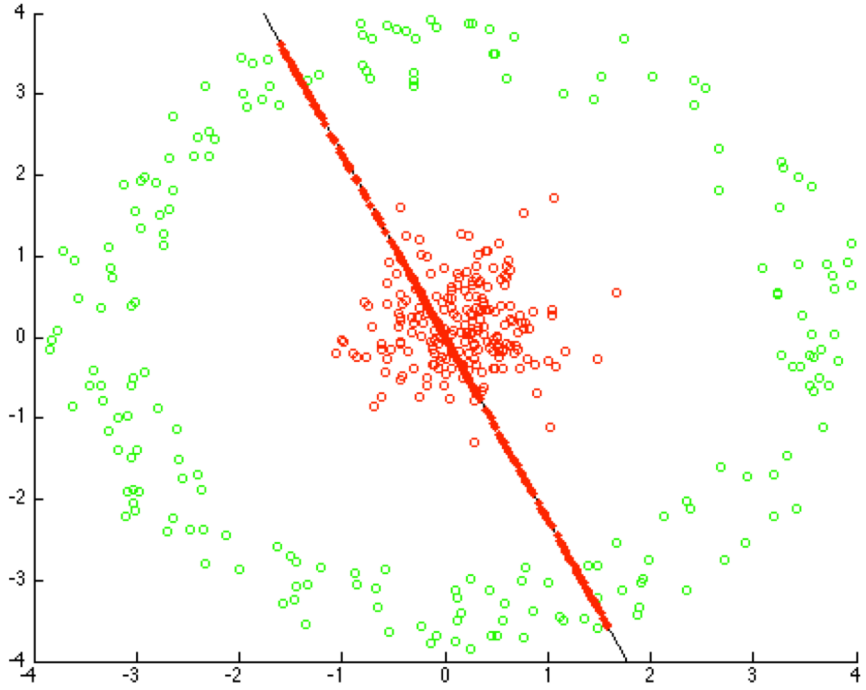
where $\mathbf{W} \in \mathbb{R}^{d \times k}$ *arbitrary* and $\mathbf{z}_1, \ldots, \mathbf{z}_n \in E_k$

hereby, $E_k = \{[1, 0, \ldots, 0], \ldots, [0, \ldots, 0, 1]\}$
is the set of unit vectors in $\mathbb{R}^k$

$e_j = [0 \ldots 1, 0 \ldots 0] \in E_k$
$\uparrow$ j-th pos.

Another example





Use Kernels!

- *Recall*: In supervised learning, kernels allowed us to solve non-linear problems by reducing them to linear ones in high-dimensional (implicitly represented) spaces
- Can take the same approach for unsupervised learning!

Ansatz: $w = \sum_{i=1}^{n} \alpha_i x_i$

$x \in \mathbb{R}^d \longmapsto \phi(x) \in \mathbb{R}^D \longmapsto z = w^T \phi(x) \in \mathbb{R}^k$

Why: for PCA, w^* is (only) eigenvector of $X^T X$

$\Rightarrow X^T X w = \lambda w \Rightarrow w = \frac{1}{\lambda} \sum_i \beta_i x_i$

($\beta = Xw$, $\sum_i \beta_i x_i = X^T \beta$)

$\Rightarrow \alpha_i = \frac{\beta_i}{\lambda}$

Recall PCA for k=1

- Optimal solution to PCA problem solves, for $\Sigma = \frac{1}{n} \boldsymbol{X}^T \boldsymbol{X}$

$$\arg \max_{||\mathbf{w}||_2=1} \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} = \arg \max_{||\mathbf{w}||_2=1} \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i)^2$$

(s.t.) (f)

Using Ansatz $w = \sum_i \alpha_i x_i$, show that both objective (f) and constraints (s.t.) can be be expressed in terms α and inner products of x_i, x_j

Derivation continued

Objective

$$w^T X^T X w = \sum_{i=1}^{n} (w^T x_i)^2 = \sum_{i=1}^{n} \left(\left(\sum_{j=1}^{n} \alpha_j x_j \right)^T x_i \right)^2$$

$$= \sum_{i=1}^{n} \left(\sum_{j} \alpha_j \underbrace{(x_j^T x_i)}_{k(x_i, x_j)} \right)^2 = \sum_{i=1}^{n} \left(\sum_{j} \alpha_j k(x_i, x_j) \right)^2 = \sum_{i=1}^{n} (\alpha^T k_i)^2$$

$$K = \begin{pmatrix} | & & | \\ k_1 & \cdots & k_n \\ | & & | \end{pmatrix}$$

$$= \alpha^T K^T K \alpha \to \max$$

Constraints

$$\|w\|_2 = w^T w = \left(\sum_{i=1}^{n} \alpha_i x_i \right)^T \left(\sum_{j=1}^{n} \alpha_j x_j \right) = \sum_{ij} \alpha_i \alpha_j \underbrace{x_i^T x_j}_{k(x_i, x_j)}$$

$$= \alpha^T K \alpha \stackrel{!}{=} 1$$

Thus, kernel PCA requires solving

$$\max_{\alpha} \; \alpha^T K^T K \alpha \quad \text{s.t.} \quad \alpha^T K \alpha = 1$$

regular PCA: $\max_{w} \; w^T X^T X w \quad \text{s.t.} \quad w^T w = 1$

Recall PCA for k=1

- Optimal solution to PCA problem solves, for $\Sigma = \frac{1}{n}\boldsymbol{X}^T\boldsymbol{X}$

$$\arg \max_{||\mathbf{w}||_2=1} \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} = \arg \max_{||\mathbf{w}||_2=1} \sum_{i=1}^{n} (\mathbf{w}^T\mathbf{x}_i)^2$$

- Applying feature maps, using $\mathbf{w} = \sum_{j=1}^{n} \alpha_j \phi(\mathbf{x}_j)$ and observing $||\mathbf{w}||_2^2 = \alpha^T\mathbf{K}\alpha$

$$\arg \max_{||\mathbf{w}||_2=1} \sum_{i=1}^{n} (\mathbf{w}^T\phi(\mathbf{x}_i))^2 = \arg \max_{\alpha^T\mathbf{K}\alpha=1} \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_i) \right)^2$$

$$= \arg \max_{\alpha^T\mathbf{K}\alpha=1} \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) \right)^2 = \arg \max_{\alpha^T\mathbf{K}\alpha=1} \sum_{i=1}^{n} \left(\alpha^T\mathbf{K}_i \right)^2$$

$$= \arg \max_{\alpha^T\mathbf{K}\alpha=1} \alpha^T\mathbf{K}^T\mathbf{K}\alpha$$

Kernel PCA (k=1)

- The Kernel-PCA problem (k=1) requires solving

$$\alpha^* = \arg \max_{\alpha^T \mathbf{K} \alpha = 1} \alpha^T \mathbf{K}^T \mathbf{K} \alpha$$

- The optimal solution is obtained in **closed form** from the eigendecomposition of ***K***:

$$\alpha^* = \frac{1}{\sqrt{\lambda_1}} \mathbf{v}_1$$

Why scaling?

$\alpha^{*T} K \alpha^* = \frac{1}{\sqrt{\lambda_1}} v_1^T K \frac{1}{\sqrt{\lambda_1}} v_1$

$= \frac{1}{\lambda_1} \cdot v_1^T K v_1 = \frac{1}{\lambda_1} \cdot v_1^T (\lambda_1 v_1) = v_1^T v_1 = 1$

$$\mathbf{K} = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mathbf{v}_i^T \qquad \lambda_1 \geq \cdots \geq \lambda_d \geq 0$$

Kernel PCA (general k)

- For general k ≥ 1, the Kernel Principal Components are given by $\alpha^{(1)}, \ldots, \alpha^{(k)} \in \mathbb{R}^n$

where $\alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}} \mathbf{v}_i$ $\in \mathbb{R}^n$

is obtained from: $\mathbf{K} = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mathbf{v}_i^T \quad \lambda_1 \geq \cdots \geq \lambda_d \geq 0$

- Given this, a new point ***x*** is projected as $\mathbf{z} \in \mathbb{R}^k$

For $i \in \{1, \ldots, k\}$: $z_i = w^{(i)T} x = \left(\sum_{j=1}^{n} \alpha_j^{(i)} x_j \right)^T x = \sum_{j=1}^{n} \alpha_j^{(i)} \underbrace{x_j^T x}_{k(x_j, x)} = \sum_{j=1}^{n} \alpha_j^{(i)} k(x_j, x)$

Kernel PCA (general k)

- For general k ≥ 1, the Kernel Principal Components are given by $\alpha^{(1)}, \ldots, \alpha^{(k)} \in \mathbb{R}^n$

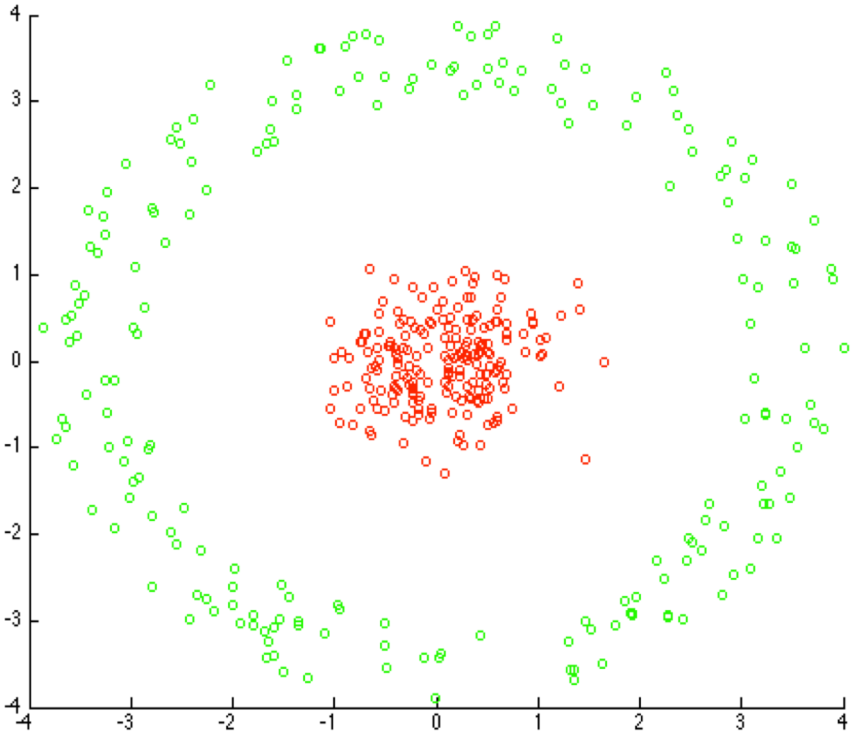
where $$\alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}} \mathbf{v}_i$$

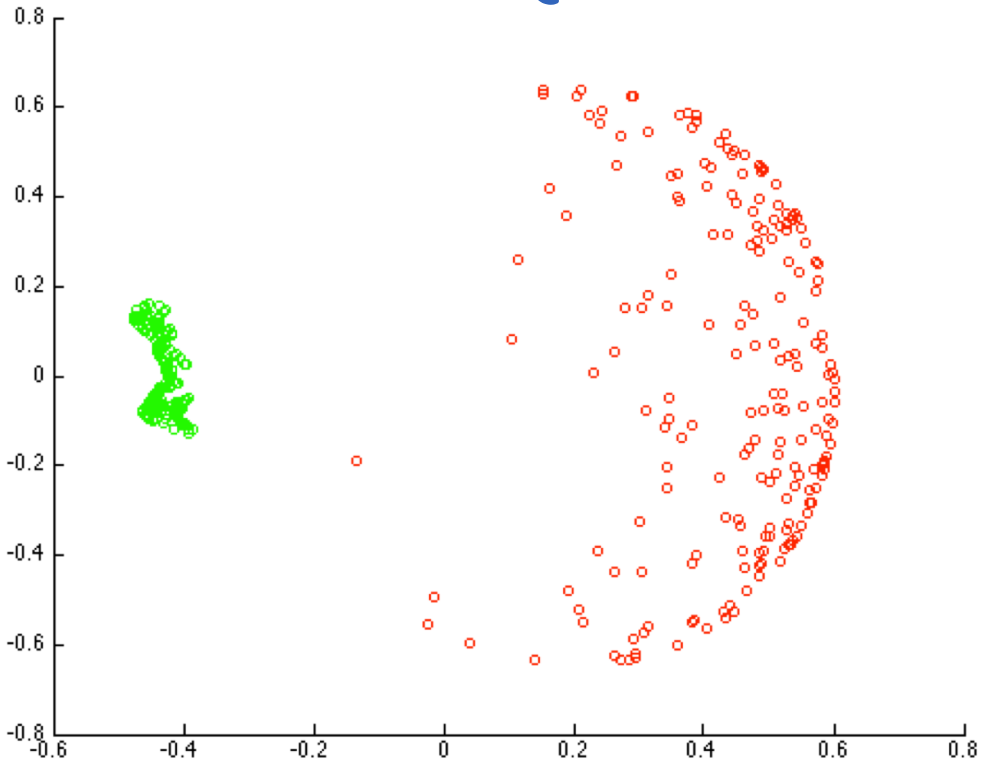
is obtained from: $$\mathbf{K} = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mathbf{v}_i^T \quad \lambda_1 \geq \cdots \geq \lambda_d \geq 0$$

- Given this, a new point ***x*** is projected as $\mathbf{z} \in \mathbb{R}^k$

$$z_i = \sum_{j=1}^{n} \alpha_j^{(i)} k(\mathbf{x}, \mathbf{x}_j)$$

Illustration





Side note: Applying k-means on kernel-principal components is sometimes called Kernel-k-means or Spectral Clustering

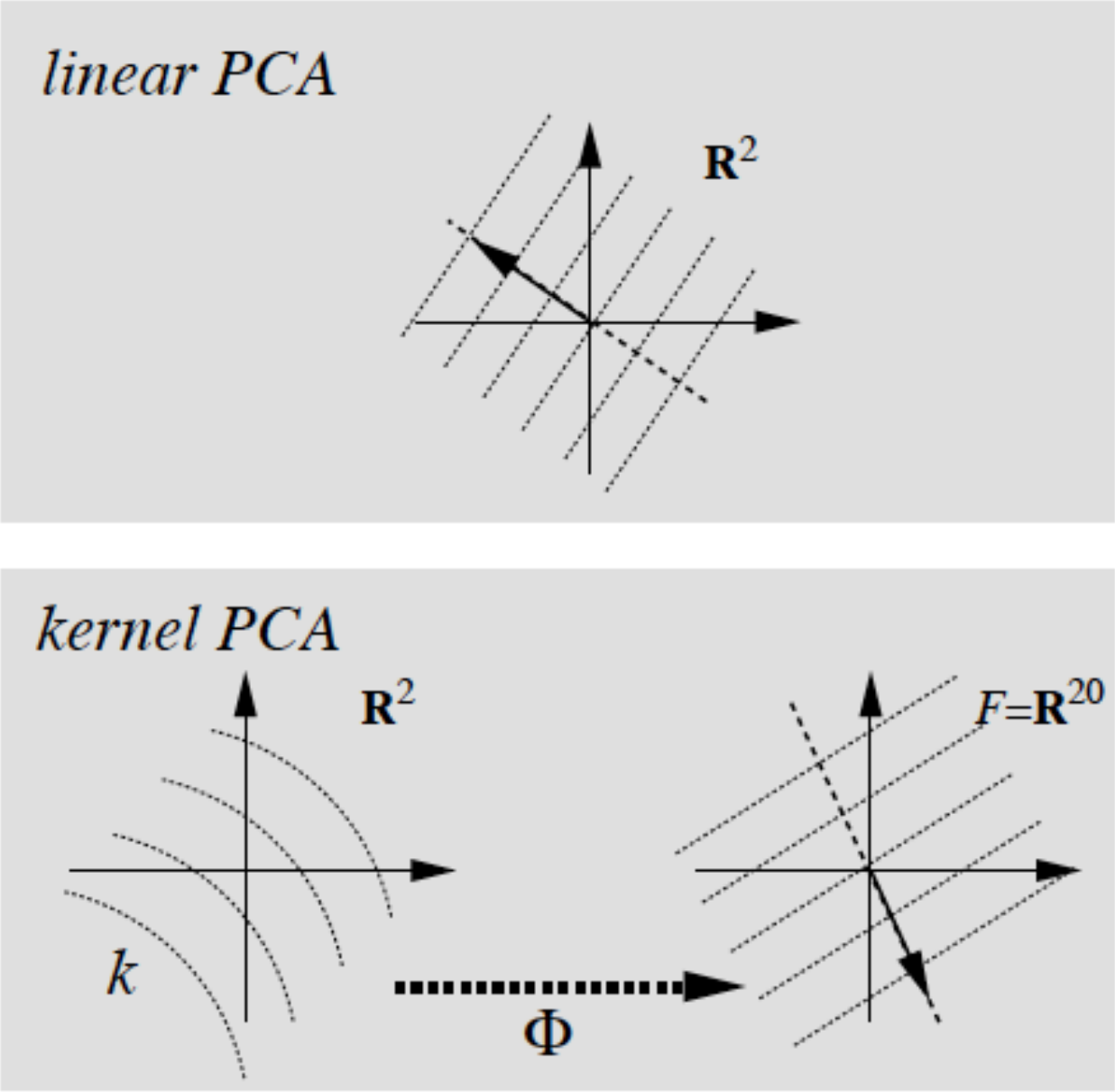
Kernel-PCA/k-means demo

Notes on Kernel-PCA

- Kernel-PCA corresponds to applying PCA in the feature space induced by the kernel *k*
- Can be used to discover non-linear feature maps in closed form
- This can be used as inputs, e.g., to SVMs given „multi-layer support vector machines“
- May want to „center“ the kernel: $\mathbf{E} = \frac{1}{n}[1,\ldots,1][1,\ldots,1]^T$

$$\mathbf{K}' = \mathbf{K} - \mathbf{KE} - \mathbf{EK} + \mathbf{EKE}$$

Illustration: Kernel-PCA [Schölkopf et al.]



Other non-linear methods

- Kernel-PCA requires data specified as kernel
  - Complexity grows with the number of data points
  - Cannot easily „explicitly“ embed high-dimensional data (unless we have an appropriate kernel)

- Alternatives
  - Autoencoders
  - Locally linear embedding (LLE)
  - Multi-dimensional scaling
  - t-SNE
  - …

Dimension reduction via Autoencoders

- Key idea: Try to learn the identity function!

$$\mathbf{x} \approx f(\mathbf{x}; \theta)$$

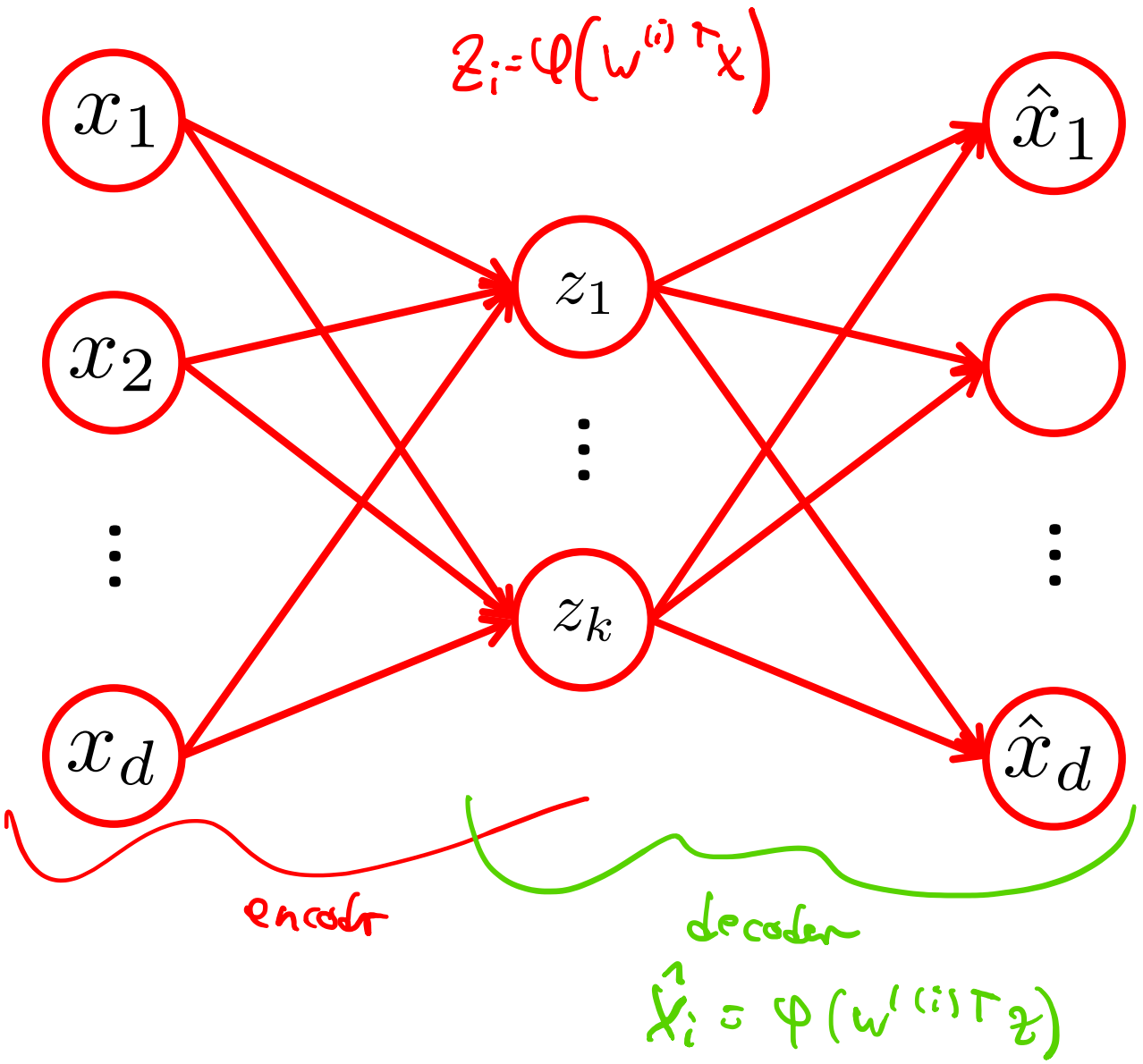
- What function f should we pick?

$$f(\mathbf{x}; \theta) = f_2(f_1(\mathbf{x}; \theta_1); \theta_2)$$

encoder

decoder

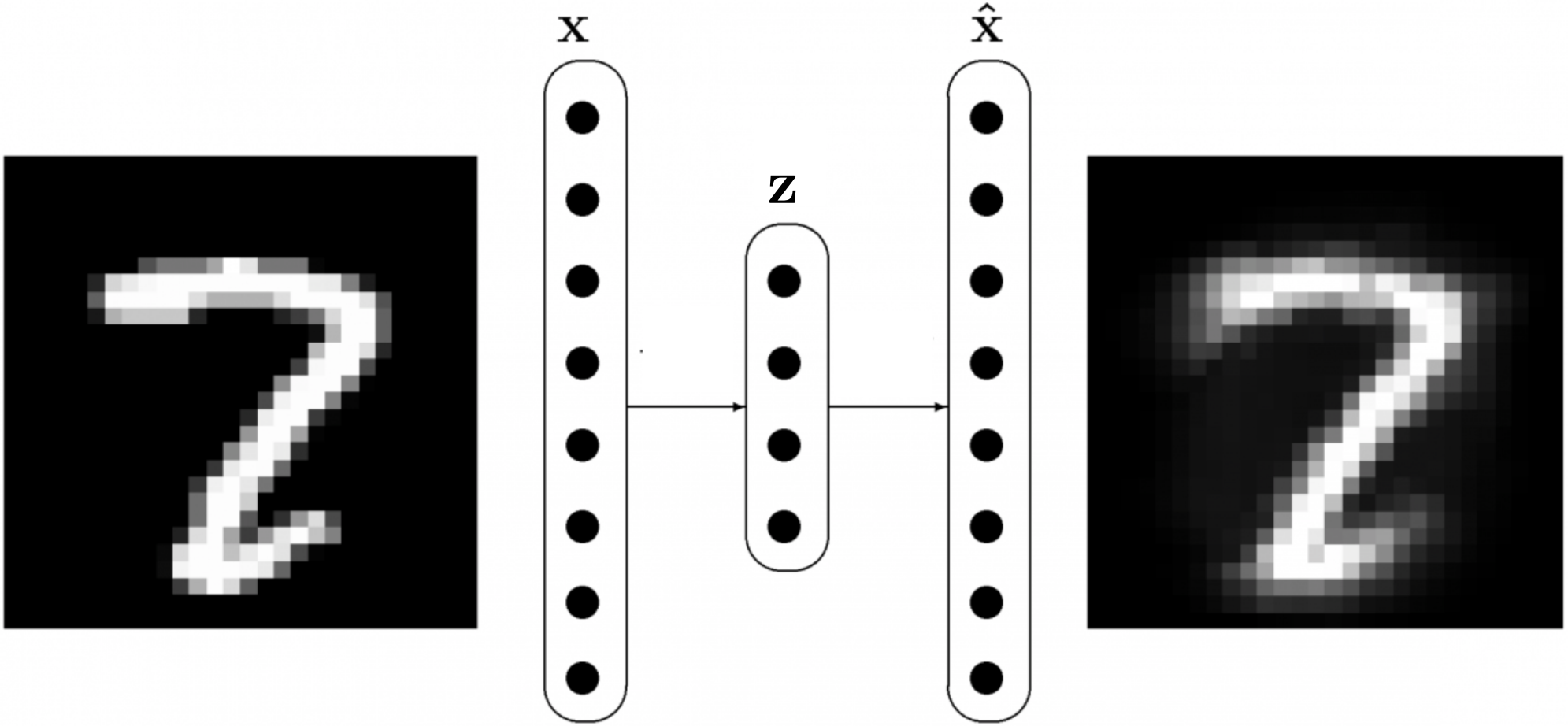
Autoencoders via Neural Nets



Neural network autoencoders

- Neural network Autodecoders are ANNs where
  - There is one output unit for each of the *d* input units
  - The number *k* of hidden units is usually smaller than the number of inputs
- The goal is to optimize the weights such that the output agrees with the input

Example



[based on cor-lab.de, Lemme et al.]

Training autoencoders

- The goal is to optimize the weights such that the output agrees with the input
  - For example, minimize the square loss

$$\min_{\mathbf{W}} \sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{f}(\mathbf{x}_i; \mathbf{W})||_2^2$$

$$= \sum_{j=1}^{d} (x_{ij} - f_j(x_i; W))^2$$

- Find local minimum via SGD (backpropagation)
- Initialization matters and is challenging
  - C.f., work on „pretraining“ (e.g., layerwise training of restricted Boltzmann machines)

Autoencoders vs. PCA

- The internal representation $z = \varphi(\boldsymbol{W}^{(1)}\boldsymbol{x})$ is the „dimensionality reduced“ input $\boldsymbol{x}$

- If the activation function is the identity $\varphi(z) = z$ then in fact fitting a NN autoencoder is equivalent to PCA!

$W_1 \in \mathbb{R}^{k \times d}$
$W_2 \in \mathbb{R}^{d \times k}$

$f(x) = f_2(f_1(x; W_1); W_2) = W_2 W_1 \cdot x$

The optimal solution to $\min_{W_1, W_2} \sum_i \| x_i - W_2 W_1 x_i \|_2^2$

is given by $W_1 = W^T$ and $W_2 = W$, where W^T is the PCA solution. $f(x) = W W^T x$

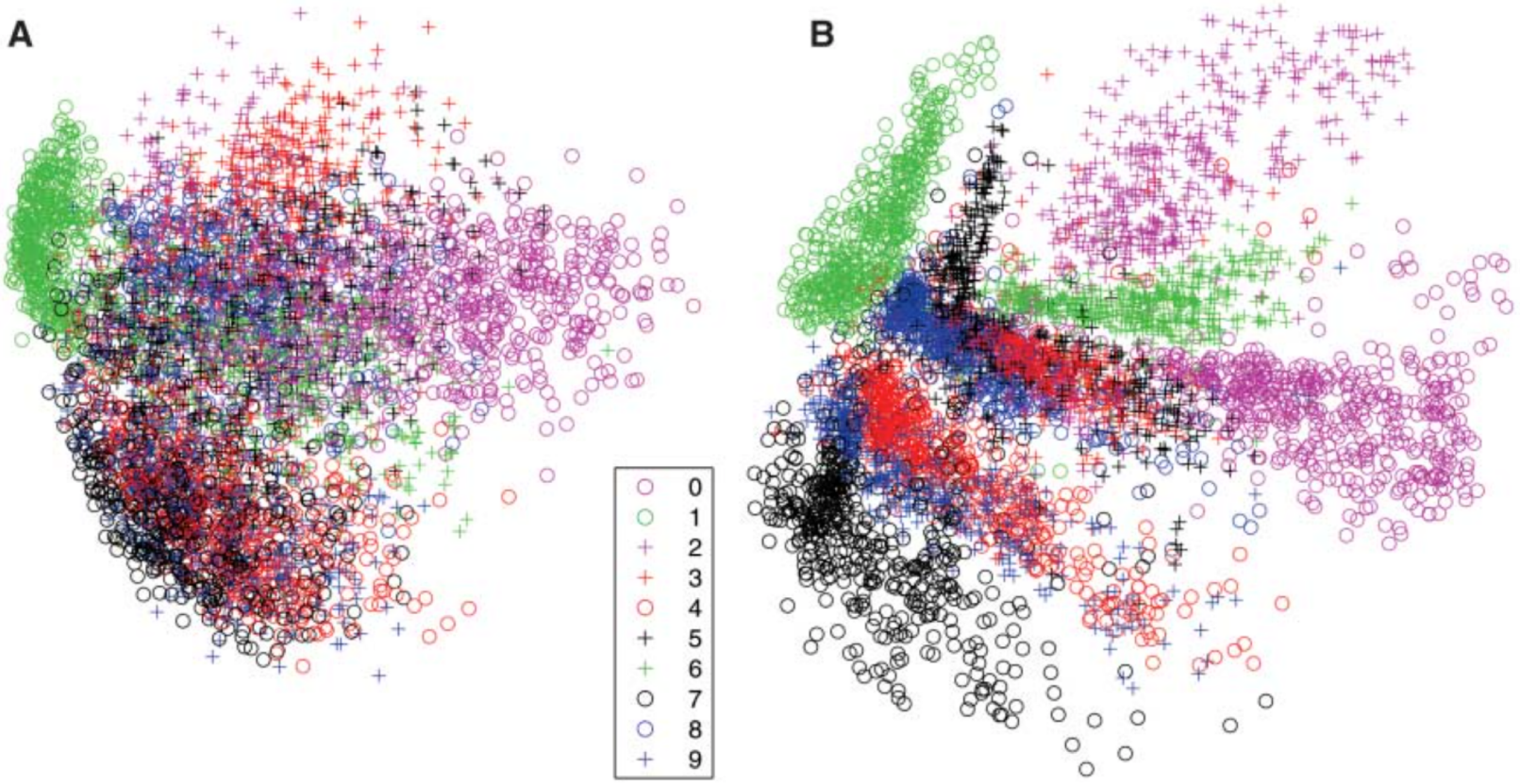
Reconstruction comparison

Top: original image
Middle: 30-dim Autoencoder
Bottom: 30-dim PCA

[Hinton & Salakhutdinov, Science'06]

Dimension reduction

Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).



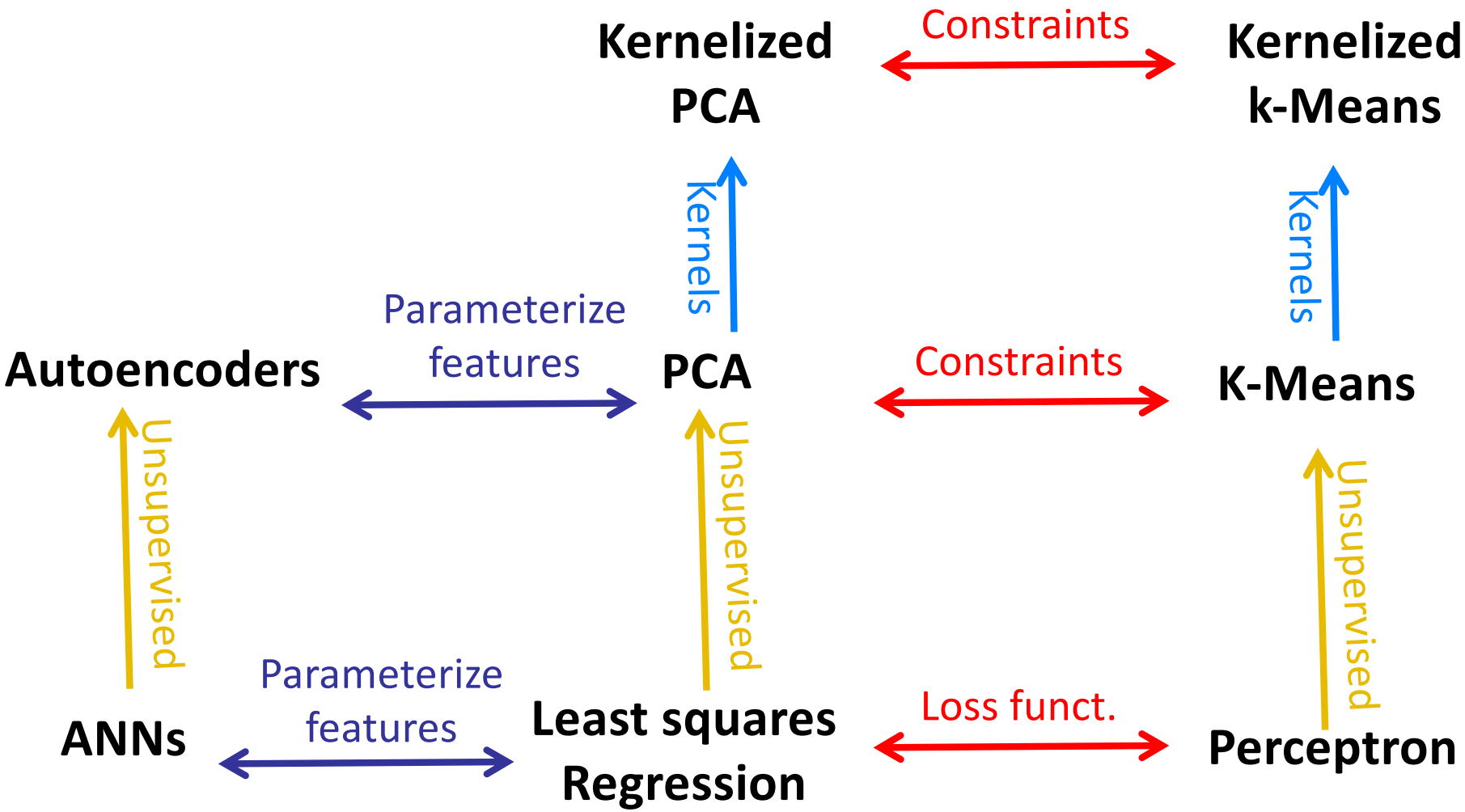
[Hinton & Salakhutdinov, Science'06]

Autoencoder Demo

What you need to be able to do

- Apply Principal Component Analysis for linear dimension reduction
- Apply kernel PCA for nonlinear dimension reduction
- Understand how to train an neural network autoencoder via backpropagation

Supervised vs. unsupervised learning



Machine learning summary so far

Representation/ features	Linear hypotheses; nonlinear hypotheses with nonlinear feature transforms, kernels, learn nonlinear features via neural nets		
Model/ objective:	Loss-function Squared loss, 0/1 loss, Perceptron loss, Hinge loss, cost sensitive losses, multi-class hinge loss, reconstruction error	+	Regularization L^2 norm, L^1 norm, early stopping, dropout
Method:	Exact solution, Gradient Descent, (mini-batch) SGD, Reductions, Lloyd's heuristic		
Evaluation metric:	Mean squared error, Accuracy, F1 score, AUC, Confusion matrices, compression performance		
Model selection:	K-fold Cross-Validation, Monte Carlo CV		