Dimensionality Reduction PCA, Autoencoders, & Other Nonlinear Methods

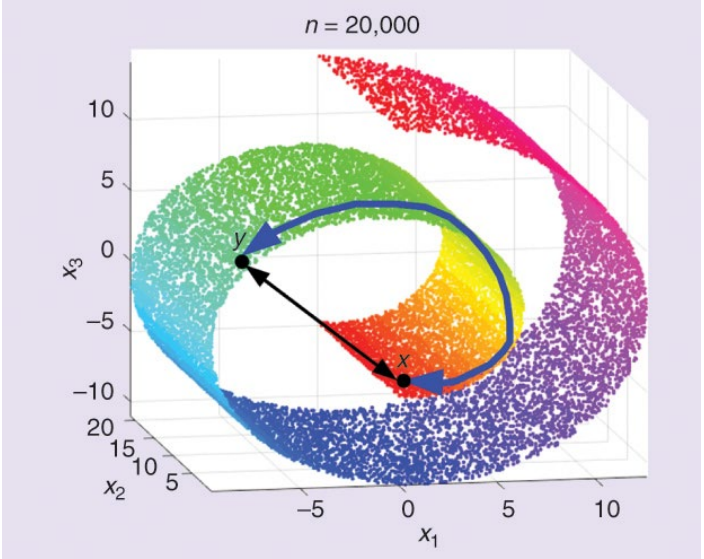
Zhiyao Duan
Associate Professor of ECE and CS
University of Rochester

Some figures are copied from the following book

- **GBC** - Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning, MIT Press.
- **LWLS** - Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, Thomas B. Schön, *Machine Learning: A First Course for Engineers and Scientists*, Cambridge University Press, 2022.

Motivation

- Data often reside on a low-dimensional subspace or manifold in the feature space
  - E.g., 64*64 grey scale images of faces has 4096 dimensions, but the intrinsic dimensionality might be just at the order of 100



(Figure from https://www.embs.org/pulse/articles/what-is-the-distance-between-objects-in-a-data-set/)

- Curse of dimensionality
  - The amount of training data required for learning increases exponentially with dimensionality
  - If N points are needed to cover one dimension, then N^d points would be needed to cover d dimensions.

- Data compression
  - Data → encoder → code → decoder → reconstructed data

Linear Autoencoder

- Let data $\boldsymbol{x} \in \mathbb{R}^p$, code $\boldsymbol{z} \in \mathbb{R}^q$, and $p > q$
- Let encoder and decoder be linear transformations
- Encoder

$$\boldsymbol{z} = \boldsymbol{W}_{q\times p}\boldsymbol{x} + \boldsymbol{b}$$

- Decoder

$$\widehat{\boldsymbol{x}} = \boldsymbol{H}_{p\times q}\boldsymbol{z} + \boldsymbol{d}$$
$$= \boldsymbol{HWx} + \boldsymbol{Hb} + \boldsymbol{d}$$

- Measure reconstruction error with L2 on dataset

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \widehat{\boldsymbol{x}}^{(i)} \right\|_2^2 = \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{HWx}^{(i)} - \boldsymbol{Hb} - \boldsymbol{d} \right\|_2^2$$

- Without loss of generality, we could set $\boldsymbol{b} = 0$, to combine biases $\boldsymbol{Hb}$ and $\boldsymbol{d}$

Linear Autoencoder

- Then we have

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)} \right\|_2^2 = \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{HW}\boldsymbol{x}^{(i)} - \boldsymbol{d} \right\|_2^2$$

- Furthermore, $\boldsymbol{d}$ can be solved as

$$\boldsymbol{d} = \frac{1}{N} \sum_{i=1}^{N} \left(\boldsymbol{x}^{(i)} - \boldsymbol{HW}\boldsymbol{x}^{(i)} \right) = (\boldsymbol{I} - \boldsymbol{HW})\bar{\boldsymbol{x}}$$

i.e., the bias $\boldsymbol{d}$ compensates for the mean of data.

- Without loss of generality, we can assume data has zero-mean, i.e., $\bar{\boldsymbol{x}} = \boldsymbol{0}$ (zero vector)
  - If data does not have zero-mean, we center data by subtracting the mean vector

$$\boldsymbol{x}^{(i)} \leftarrow \boldsymbol{x}^{(i)} - \bar{\boldsymbol{x}}, \forall i$$

Formulation for Data with Zero Mean

- From now on, we assume data $\boldsymbol{X}$ has zero mean, then we have

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)} \right\|_2^2 = \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{H}\boldsymbol{W}\boldsymbol{x}^{(i)} \right\|_2^2$$

- In matrix notation, we have data matrix $\boldsymbol{X}_{N\times p} = \begin{bmatrix} \boldsymbol{x}^{(1)^T} \\ \vdots \\ \boldsymbol{x}^{(N)^T} \end{bmatrix}$

$$E(\boldsymbol{\theta}) = \left\| \boldsymbol{X} - \hat{\boldsymbol{X}} \right\|_F^2 = \left\| \boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}^T\boldsymbol{H}^T \right\|_F^2$$

where $\boldsymbol{\theta} = \left\{ \boldsymbol{W}_{q\times p}, \boldsymbol{H}_{p\times q} \right\}$

- $rank(\boldsymbol{X}\boldsymbol{W}^T\boldsymbol{H}^T) \leq \min\{rank(\boldsymbol{X}), rank(\boldsymbol{W}), rank(\boldsymbol{H})\} \leq q$
- We try to find the best rank-q approximation $\hat{\boldsymbol{X}}$ for $\boldsymbol{X}$

Singular Value Decomposition (SVD)

- According to the Eckart-Young-Mirsky theorem, the best rank-q approximation $\widehat{\boldsymbol{X}}$ for $\boldsymbol{X}$ (zero-mean data matrix) in the sense of squared Frobenius norm is obtained by truncating the SVD to keep the q largest singular values
- SVD of $\boldsymbol{X}$ (assuming full rank)

$$\boldsymbol{X} = \boldsymbol{U}_{N\times N}\boldsymbol{\Sigma}_{N\times p}\boldsymbol{V}^T_{p\times p}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices, and $\boldsymbol{\Sigma}$ is a diagonal matrix containing singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p > 0$

- Let $\boldsymbol{\Sigma}_1$ contain the first q singular values, the $\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_2 \end{bmatrix}$
- Correspondingly, let $\boldsymbol{U} = [\boldsymbol{U}_1, \boldsymbol{U}_2]$ and $\boldsymbol{V} = [\boldsymbol{V}_1, \boldsymbol{V}_2]$

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_p \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

Solving Encoder and Decoder Matrices

- The best rank-q approximation of $\boldsymbol{X}$ is

$$\widehat{\boldsymbol{X}} = \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^T$$

- Remember our linear autoencoder

$$\widehat{\boldsymbol{X}} = \boldsymbol{X}\boldsymbol{W}^T\boldsymbol{H}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{W}^T\boldsymbol{H}^T$$

$$= [\boldsymbol{U}_1, \boldsymbol{U}_2] \begin{bmatrix} \boldsymbol{\Sigma}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_1^T \\ \boldsymbol{V}_2^T \end{bmatrix} \boldsymbol{W}^T\boldsymbol{H}^T$$

- As $\boldsymbol{V}$ is orthogonal, we have

$$\boldsymbol{W} = \boldsymbol{V}_1^T$$
$$\boldsymbol{H} = \boldsymbol{V}_1$$

- The process does not depend on q, so we can decide q after SVD!

Principal Component Analysis (PCA)

- The above procedure is called PCA
  - Center data matrix $\boldsymbol{X}_{N\times p}$
  - Perform SVD: $\boldsymbol{X} = \boldsymbol{U}_{N\times N}\boldsymbol{\Sigma}_{N\times p}\boldsymbol{V}^T_{p\times p}$
  - Compute all principal components (code): $\boldsymbol{Z}_{N\times p} = \boldsymbol{X}\boldsymbol{V}$
    - For each data point: $\boldsymbol{z} = \boldsymbol{V}^T\boldsymbol{x}$
  - Return transformation matrix: $\boldsymbol{V}^T$

- After learning the transformation matrix $\boldsymbol{V}$, we can apply it to new data $\boldsymbol{x}'$ to compute its latent code $\boldsymbol{z}' = \boldsymbol{V}^T\boldsymbol{x}'$

A New Representation

- The principal components (code), $\boldsymbol{z}$, is a new representation of input data $\boldsymbol{x}$

$$\boldsymbol{z} = \boldsymbol{V}^T \boldsymbol{x}$$

- This is a linear transformation of input data through $\boldsymbol{V}^T$
- Columns of $\boldsymbol{V}$, called principal axes, form a new orthogonal basis of the feature space $\mathbb{R}^p$
  - Columns of $\boldsymbol{V}$ are orthogonal to each other
  - Columns of $\boldsymbol{V}$ are ordered by their corresponding singular values from high to low
  - Each dimension of $\boldsymbol{z}$ is the projection of $\boldsymbol{x}$ onto the corresponding basis vector
  - Let $\boldsymbol{V}_{1_{p\times q}}$ be the first q columns of $\boldsymbol{V}_{p\times p}$, then $\boldsymbol{z}_1 = \boldsymbol{V}_1^T \boldsymbol{x}$ projects $\boldsymbol{x}$ to a subspace $\mathbb{R}^q$
  - This is the best projection in the sense that the reconstruction (decoding)

$$\hat{\boldsymbol{x}} = \boldsymbol{V}_1 \boldsymbol{z}_1$$

is the closest to input data $\boldsymbol{x}$ measured by L2 distance considering all training data

Covariance Eigenvalue Decomposition

- Look at the sample covariance matrix of training data matrix $\boldsymbol{X}$ (zero mean)

$$Cov(\boldsymbol{X}) = \frac{1}{N-1}\boldsymbol{X}^T\boldsymbol{X}$$

$$= \frac{1}{N-1}(\boldsymbol{U\Sigma V}^T)^T(\boldsymbol{U\Sigma V}^T) = \frac{1}{N-1}\boldsymbol{V\Sigma}^T\boldsymbol{U}^T\boldsymbol{U\Sigma V}^T = \boldsymbol{V}\left(\frac{1}{N-1}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\right)\boldsymbol{V}^T$$

$$= \boldsymbol{V\Lambda V}^T$$

where $\boldsymbol{\Lambda} = \mathrm{diag}\left(\frac{\sigma_1^2}{N-1}, \frac{\sigma_2^2}{N-1}, \cdots, \frac{\sigma_p^2}{N-1}\right)$

- This is the eigenvalue decomposition of $Cov(\boldsymbol{X})$
- $\boldsymbol{\Lambda}$ are the eigenvalues, ordered from high to low
- Columns of $\boldsymbol{V}$ are the corresponding eigenvectors

Variance Preserving

- Look at the sample covariance matrix of the principal components (codes) of training data, $\boldsymbol{Z}$

$$Cov(\boldsymbol{Z}) = \frac{1}{N-1}\boldsymbol{Z}^T\boldsymbol{Z}$$

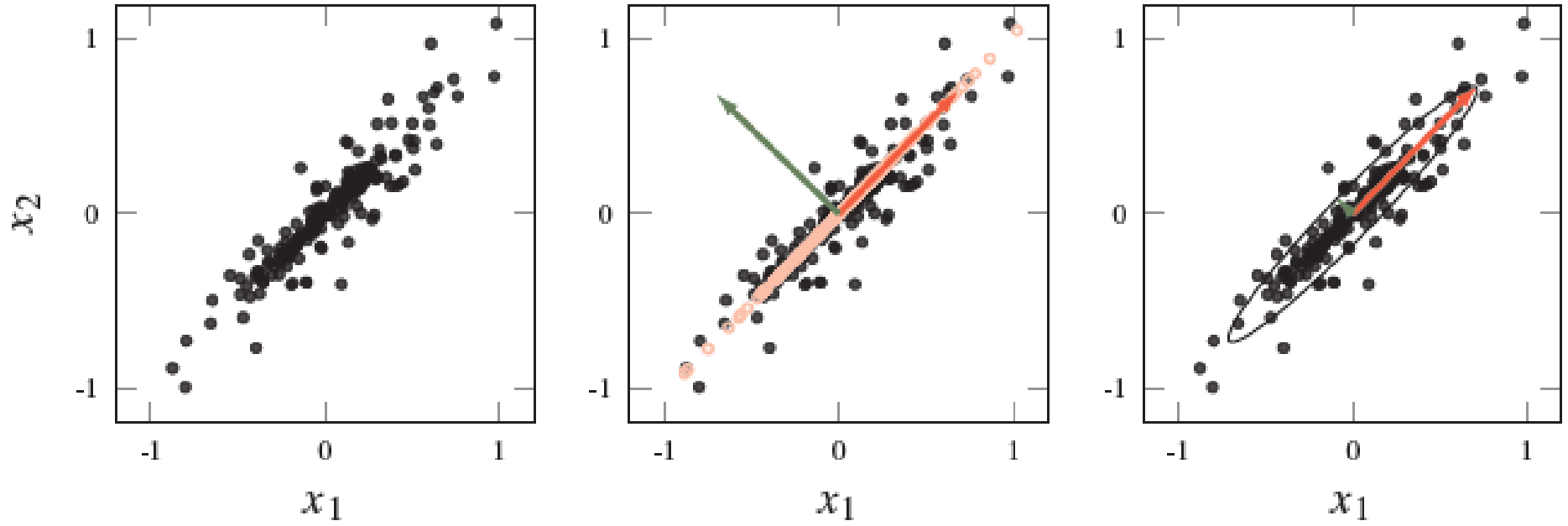
$$= \frac{1}{N-1}(\boldsymbol{XV})^T(\boldsymbol{XV}) = \frac{1}{N-1}\boldsymbol{V}^T\boldsymbol{X}^T\boldsymbol{XV} = \boldsymbol{V}^T\left(\frac{1}{N-1}\boldsymbol{X}^T\boldsymbol{X}\right)\boldsymbol{V}$$

$$= \boldsymbol{V}^T Cov(\boldsymbol{X})\boldsymbol{V} = \boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{\Lambda}$$

- This shows that different dimensions in the new representation, i.e., principal components of training data, are statistically uncorrelated from each other
- These dimensions are ordered by their variance from high to low
- The first q dimensions transformed by PCA preserve the largest data variance among all q-dimensional subspaces

PCA Illustration – 2D

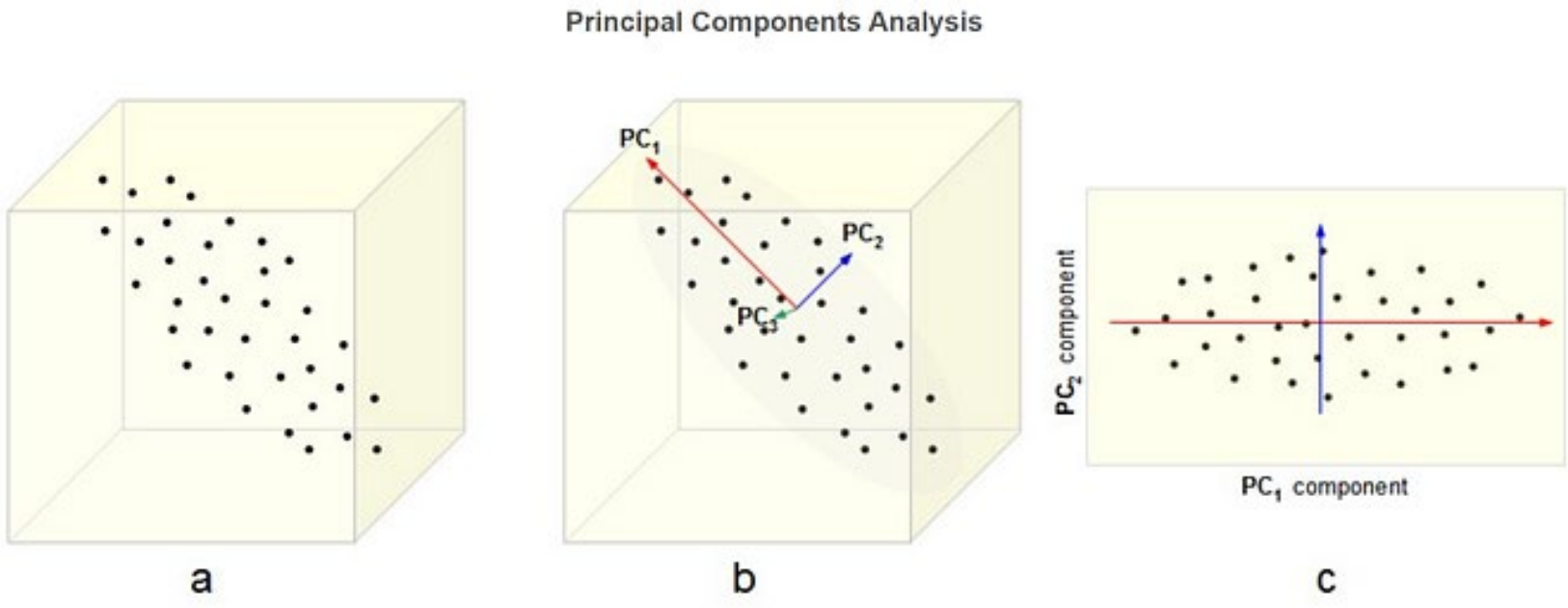
- Principal axes: red and green arrows
- Principal components: projections of data points onto principal axes
- Data variances along principal axes: shown as lengths of arrows in the right figure



(Fig. 10.11 in LWLS)

PCA Illustration – 3D

- Note: here PC actually refers to principal axis in our terminology

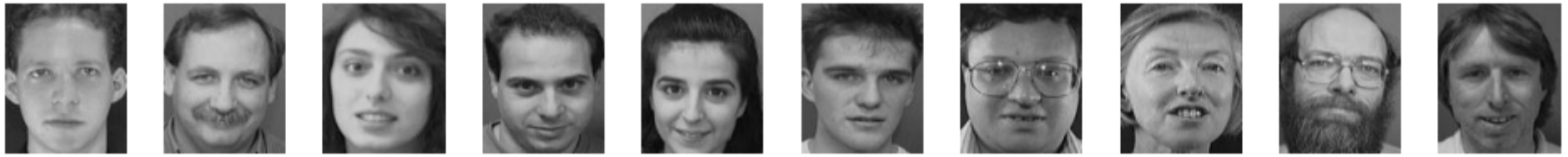


(Figure from https://medium.com/@kavita.lolayekar/the-why-behind-pca-principal-component-analysis-2f7b3fe7b7fd)

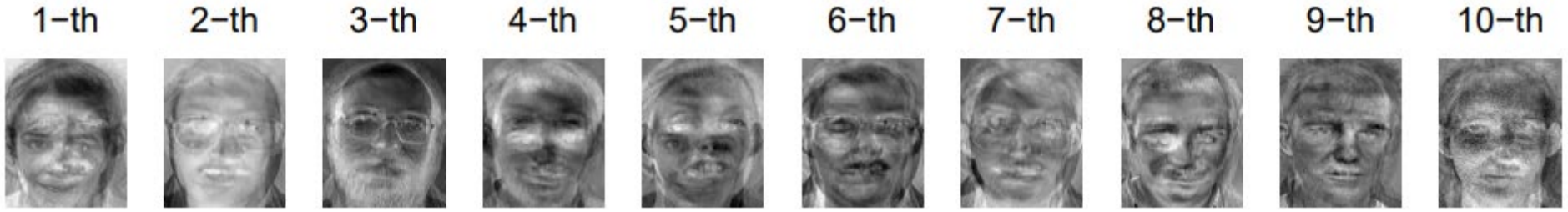
PCA on Face Images

- Perform PCA on a training set of grey-scale face images

(Figures from http://staff.ustc.edu.cn/~zwp/teach/MVA/pcaface.pdf)



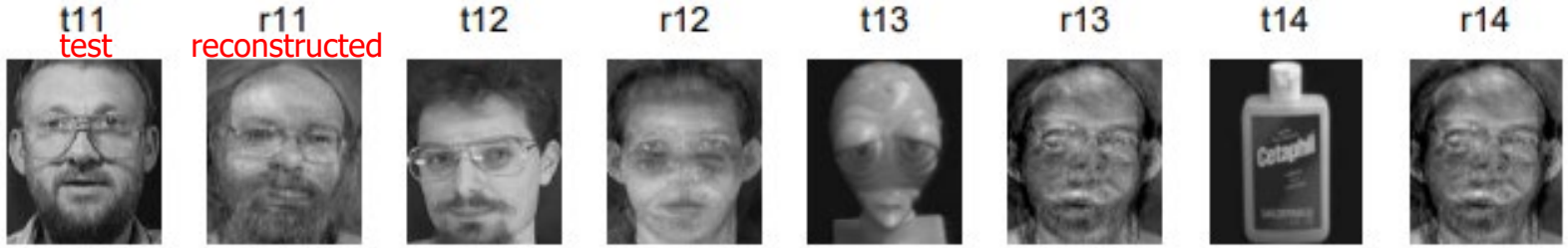
- The first ten principal axes (i.e., the first ten eigenvectors of data covariance matrix), called eigenfaces



- Reconstructed training images



- Reconstructed unseen images



Nonlinear Autoencoders

- PCA can be viewed as a linear autoencoder

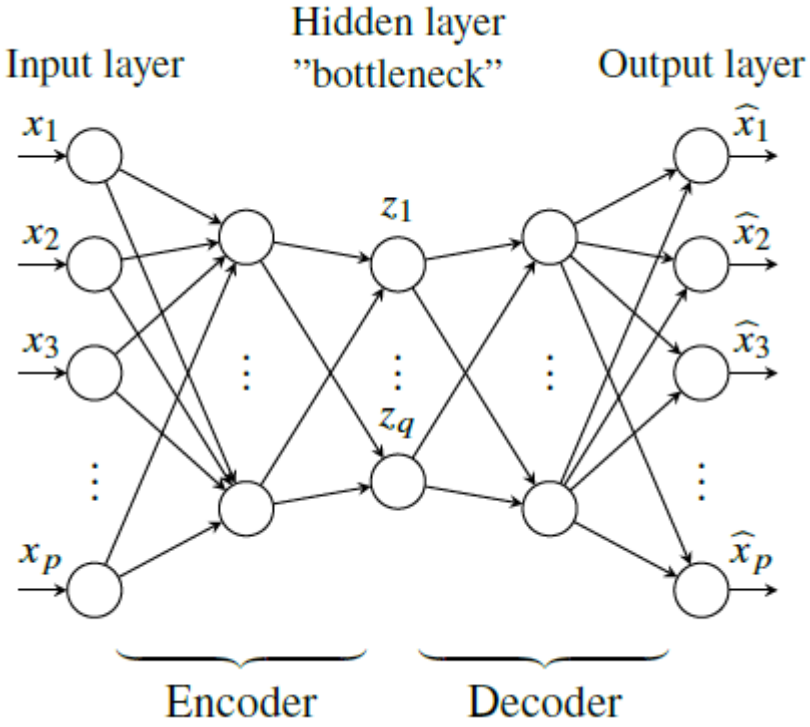
- How about using a nonlinear encoder and a nonlinear decoder?
  - A feedforward network that tries to predict the input

$$\boldsymbol{z} = f(\boldsymbol{x}), \hat{\boldsymbol{x}} = g(\boldsymbol{z})$$

  - Reconstruction loss, e.g., mean squared error

$$L_{recon} = \frac{1}{N}\sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - g\left(f\left(\boldsymbol{x}^{(i)}\right)\right) \right\|_2^2$$

  - Training with backpropagation

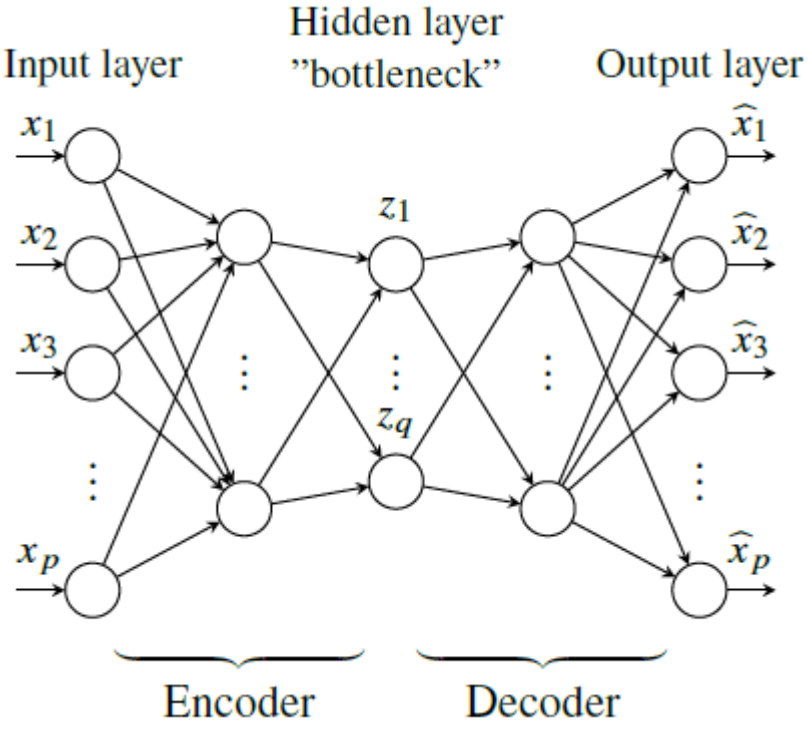


(Fig. 10.10 in LWLS)

Layer Size and Depth

- Overcomplete autoencoders
  - Hidden layer size > input size
  - Easily to learn identity map even without nonlinearity, i.e., overfitting training data
  - Needs some kind of regularization

- Undercomplete autoencoders
  - Hidden layer size < input size
  - May still learn identity map, if the nonlinearity is way too rich
  - Benefits of using more than one hidden layer



Regularizing Hidden Layer

- Regularizing hidden layer is one way to prevent from learning identity map
  - Sparsity regularization: forces the network to respond to unique statistical features in data
  - Called sparse autoencoders
    - Can be implemented by changing the loss to

$$L_{recon} + \lambda \sum_{i=1}^{N} \left\| \boldsymbol{z}^{(i)} \right\|_1$$

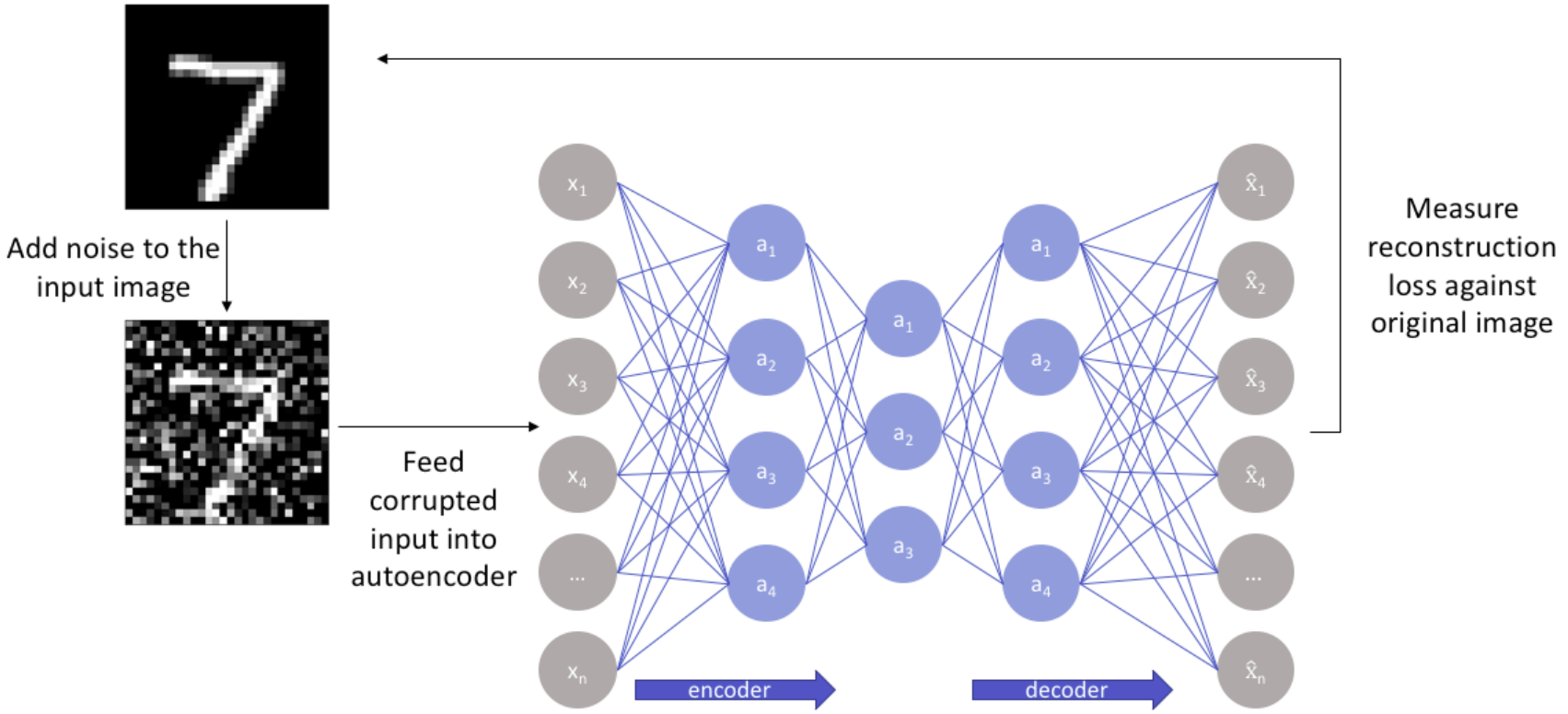
    - Can also be implemented using ReLU activation

  - Penalizing derivatives: forces the encoder to not change much when input does not
  - Called contractive autoencoders
    - Can be implemented by changing the loss to

$$L_{recon} + \lambda \sum_{i=1}^{N} \left\| \boldsymbol{\nabla}_{\boldsymbol{x}^{(i)}} \boldsymbol{z}^{(i)} \right\|_F^2$$

Denoising Autoencoders

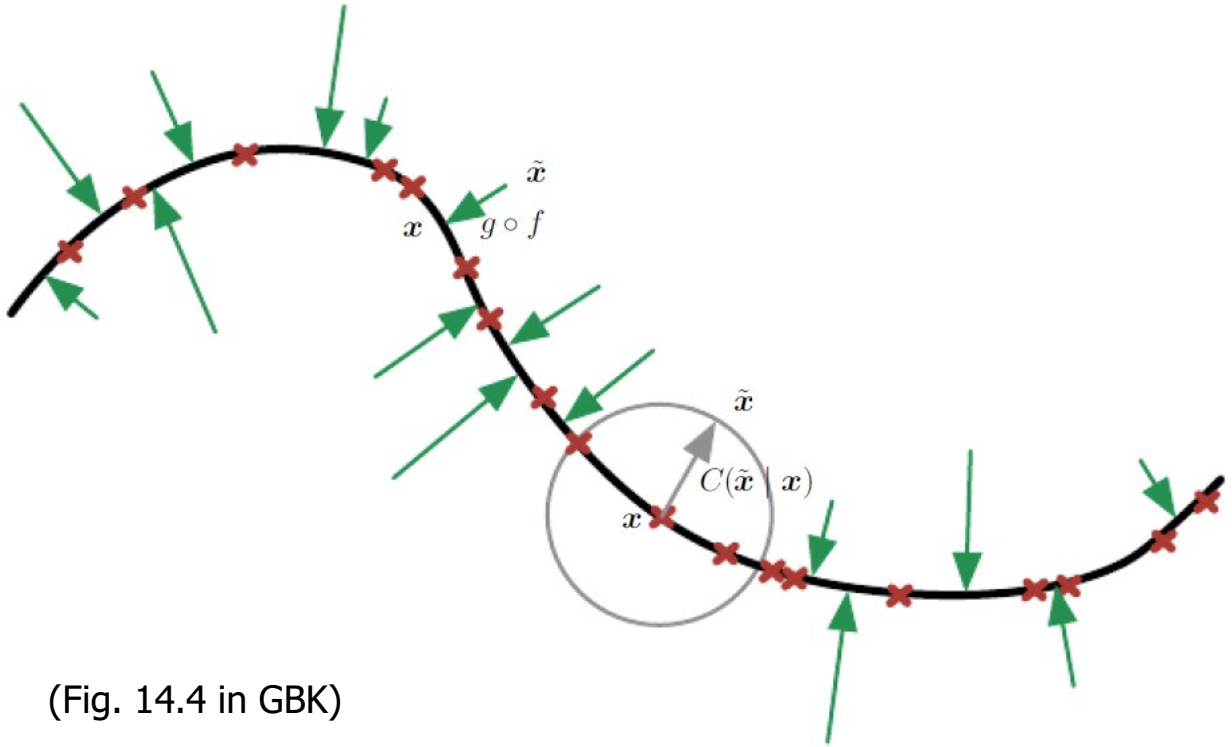
- Another way to prevent from learning identity map is to randomly corrupt the input data into $\boldsymbol{x}' = c(\boldsymbol{x})$ before feeding to the network, but still try to reconstruct the clean data $\boldsymbol{x}$:



(figure from https://www.v7labs.com/blog/autoencoders-guide#:~:text=An%20autoencoder%20is%20an%20unsupervised,even%20generation%20of%20image%20data)

Score Matching

- Score: gradient field of log probability of data $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$
- Denoising autoencoders learn a vector field $g\big(f(\tilde{\boldsymbol{x}})\big) - \boldsymbol{x}$, which is an estimate of the score around data manifold



(Fig. 14.4 in GBK)

Multidimensional Scaling (MDS)

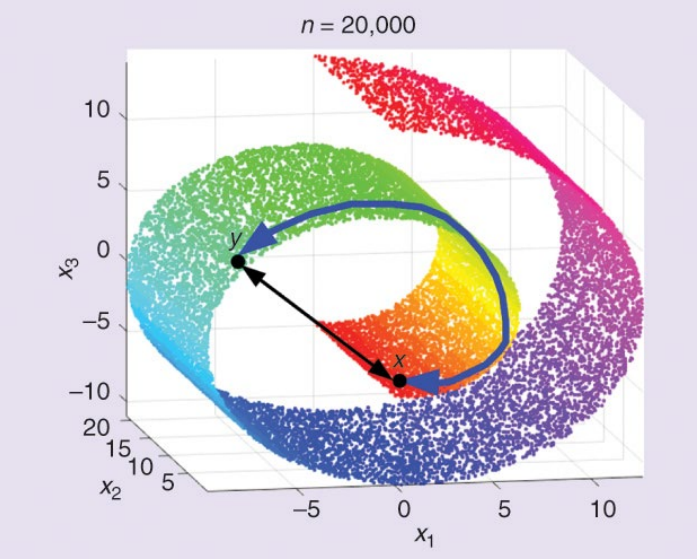
- Sometimes we only have pairwise distances or similarities between data points, and we want to embed these data points into a low-dimensional Euclidean space

	Atl	Chi	Den	Hou	LA	Mia	NYC	SF	Sea	DC
Atlanta	0									
Chicago	587	0								
Denver	1212	920	0							
Houston	701	940	879	0						
LA	1936	1745	831	1374	0					
Miami	604	1188	1726	968	2339	0				
NYC	748	713	1631	1420	2451	1092	0			
SF	2139	1858	949	1645	347	2594	2571	0		
Seattle	2182	1737	1021	1891	959	2734	2406	678	0	
DC	543	597	1494	1220	2300	923	205	2442	2329	0



Nonlinear Dimensionality Reduction

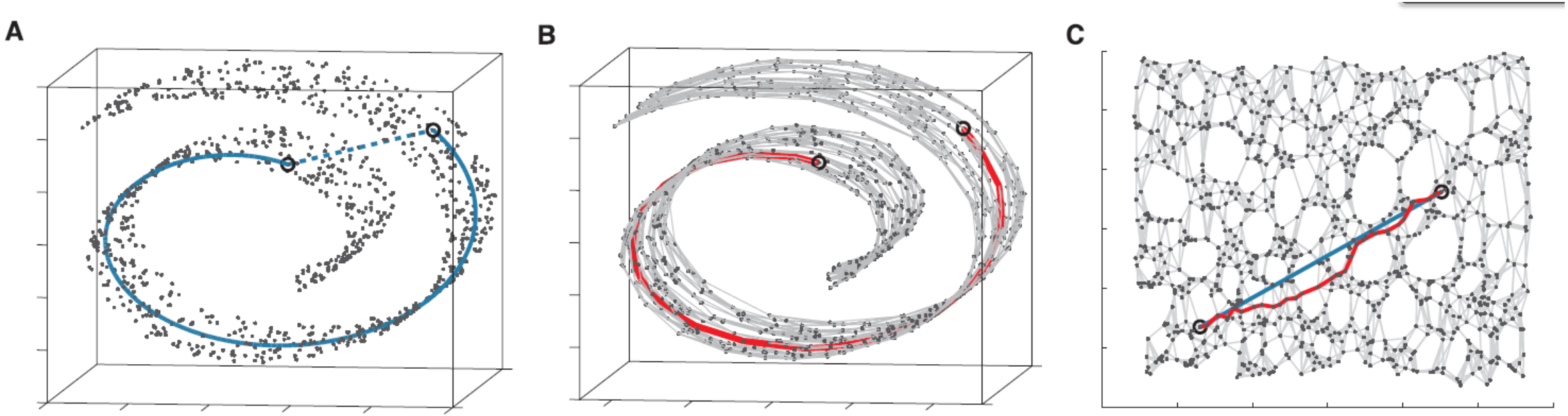
- Data points often lie on a nonlinear manifold, which cannot be captured by a linear dimensionality reduction method like PCA
  - What principal axes will PCA compute for the Swiss roll?
- Autoencoders use nonlinear activation functions to achieve nonlinear dimensionality reduction
  - With the objective of data reconstruction
- Other ideas?



(Figure from https://www.embs.org/pulse/articles/what-is-the-distance-between-objects-in-a-data-set/)

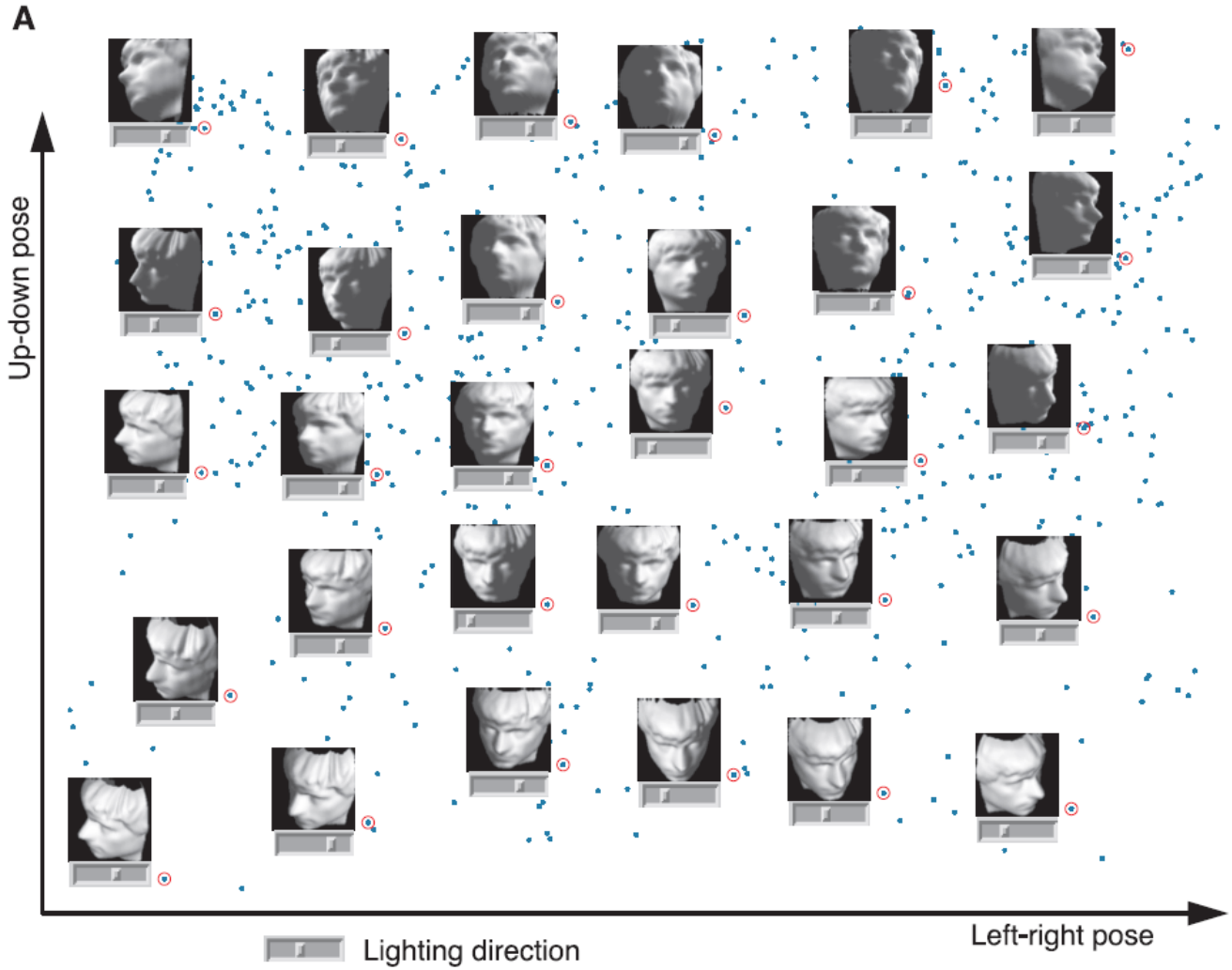
Isometric Feature Mapping (Isomap)

- Key idea: find a Euclidean embedding that preserves the pairwise geodesic distances
  - Step 1: construct neighborhood graph, ϵ-neighborhood or K-nearest-neighborhood
  - Step 2: compute shortest paths, between all pairs of data points
  - Step 3: construct d-dimensional embedding, through multidimensional scaling (MDS)

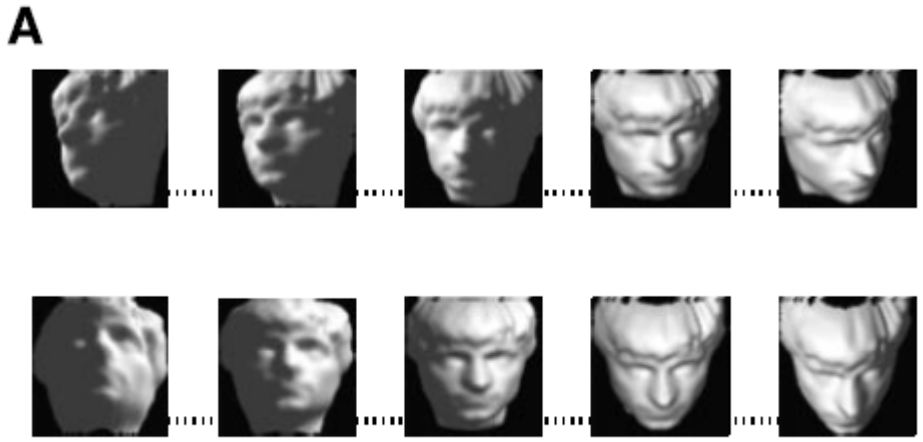


(Fig. 3 in [Tenenbaum et al., *Science*, 2000])

Isomap Examples

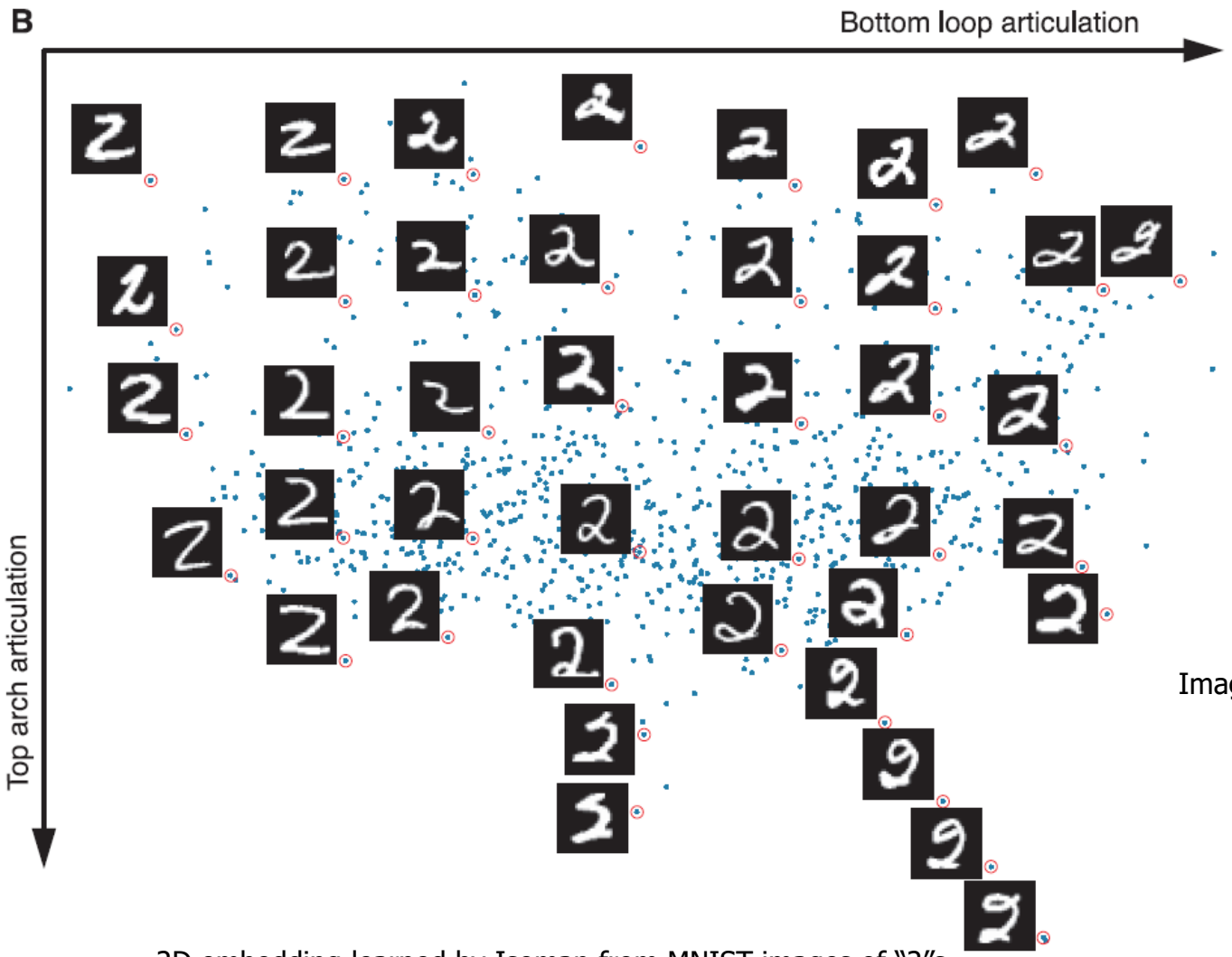


3D embedding learned by Isomap from 64*64 face images with different poses and lighting directions. (Fig. 1A in [Tenenbaum et al., *Science*, 2000])



Images along straight lines in the embedding space connecting end images (Fig. 4A in [Tenenbaum et al., *Science*, 2000])

Isomap Examples



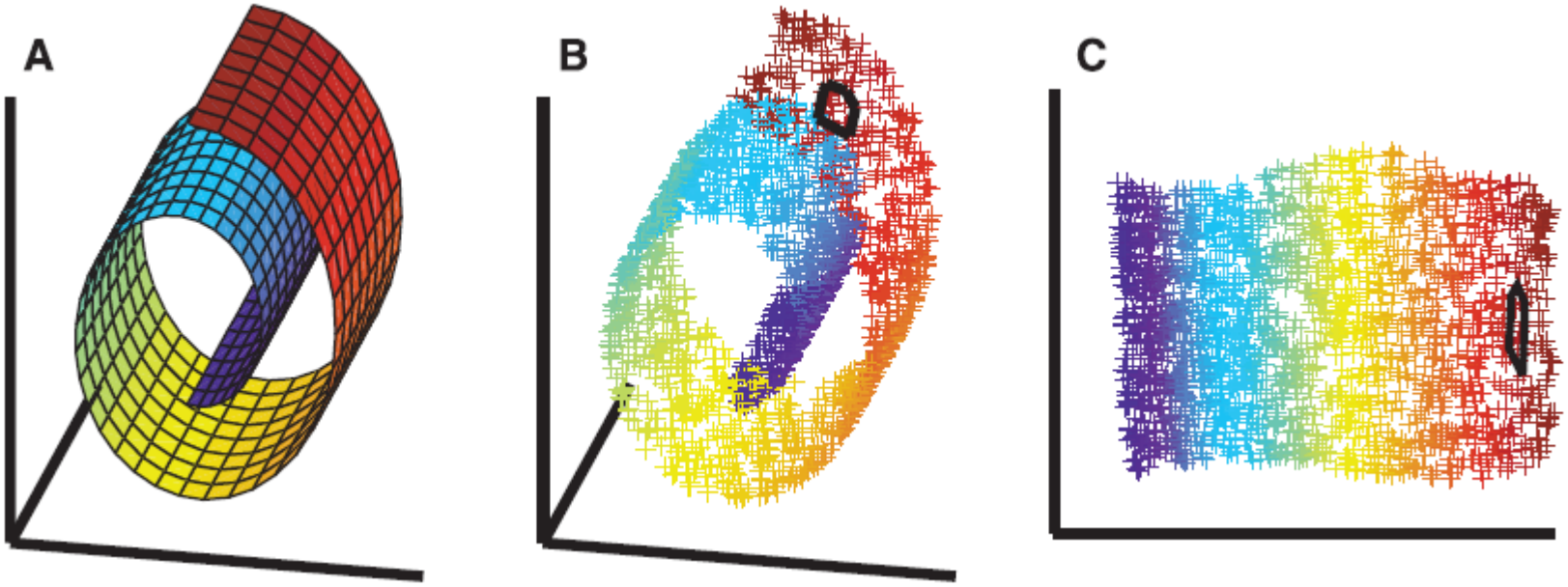
2D embedding learned by Isomap from MNIST images of "2"s
(Fig. 1B in [Tenenbaum et al., *Science*, 2000])



Images along straight lines in the embedding space connecting end images
(Fig. 4C in [Tenenbaum et al., *Science*, 2000])

Locally Linear Embedding (LLE)

- Key idea: learn a Euclidean embedding that preserves locally linear relations among data points



(Fig. 1 in [Roweis & Saul, *Science*, 2000])

LLE Algorithm

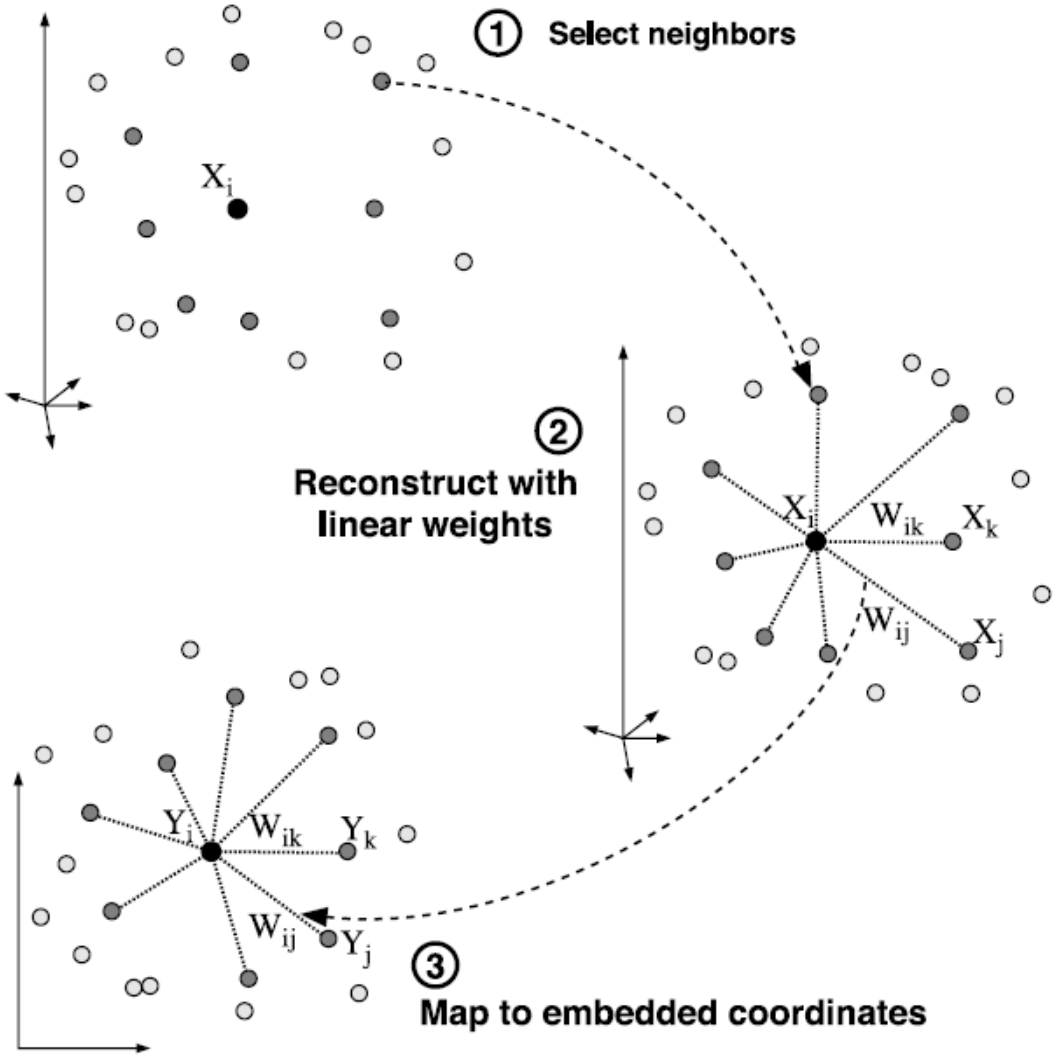
- Step 1: construct neighborhood graph (e.g., K-nearest-neighborhood)

- Step 2: compute linear weights W_{ij} that best linearly reconstruct a point X_i from its neighbors, i.e., minimizing

$$\varepsilon(W) = \sum_{\mathrm{i}} \left| \vec{X}_{\mathrm{i}} - \Sigma_{\mathrm{j}} W_{\mathrm{ij}} \vec{X}_{\mathrm{j}} \right|^2$$

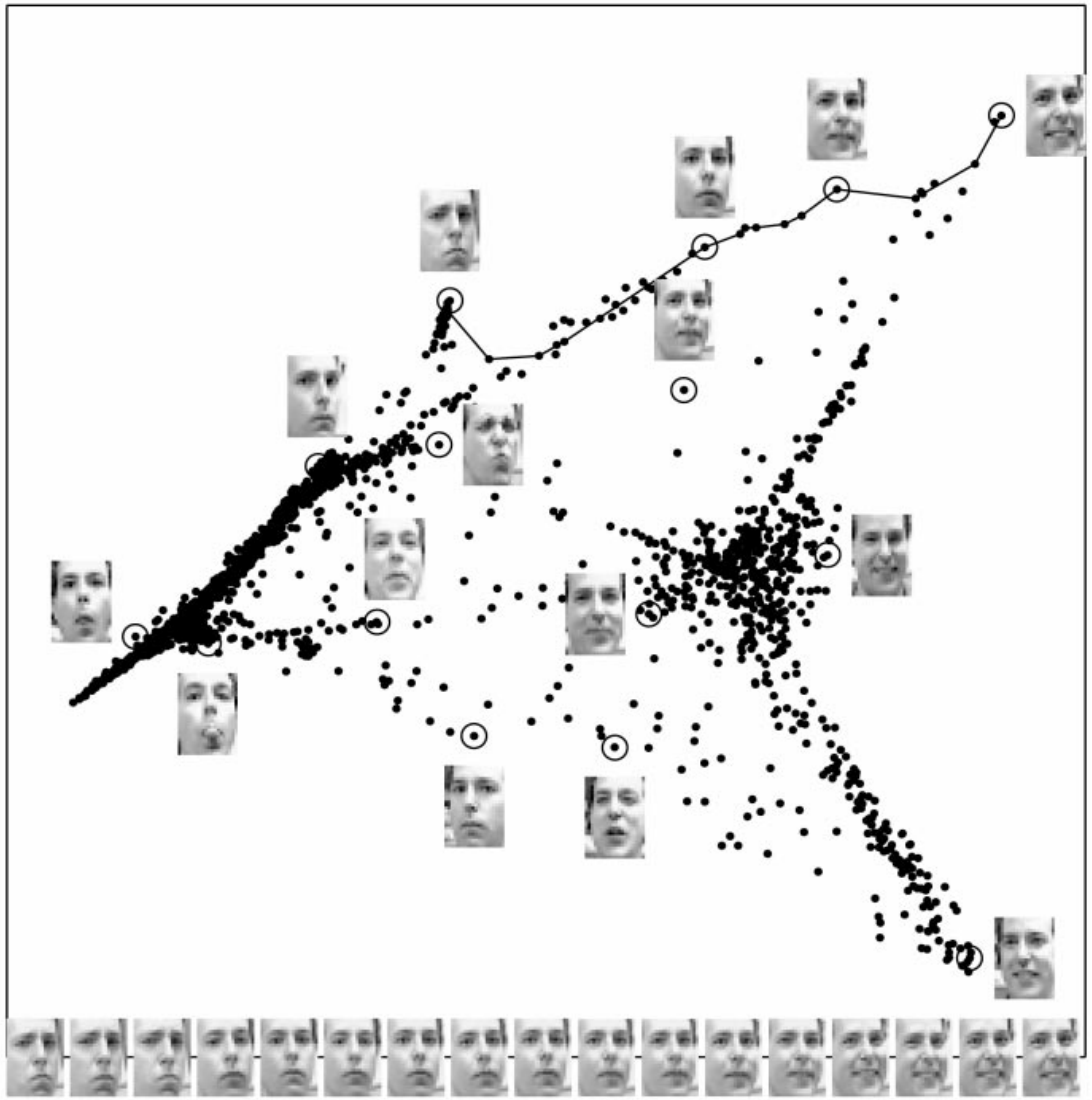
- Step 3: Compute the low-dimensional embedding vectors Y_i best reconstructed by W_{ij}, i.e., minimizing

$$\Phi(Y) = \sum_{\mathrm{i}} \left| \vec{Y}_{\mathrm{i}} - \Sigma_{\mathrm{j}} W_{\mathrm{ij}} \vec{Y}_{\mathrm{j}} \right|^2$$



(Fig. 2 in [Roweis & Saul, *Science*, 2000])

LLE Examples



(Fig. 3 in [Roweis & Saul, *Science*, 2000])

Summary

- Why dimensionality reduction?
  - Intrinsic dimension << feature dimension
  - Curse of dimensionality
  - Data compression
- PCA: linear, minimizes reconstruction error, preserves large data variances
  - SVD of zero-mean data matrix
- MDS: linear, preserves pairwise distances
- Autoencoder: nonlinear, minimizes reconstruction error
- Isomap: nonlinear, preserves geodesic distances
- LLE: nonlinear, preserves local linear relations
- More advanced methods, commonly used for data visualization
  - t-SNE [van der Maaten & Hinton, 2008]
  - UMAP [McInnes & Healy, 2018]