CS6208 : Advanced Topics in Artificial Intelligence
Graph Machine Learning

Lecture 6 : Graph-based Visualization

Semester 2 2022/23

Xavier Bresson

https://twitter.com/xbresson

Department of Computer Science
National University of Singapore (NUS)

Course lectures

- Introduction to Graph Machine Learning
- Part 1: GML without feature learning (before 2014)
  - Introduction to Graph Science
  - Graph Analysis Techniques without Feature Learning
    - Graph clustering
    - Graph SVM
    - Recommendation on graphs
    - Graph-based visualization
- Part 2 : GML with shallow feature learning (2014-2016)
  - Shallow graph feature learning
- Part 3 : GML with deep feature learning, a.k.a. GNNs (after 2016)
  - Graph Convolutional Networks (spectral and spatial)
  - Weisfeiler-Lehman GNNs
  - Graph Transformer & Graph ViT/MLP-Mixer
  - Benchmarking GNNs
  - Molecular science and generative GNNs
  - GNNs for combinatorial optimization
  - GNNs for recommendation
  - GNNs for knowledge graphs
  - Integrating GNNs and LLMs

Outline

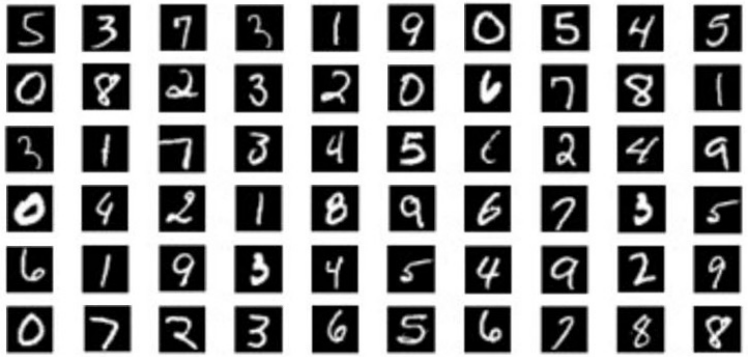
- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- Non-linear visualization techniques
  - LLE
  - Laplacian eigenmaps
  - TSNE
  - UMAP
- Conclusion

Outline

- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- Non-linear visualization techniques
  - LLE
  - Laplacian eigenmaps
  - TSNE
  - UMAP
- Conclusion

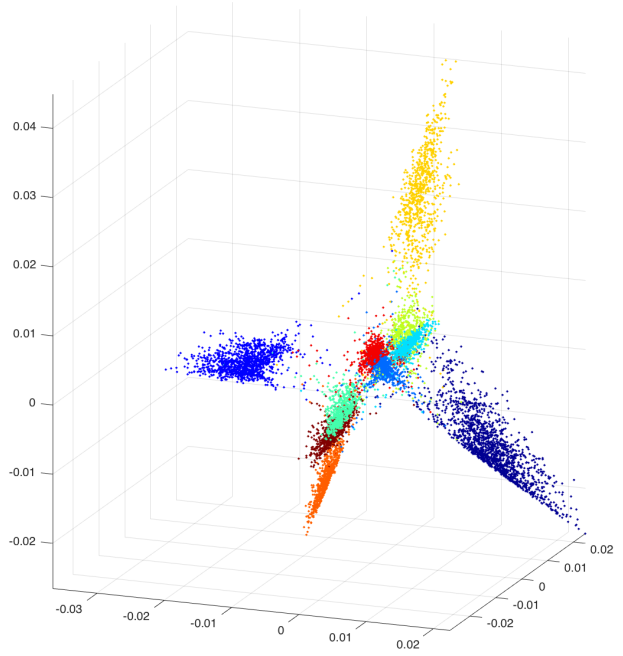
Visualization

- The visualization task involves projecting high-dimensional data, s.a. images, text documents, user/product attributes, sequences of actions, etc into 2D or 3D low-dimensional Euclidean spaces to reveal underlying data structures.
- This projection is achieved using dimensionality reduction techniques, which aim to compress the original information while discarding unnecessary details and noise.



28 x 28 MNIST images

$\Rightarrow$



Visualization of MNIST images in R^3

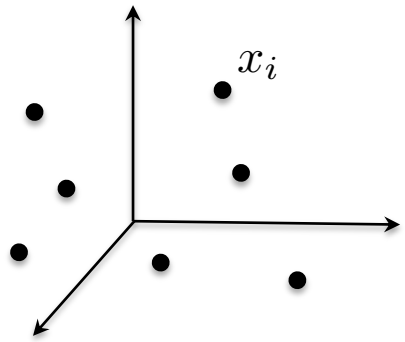
Dimentionality reduction

- Two classes of dimensionality reduction techniques have been developed :
  - Linear Techniques: These methods produce low-dimensional Euclidean (flat) spaces.
  - Common examples include Principal Component Analysis (PCA)[1], Linear Discriminant Analysis (LDA)[2], and Independent Component Analysis (ICA)[3].
  - Non-Linear Techniques: These methods compute low-dimensional manifolds, i.e. curved hyper-surfaces.
  - Standard techniques are Kernel methods[4], Locally Linear Embedding (LLE)[5], Laplacian Eigenmaps[6], t-distributed Stochastic Neighbor Embedding (TSNE)[7], and Uniform Manifold Approximation and Projection (UMAP)[8].

[1] Pearson, On lines and planes of closest fit to systems of points in space, 1901
[2] Fisher, The Use of Multiple Measurements in Taxonomic Problems, 1936
[3] Herault, Jutten, Architectures neuromimétiques adaptatives: Détection de primitives, 1985
[4] Scholkopf et-al, Nonlinear Component Analysis as a Kernel Eigenvalue Problem, 1998
[5] Roweis, Saul, Nonlinear dimensionality reduction by locally linear embedding, 2000
[6] Belkin, Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, 2003
[7] Van der Maaten, Hinton, Visualizing data using t-SNE, 2008
[8] McInnes et-al, UMAP: Uniform manifold approximation and projection for dimension reduction, 2018

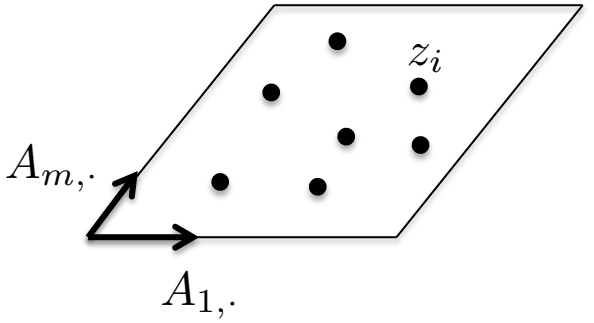
Linear dimensionality reduction

- **Assumption: The data distribution exists within a low-dimensional Euclidean space.**



Linear Dimensionality Reduction

$\Rightarrow$



High-dimensional Euclidean space

$x_i \in \mathbb{R}^d, d \gg 1$

Projection map

$\varphi : x_i \to z_i = \varphi(x_i) = Ax_i$

Low-dimensional hyper-plane

$z_i \in \mathbb{R}^m, m \ll d$

Linear techniques

- Task formalization : Restrict the mapping φ to be a linear operator A.
- Several techniques exist to compute a linear operator A.
- PCA, LDA, ICA, Non-negative matrix factorization[1] (NMF), Sparse Coding[2], etc.

$$z = \varphi(x) = Ax = \begin{bmatrix} \langle A_{1,.}, x \rangle \\ \vdots \\ \langle A_{k,.}, x \rangle \end{bmatrix} = \begin{bmatrix} z_1 \\ \vdots \\ z_k \end{bmatrix}$$

with

$x \in \mathbb{R}^d, d \gg 1$, high-dimensional data point

$z \in \mathbb{R}^m, m \ll d$, low-dimensional data point

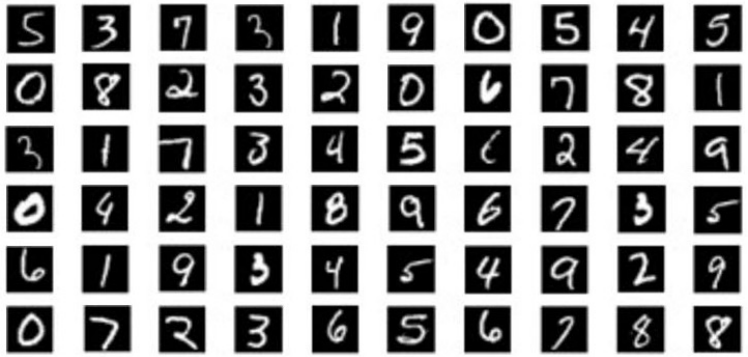
$A \in \mathbb{R}^{m \times n}$, dictionary of patterns or basis functions

$A_{i,.} \in \mathbb{R}^n$, i-th pattern/linear filter

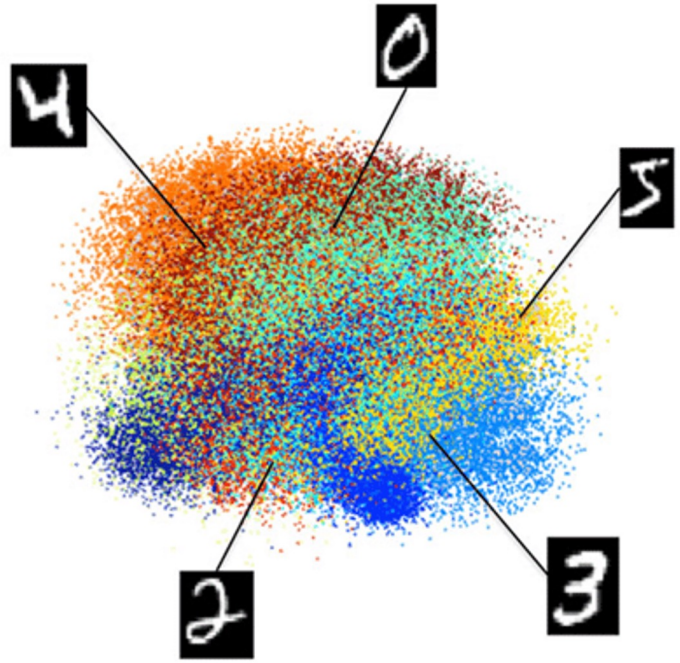
[1] Lee, Seung, Learning the parts of objects by non-negative matrix factorization, 1999
[2] Olshausen, Field, Learning a sparse code for natural images, 1996

Linear dimensionality reduction

- An example where linear dimensionality reduction falls short in producing clear patterns.
- This highlights the need for greater expressivity to uncover the underlying structures.



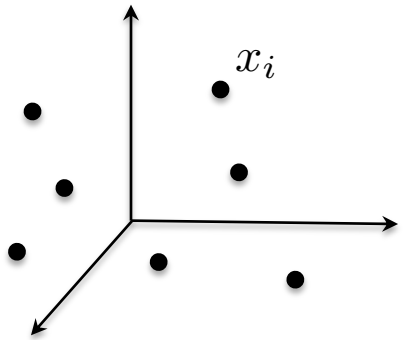
28 x 28 MNIST images



Visualization of MNIST images in $\mathbb{R}^3$ with PCA

Non-linear dimensionality reduction

- **Assumption: Data distribution resides on low-dimensional curved spaces, known as manifolds (which can be smooth or non-smooth).**
- **Techniques designed to uncover these structures are referred to as manifold learning.**



High-dimensional Euclidean space

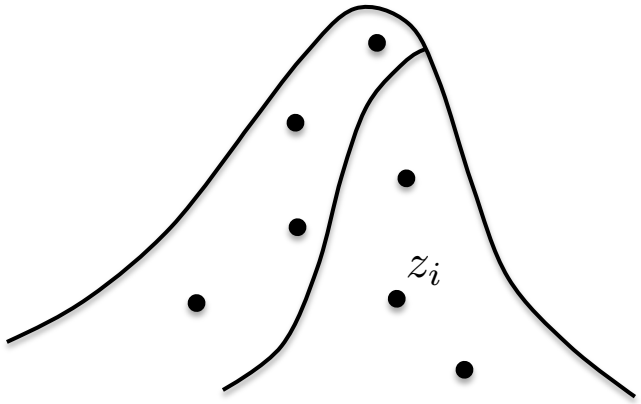
$x_i \in \mathbb{R}^d, d \gg 1$

Non-Linear Dimensionality Reduction

$\Rightarrow$

Projection map

$\varphi : x_i \to z_i = \varphi(x_i)$

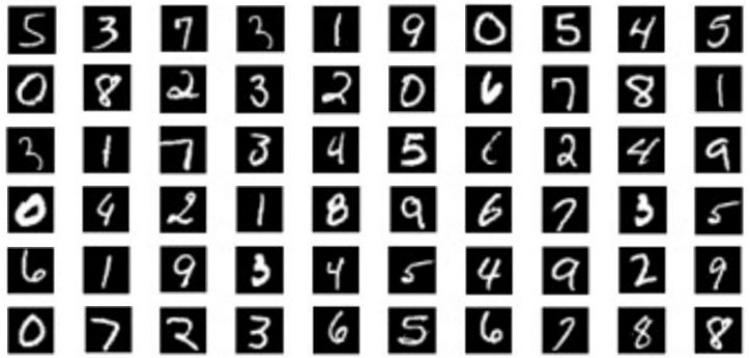


Low-dimensional manifold

$\mathcal{M} \subset \mathbb{R}^d,\ \dim(\mathcal{M}) = m,\ m \ll d$

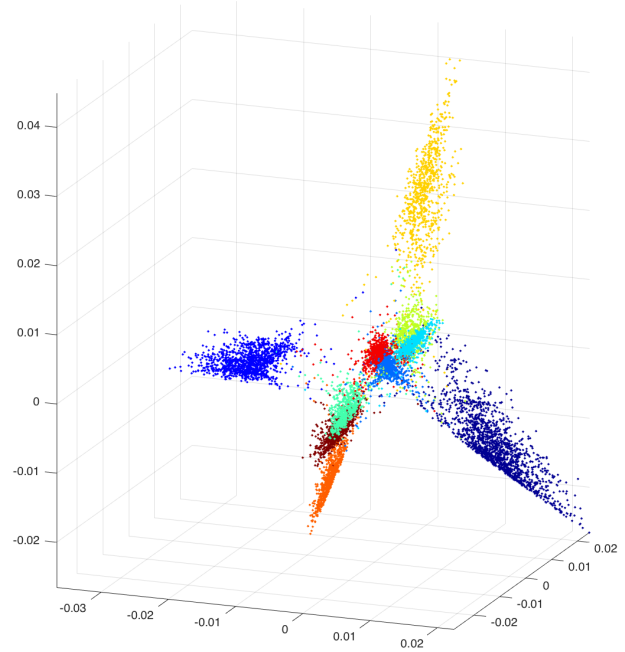
Dimensionality reduction

- An example where non-linear reduction effectively reveals clear patterns.
- Several non-linear techniques are available, each suited to different data distributions.



28 x 28 MNIST images
$x \in \mathbb{R}^{28 \times 28 = 684}$

$$\varphi(x) = z$$
$$x \in \mathbb{R}^{684}, \; z \in \mathbb{R}^3$$
$$\varphi = \text{LapEigenmaps}^{[1]}$$
$$\Rightarrow$$



Visualization of MNIST images in 3D
$z \in \mathbb{R}^3$

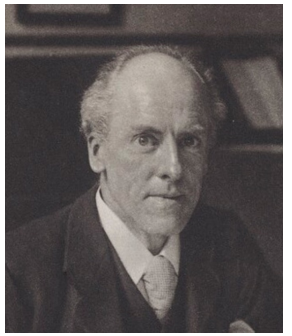
[1] Belkin, Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, 2003

Outline

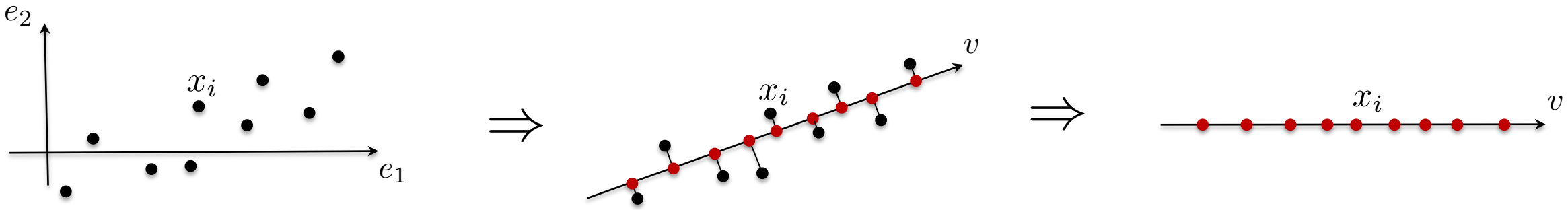
- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- Non-linear visualization techniques
  - LLE
  - Laplacian eigenmaps
  - TSNE
  - UMAP
- Conclusion

Principal component analysis

- PCA[1], introduced in 1901, is inspired by the principal axis theorem in mechanics.
- It is the most popular technique for linear dimensionality reduction.
- It aims to capture the direction of greatest variance within the data distribution.



Karl Pearson
1857-1936



Original data distribution in R^2

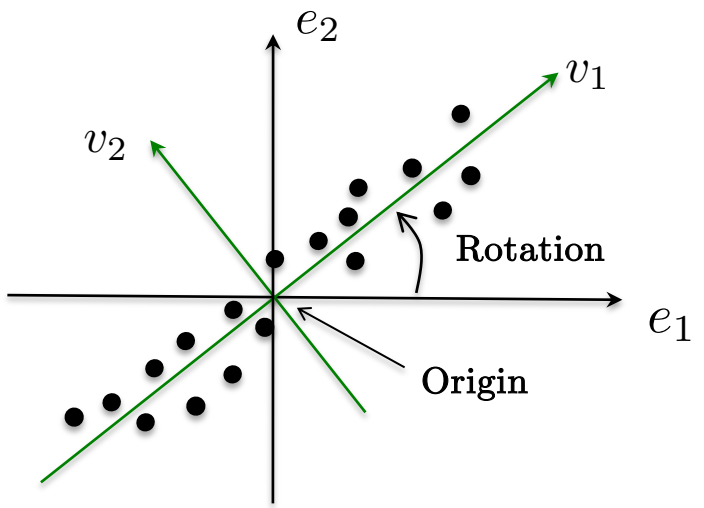
Projection of data points into the direction of the largest variation v of the distribution

Principal component v and approximation of the original distribution w.r.t. the largest variance

[1] Pearson, On lines and planes of closest fit to systems of points in space, 1901

Task formulation

- Given a set of data points, PCA projects the data onto an orthogonal basis that best captures its variance.
- Assuming the data distribution is centered at the origin, PCA defines an orthogonal transformation, i.e. a rotation matrix, that maps the data to a new coordinate system $(v_1,v_2,...,v_K)$ known as principal directions such that
  - The first basis function or principal direction v_1 captures the largest possible variance in data.
  - The second basis function or principal direction v_2 captures the second largest possible variance while being orthogonal to the first principal direction $\langle v_1,v_2\rangle=0$.
  - For each subsequent direction, the v_k's capture the k-th largest possible data variance, maintaining orthogonality to all previous directions.



Covariance matrix

- **Data variance across feature dimensions is captured by the covariance matrix :**

$$C = X^T X \in \mathbb{R}^{d \times d},$$

with data matrix $X \in \mathbb{R}^{n \times d}$ where

n is the number of data points

d is the number of data features

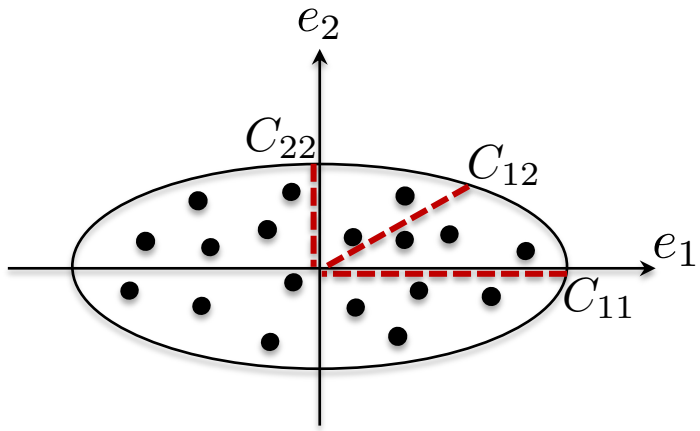
Then we have

$$C_{11} = X_{\cdot,1}^T X_{\cdot,1} = \|X_{\cdot,1}\|_2^2 = \sum_{i=1}^{n} X_{i1}^2 \text{ variance in the direction } e_1$$

$$C_{12} = X_{\cdot,1}^T X_{\cdot,2} = \sum_{i=1}^{n} X_{i1} X_{i2} \text{ cross-variance in the direction } e_1\text{-}e_2$$

Reminder : Data is centered in each feature dimension j, i.e.

$$\mathbb{E}(X_{\cdot,j}) = \sum_{i=1}^{n} X_{ij} = 0, \ \forall j \in \{1, ..., d\}$$



Vector e_1 is the direction of the largest variance with value C_{11}

Direction of the largest variation

- **Consider an arbitrary centered data distribution.**
- **Let us compute the direction of the largest data variance, $\mathbf{v}_{\text{largest}}$:**

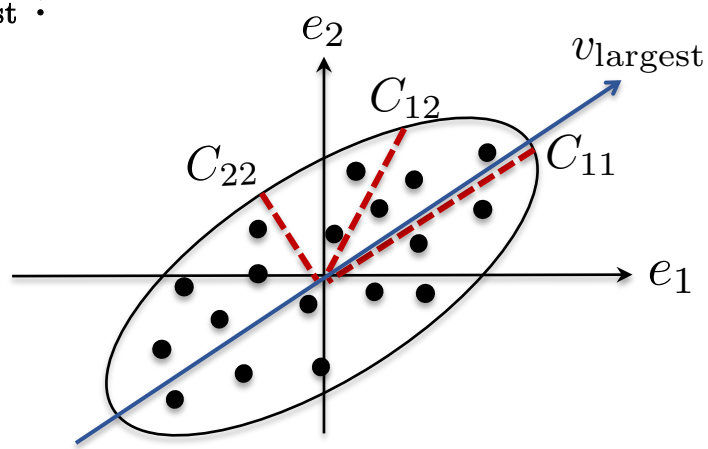
$$v_{\text{largest}} \in \mathbb{R}^d = \text{argmax}_{\|v\|_2=1} \sum_{i=1}^{n} \left(x_i^T v\right)^2, \; x_i \in \mathbb{R}^d$$

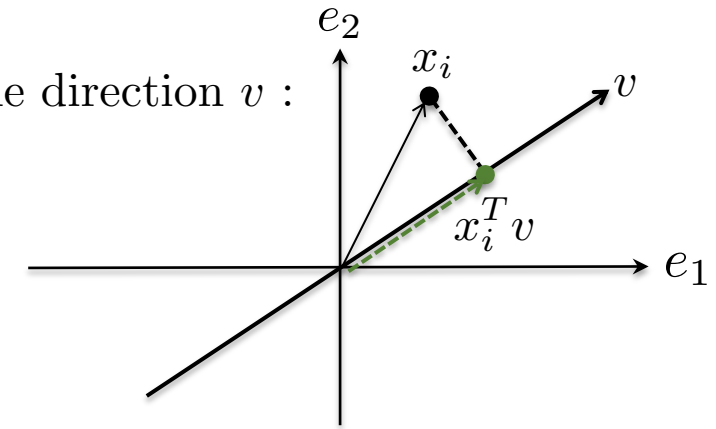
$$= \text{argmax}_{\|v\|_2=1} \; v^T C v$$

given that $\sum_i \left(x_i^T v\right)^2 = \|Xv\|_2^2 = (Xv)^T(Xv) = v^T X^T X v$

and by definition $C = X^T X$

$x_i^T v$ is the projection of x_i on the direction v :





Eigenvalue decomposition

- **Next, we perform the eigenvalue decomposition (EVD) of the positive semi-definite (PSD) covariance matrix C :**

$Cv_j = \lambda_j v_j \in \mathbb{R}^d, \ j = 1, ..., d$

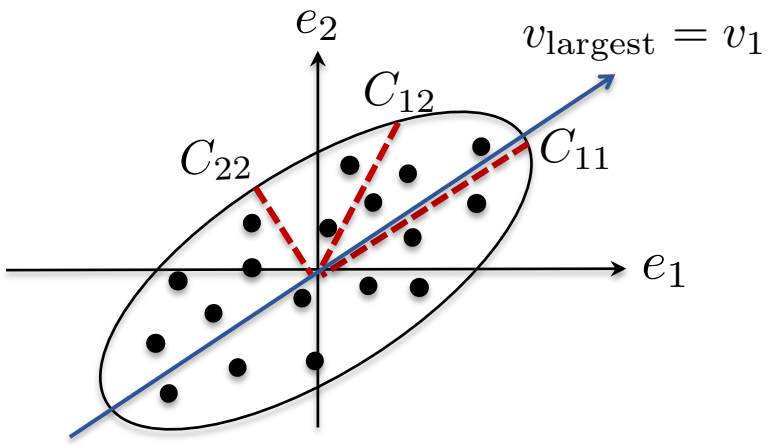
with the eigenvalues: $\lambda_{\max} = \lambda_1 \geq \lambda_2 \geq ... \geq \lambda_d = \lambda_{\min} \geq 0$

Let us consider the largest eigenvalue, we have

$v_{\max}^T C v_{\max} = \lambda_{\max} v_{\max}^T v_{\max} = \lambda_{\max} \|v_{\max}\|_2^2 = \lambda_{\max} \geq \lambda_j, \ \forall j \neq 1$

In other words, we have $v_{\text{largest}} = v_{\max} = v_1$ as

$\text{argmax}_{\|v\|_2=1} \ v^T C v = v_{\max}^T C v_{\max} = \lambda_{\max} = \lambda_1$



Direction of the second largest variation

- **The direction of the greatest data variance, known as the first Principal Direction (PD), is given by the spectral solution and corresponds to the eigenvector $\mathbf{v}_1$ associated with the largest eigenvalue of the covariance matrix C :**

$$Cv_1 = \lambda_1 v_1 \;\rightarrow\; v_1^T C v_1 = \lambda_1 v_1^T v_1 = \lambda_1 \|v_1\|_2^2 = \lambda_1 = \text{argmax}_{\|v\|_2=1} \sum_{i=1}^{n} \left(x_i^T v\right)^2$$

- **Similarly, the direction of the second largest data variance, or the second PD, is defined as :**

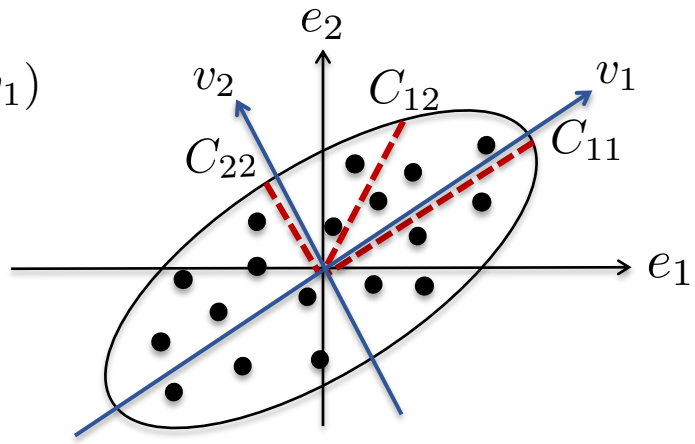
$$v_2 \in \mathbb{R}^d = \text{argmax}_{\|v\|_2=1} \sum_{i=1}^{n} \left(x_i^T v\right)^2, \quad \text{s.t. } v^T v_1 = 0 \ (v \text{ is orthogonal to } v_1)$$

and the solution is given by the second eigenvalue and its eigenvector:

$v_2^T C v_2 = \lambda_2 v_2^T v_2 = \lambda_2 \|v_2\|_2^2 = \lambda_2 \geq \lambda_j, \ \forall j \geq 3$ and $\lambda_2 \leq \lambda_1$

In other words, we have

$\text{argmax}_{\|v\|_2=1, v^T v_1} \ v^T C v = v_2^T C v_2 = \lambda_2$ (second largest variance)



PCA as EVD of covariance matrix

- **In the same way, the direction of the third largest data variance is defined as**

$$v_3 \in \mathbb{R}^d = \text{argmax}_{\|v\|_2=1} \sum_{i=1}^{n} \left(x_i^T v\right)^2, \quad \text{s.t. } v^T v_1 = 0 \text{ and } v^T v_2 = 0$$

(v is orthogonal to v_1 and v_2)

The solution is given by the third eigenvalue and its eigenvector:

$Cv_3 = \lambda_3 v_3$

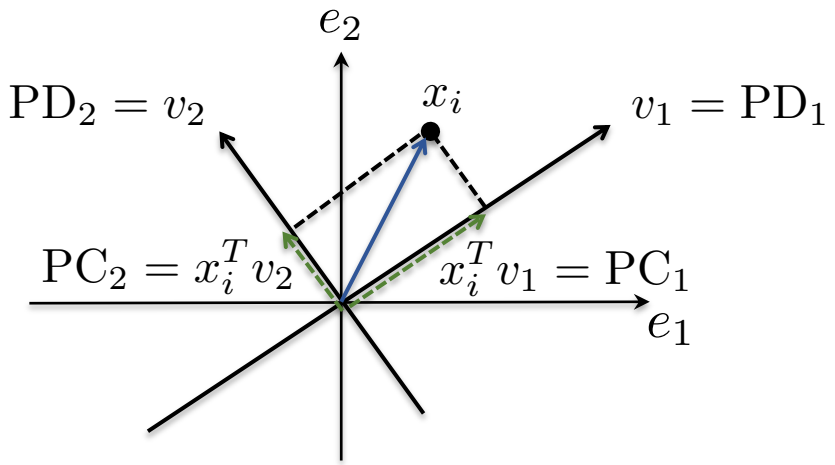
Altogether, we consider the full matrix factorization of C with EVD:

$C = V\Lambda V^T \in \mathbb{R}^{d\times d}$

with $V = \left[v_1, ..., v_d\right] \in \mathbb{R}^{d\times d}$, $V^T V = \mathrm{I}_d \in \mathbb{R}^{d\times d}$ (Identity matrix), $\Lambda = \text{diag}(\lambda_1, ..., \lambda_d) \in \mathbb{R}^{d\times d}$

Principal directions and components

- The principal directions (PDs) indicate the directions along which the data has the greatest variance.
- The EVD of the covariance matrix C provides
  - The principal directions v_j in R^d as the eigenvectors of C.
  - The magnitude of the variances along each direction $C_{jj} = \lambda_j$ as the eigenvalues of C.
- The principal components of a data point x_i are defined as the projection onto the basis formed by these principal directions :



$$x_i{}^{\mathrm{pca}} = V^T x_i \in \mathbb{R}^d \quad \text{(Rotated data along the PDs)}$$

$$X^{\mathrm{pca}} = XV \in \mathbb{R}^{n \times d} \quad \text{(Rotated data matrix } X \text{ along the PDs)}$$

Dimensionality reduction

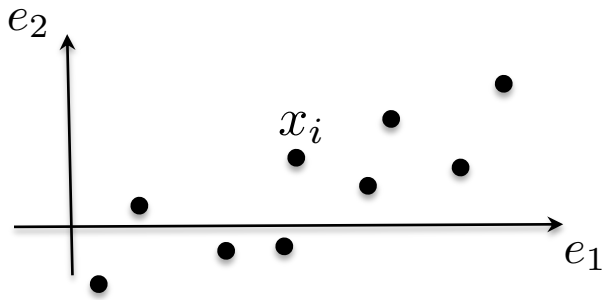
- **Suppose that the data is primarily concentrated along the first principal directions, the remaining directions, which mostly capture noise or insignificant details, can be discarded.**
- **The first K principal directions (PDs) can be selected as :**

$$k \text{ such that } \|X - X_k^{\text{pca}}\|_F^2 \leq \varepsilon$$

$$\text{where } X_k^{\text{pca}} = XV_k \in \mathbb{R}^{n \times k} \text{ is the approximation of } X \text{ with the first k PDs}$$

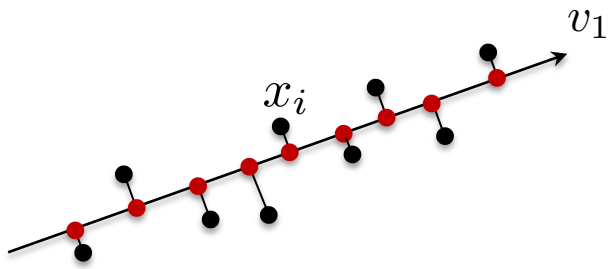
$$\text{and } V_k = \Big[v_1, ..., v_k\Big] \in \mathbb{R}^{d \times k}, V_k^T V_k = \mathrm{I}_k \in \mathbb{R}^{k \times k}$$

$$\text{is the truncated } V \text{ with the first k PDs}$$



Original data distribution in $\mathbb{R}^2$

$\Rightarrow$



Projection of data points into the direction of the largest variation v of the distribution

$$x_i{}^{\text{pca}} = V^T x_i \in \mathbb{R}^d$$
$$\approx V_k^T x_i \in \mathbb{R}^k$$
$$\approx v_1^T x_i \in \mathbb{R} \text{ (this example)}$$

Number of reduced dimensions

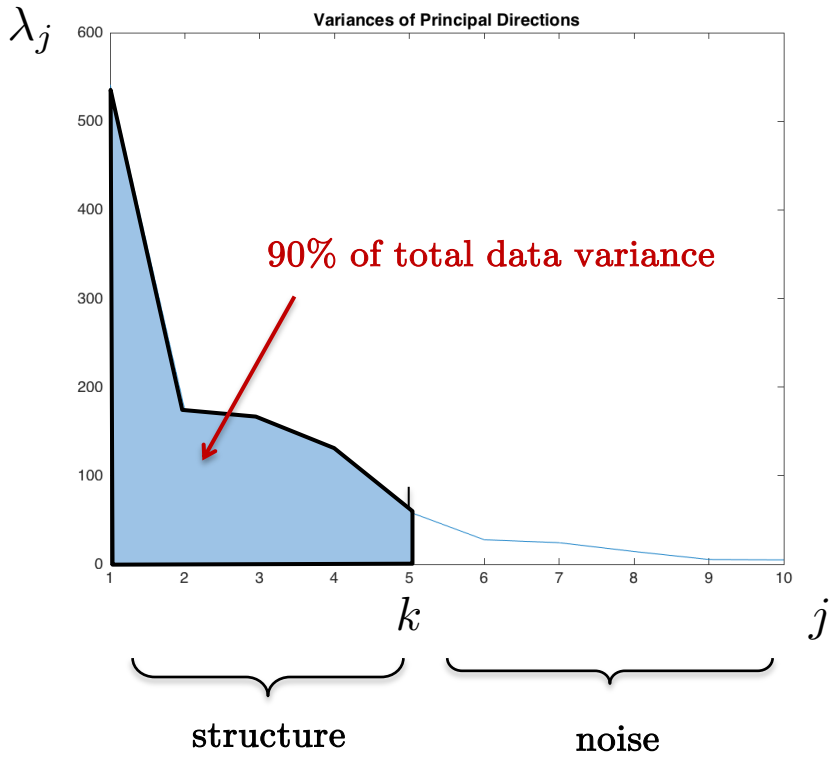
- Simple selection of the hyper-parameter k.
- Since principal directions capture the most significant variances in the data distribution, simply retain the first k directions that collectively account for e.g. 90% of the total data variance.

Select k such that $\|X - X_k^{\text{pca}}\|_F^2 \leq \varepsilon$

or simply

Select k such that $\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{d} \lambda_j} \geq 0.9$

YaleBFaces dataset



PCA as SVD of data matrix

- We identified the principal directions of variance by performing EVD on the covariance matrix.
- Alternatively, the same information can be derived using singular value decomposition (SVD) on the data matrix X :

$$X = U\Sigma W \in \mathbb{R}^{n\times d}$$

with $U \in \mathbb{R}^{n\times n},\ U^TU = \mathrm{I}_n, W \in \mathbb{R}^{d\times d},\ W^TW = \mathrm{I}_d, \Sigma \in \mathbb{R}^{n\times d}$

We have

$$C = X^TX = (U\Sigma W)^T(U\Sigma W)$$

$$W\Sigma(U^TU)\Sigma W^T = W\Sigma^2W^T \text{ (SVD)}$$

$$C = X^TX = V\Lambda V^T \text{ (EVD)}$$

As a result

$$V = W,\ \Lambda = \Sigma^2 \rightarrow \lambda_j = \sigma_j^2,\ X_k^{\text{pca}} = XV_k = U\Sigma_kW_k \in \mathbb{R}^{n\times k}$$

with Σ_k, W_k are truncated matrices of Σ, W with the k largest singular values

EVD or SVD

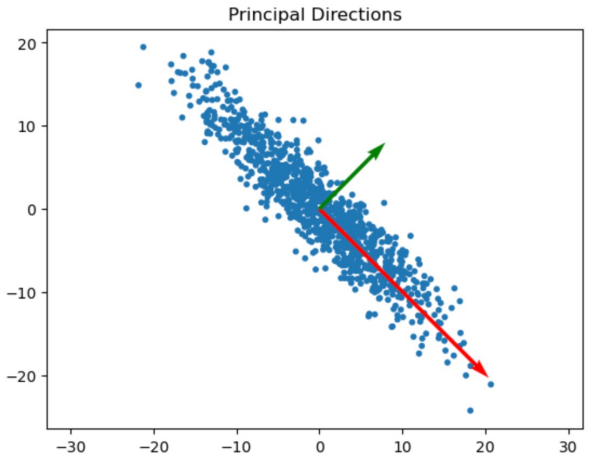
- The choice depends on the value of the size $(n \times d)$ of the data matrix X :
  - For $d < n$: Use EVD, complexity is $O(d^3)$
  - For $n < d$: Apply SVD, complexity is $O(\min(nd^2, n^2d))$
- Examples
  - MNIST dataset : $60{,}000 \times 684 \Rightarrow$ Apply EVD
  - Microarray-based gene expression dataset[1] : $240 \times 7{,}399 \Rightarrow$ Apply SVD

[1] Rosenwald et-al, The use of molecular profiling to predict survival after chemotherapy for diffuse large-B-cell lymphoma, 2002

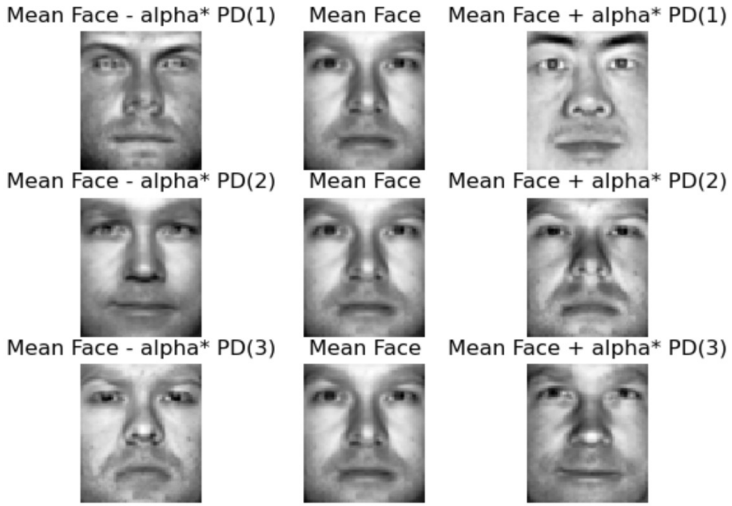
Lab 1 : (Standard) linear PCA

- Run code01.ipynb :
  - Visualize the principal directions of a Gaussian distribution.
  - Plot the distribution of data variances and generate new faces.
  - Compute the 3D PCA embedding of MNIST.

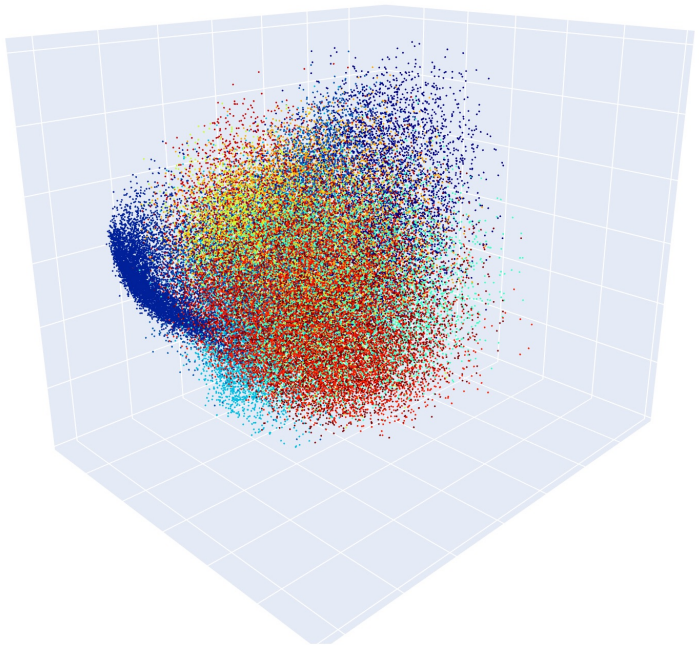
```python
# Zero-centered data
Xzc = X - np.mean(X,axis=0)

# Covariance matrix
CovX = (Xzc.T).dot(Xzc)

# Compute largest 5 eigenvectors/eigenvalues
nb_pca = 5
CovX = scipy.sparse.csr_matrix(CovX)
lamb, U = scipy.sparse.linalg.eigsh(CovX, k=nb_pca, which='LM') # U = d x nb_pca
EVec = U[:,::-1] # largest = index 0
EVal = lamb[::-1]

# Principal Components
Xpc = X.dot(EVec)

# Principal Directions
v1 = EVec[:2,0]
v2 = EVec[:2,1]
print(v1,v2,EVal[:2])
```



Principal Directions of a Gaussian



New faces generated with PCA



PCA of MNIST

Outline

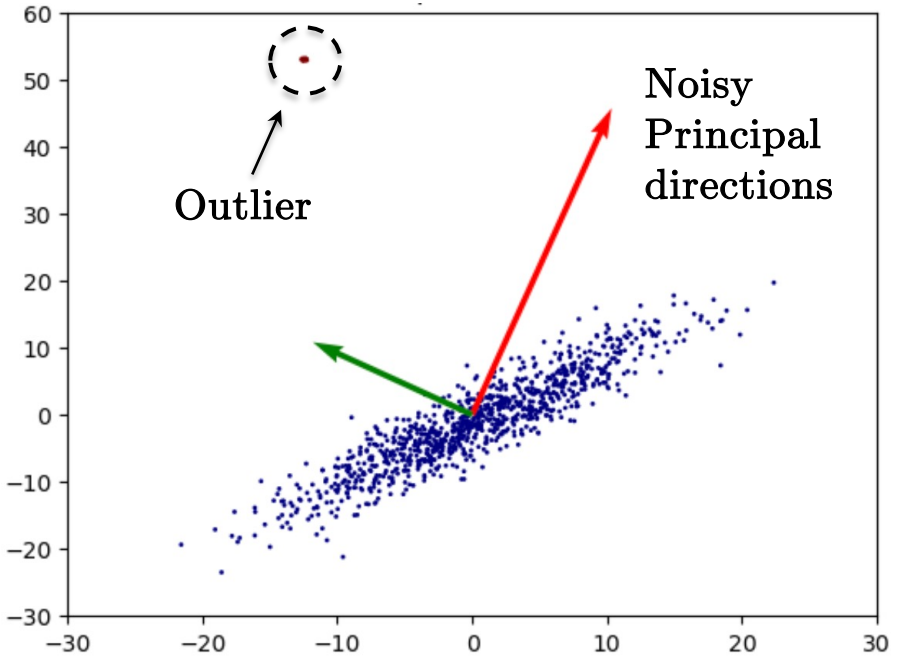
- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- Non-linear visualization techniques
  - LLE
  - Laplacian eigenmaps
  - TSNE
  - UMAP
- Conclusion

Robust PCA

- Standard PCA is sensitive to outliers; even a single outlier can significantly change the PCA solution.
- Robust PCA[1] is a technique designed to separate outliers from the data, allowing PCA to be performed on the clean part of the data.


Emmanuel Candes


Yi Ma



[1] Candes, Li, Ma, Wright, Robust principal component analysis, 2011 (8,000 citations)

Task formalization

- **Standard PCA :** $\min_{L \in \mathbb{R}^{n \times d}} \|X - L\|_F^2 \text{ s.t. } \text{rank}(L) = k$

- **Robust PCA :** $\min_{L,S \in \mathbb{R}^{n \times d}} \text{rank}(L) + \lambda \text{ card}(S) \text{ s.t. } X = L + S \in \mathbb{R}^{n \times d}$ (1)

  where $X \in \mathbb{R}^{n \times d}$ is the (noisy) data matrix

  L is a low-rank matrix that captures the clean/standard PCA (data structure)

  S is a sparse matrix that captures outliers (noise)

  card(.) is the cardinality of the matrix, i.e. the number of elements of the matrix

  The combinatorial optimization problem (1) is NP-hard,

  and thus requires a continuous relaxation:

  $\min_{L,S \in \mathbb{R}^{n \times d}} \|L\|_\star + \lambda \|S\|_1 \text{ s.t. } X = L + S \in \mathbb{R}^{n \times d}$ (2)

  where $\|\cdot\|_1$ is the L_1 norm

  $\|\cdot\|_\star$ is the nuclear norm (L_1 norm of the singular values)

  Theoretical result: Solution (2) is (almost) the solution of (1) !

Optimization algorithm

- Alternating direction method of multipliers (ADMM) technique[1,2] :
- Provides a fast, robust, and accurate solution to the relaxed problem (2).
- The core idea is to decompose the problem into simpler sub-problems using Lagrangian multipliers.

$$\min_{L,S\in\mathbb{R}^{ntimesd}} \|L\|_\star + \lambda \|S\|_1 \quad \text{s.t.} \quad X = L + S \in \mathbb{R}^{n\times d} \quad (2)$$

which is equivalent to

$$\min_{L,S,Z\in\mathbb{R}^{n\times d}} \|L\|_\star + \lambda \|S\|_1 + \langle Z, X-(L+S)\rangle + \frac{r}{2}\|Z-\big(X-(L+S)\big)\|_F^2, \; r>0$$

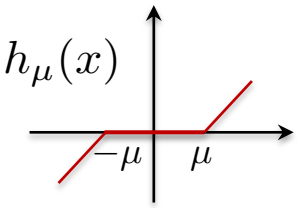
Initialization : $L^{m=0} = X \in \mathbb{R}^{n\times d}, \; S^{m=0} = Z^{m=0} = 0_{n\times d}$

Iterate until convergence : $m = 1, 2, ...$

$$L^{m+1} = U h_{1/r}(\Lambda) V^T \in \mathbb{R}^{n\times d} \quad \text{with } U\Lambda V^T \overset{\text{SVD}}{=} X - S^m + Z^m/r$$

$$S^{m+1} = h_{\lambda/r}\big(X - L^{m+1} + Z^m/r\big) \in \mathbb{R}^{n\times d}$$

$$Z^{m+1} = Z^m + r\big(X - L^{m+1} - S^{m+1}\big) \in \mathbb{R}^{n\times d}$$



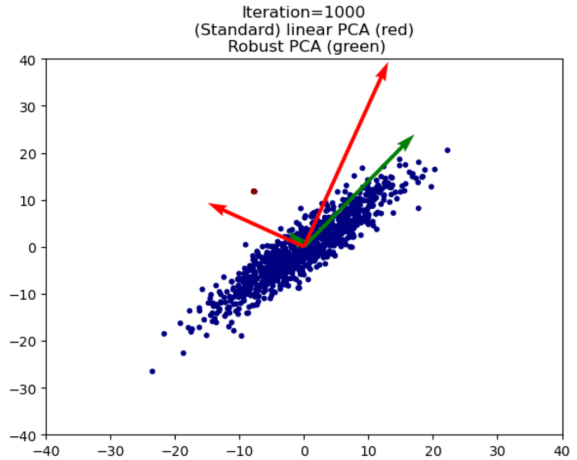
Shrinkage operator

[1] Glowinski, Le Tallec, Augmented Lagrangian and operator-splitting methods in nonlinear mechanics, 1989
[2] Boyd et-al, Distributed optimization and statistical learning via the alternating direction method of multipliers, 2011 (23,000 citations)

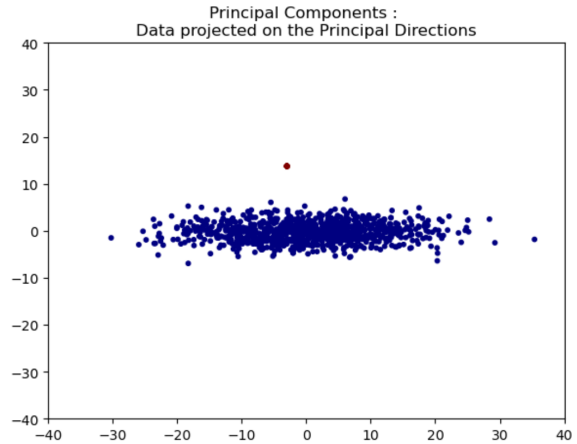
Lab 2 : Robust PCA

- Run code02.ipynb :
  - Visualize the principal directions and components of a noisy Gaussian distribution.
  - Compute the robust PCA solution and compare with the standard (noisy) solution.

```python
# Run Robust PCA
X = Xref - np.mean(Xref,axis=0)
L = X
S = np.zeros(X.shape)
Z = np.zeros(X.shape)
n,m = X.shape
min_nm = np.min([n,m])
r = 1
lambdaN = 1.
lambdaS = 0.1
for i in range(1001):

    # Update L
    Lold = L
    L = X - S + Z/r
    Usvd, Ssvd, Vsvd = np.linalg.svd(L) # L = U*S*V'
    Sdiag = shrink( Ssvd , lambdaN/ r )
    I = np.array(range(min_nm))
    Sshrink = np.zeros([n,m])
    Sshrink[I,I] = Sdiag
    L = Usvd.dot(Sshrink.dot(Vsvd))

    # Update S
    Sold = S
    S = X - L + Z/r
    S = shrink( S , lambdaS/ r )

    # Update Z
    Z = Z + r * ( X - L - S )
```



Robust PCA (green)
Noisy PCA (red)



Robust principal components

Outline

- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- Non-linear visualization techniques
  - LLE
  - Laplacian eigenmaps
  - TSNE
  - UMAP
- Conclusion

Graph-based PCA

- **Standard PCA :** $\min_{L \in \mathbb{R}^{n \times d}} \|X - L\|_F^2 \text{ s.t. } \text{rank}(L) = k$

- **Robust PCA :** $\min_{L,S \in \mathbb{R}^{n \times d}} \text{rank}(L) + \lambda \text{ card}(S) \text{ s.t. } X = L + S \in \mathbb{R}^{n \times d}$

- **Graph PCA[1] :** $\min_{L,S \in \mathbb{R}^{n \times d}} \text{rank}(L) + \lambda_S \text{ card}(S) + \lambda_G \|L\|_{\mathcal{G}} \text{ s.t. } X = L + S \in \mathbb{R}^{n \times d}$

  where $\|\cdot\|_{\mathcal{G}}$ is a graph smoothness term.

  The aim is to enhance PCA with data similarities represented by a graph.

  Problem is still NP-hard, and requires a new continuous relaxation:

  $$\min_{L,S \in \mathbb{R}^{n \times d}} \|L\|_\star + \lambda \|S\|_1 + \lambda_G \|L\|_{\text{Dir}_{\mathcal{G}}} \text{ s.t. } X = L + S \in \mathbb{R}^{n \times d}$$

  where $\|\cdot\|_{\text{Dir}_{\mathcal{G}}}$ is the graph Dirichlet norm, defined as

  $\|L\|_{\text{Dir}_{\mathcal{G}}} = \text{tr}(L^T \mathcal{L}_{\mathcal{G}} L)$ with the graph Laplacian $\mathcal{L}_{\mathcal{G}} \in \mathbb{R}^{n \times n}$.

[1] Shahid, Kalofolias, Bresson, Bronstein, Vandergheynst, Robust principal component analysis on graphs, 2015

Optimization algorithm

- **ADMM technique :**

$$\min_{L,S\in\mathbb{R}^{n\times d}} \|L\|_\star + \lambda_S \|S\|_1 + \lambda_G \|L\|_{\mathrm{Dir}_\mathcal{G}} \quad \text{s.t.} \quad X = L + S \in \mathbb{R}^{n\times d}$$

is equivalent to

$$\min_{L,S,M\in\mathbb{R}^{n\times d}} \|L\|_\star + \lambda_S \|S\|_1 + \lambda_G \|M\|_{\mathrm{Dir}_\mathcal{G}} \quad \text{s.t.} \quad X = L + S \in \mathbb{R}^{n\times d},\ M = L \in \mathbb{R}^{n\times d}$$

as well as $\min_{L,S,M,Z_1,Z_2\in\mathbb{R}^{n\times d}} \|L\|_\star + \lambda \|S\|_1 + \lambda_G \|M\|_{\mathrm{Dir}_\mathcal{G}} +$

$$\langle Z_1, X - (L+S)\rangle + \frac{r_1}{2}\|Z_1 - \big(X - (L+S)\big)\|_F^2 + \langle Z_2, L - M\rangle + \frac{r_2}{2}\|Z_2 - (L-M)\|_F^2$$

Initialization : $L^{m=0} = X \in \mathbb{R}^{n\times d},\ S^{m=0} = M^{m=0} = Z_1^{m=0} = Z_2^{m=0} = 0^{n\times d}$

Iterate until convergence : $m = 1, 2, ...$

$$L^{m+1} = U h_{1/r}(\Lambda) V^T \in \mathbb{R}^{n\times d} \quad \text{with } U\Lambda V^T \overset{\text{SVD}}{=} X - S^m + Z_1^m/r_1$$

$$S^{m+1} = h_{\lambda/r}\big(X - L^{m+1} + Z_1^m/r_1\big) \in \mathbb{R}^{n\times d}$$

$$M^{m+1} = \big(\mathrm{I}_n + \lambda_G \mathcal{L}_\mathcal{G}\big)^{-1}(L^{m+1} + Z_2^m/r_2) \in \mathbb{R}^{n\times d}$$

$$Z_1^{m+1} = Z_1^m + r_1\big(X - L^{m+1} - S^{m+1}\big) \in \mathbb{R}^{n\times d}$$

$$Z_2^{m+1} = Z_2^m + r_2\big(L^{m+1} - M^{m+1}\big) \in \mathbb{R}^{n\times d}$$

Application to video surveillance

- **Separate the background from moving objects :**

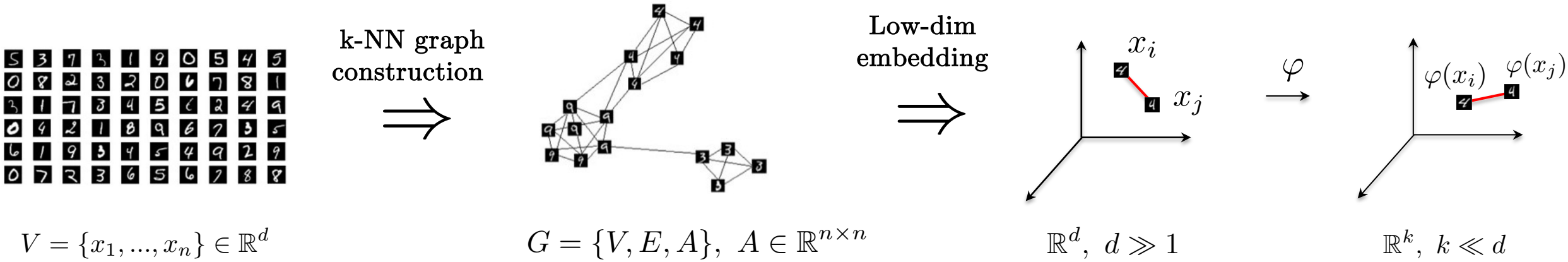
$$X = L + S$$

Outline

- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- **Non-linear visualization techniques**
  - LLE
  - Laplacian eigenmaps
  - TSNE
  - UMAP
- Conclusion

Non-linear visualization techniques

- All non-linear dimensionality reduction techniques follow a two-step approach:
  - Construct a k-nearest neighbor (kNN) graph G from the n high-dimensional data points $\{x_i\} \in R^d$, $d \gg 1$.
  - Compute a low-dimensional embedding of the graph that preserves
    - The geometric distance between neighboring data points, i.e. the graph structure, and
    - Additional properties specific to the chosen dimensionality reduction technique, s.a. physical forces.



Non-linear visualization techniques

- We will explore the following non-linear visualization techniques:
  - LLE[1] and Laplacian Eigenvectors[2], which are spectral techniques.
  - TSNE[3], a technique based on probability matching.
  - UMAP[4], which uses a physics-based approach.

[1] Roweis, Saul, Nonlinear dimensionality reduction by locally linear embedding, 2000 (18,000 citations)
[2] Belkin, Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, 2003 (10,000 citations)
[3] Van der Maaten, Hinton, Visualizing data using t-SNE, 2008 (46,000 citations)
[4] McInnes et-al, UMAP: Uniform manifold approximation and projection for dimension reduction, 2018 (13,000 citations)

Outline

- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- **Non-linear visualization techniques**
  - **LLE**
  - Laplacian eigenmaps
  - TSNE
  - UMAP
- Conclusion

LLE

Sam Roweis
1973-2010

Lawrence Saul

- Locally Linear Embedding[1](LLE) was one of the pioneering non-linear visualization techniques.
- It involves three key steps:
  - First, construct a k-nearest neighbor graph G the high-dimensional data distribution $\{x_i\} \in R^d$, $d \gg 1$.
  - Second, approximate the high-dimensional data distribution as a manifold $\mathcal{M}$ discretized with local linear patches, i.e. a data point x_i and its neighbors $\{x_j\}_{j \in Ni}$ lie on a locally linear patch of $\mathcal{M}$.
  - Third, project the high-dimensional data points $\{x_i\} \in R^d$ into a low-dimensional Euclidean space $\{z_i\} \in R^k$, $k \ll d$ by preserving data proximity, i.e. if data i close to data j then z_i should be similar to z_j.

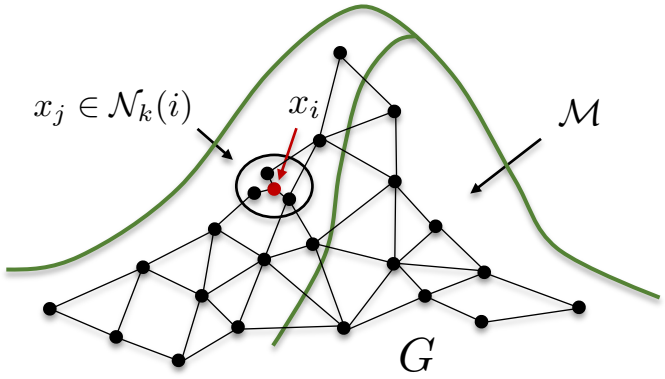
[1] Roweis, Saul, Nonlinear dimensionality reduction by locally linear embedding, 2000

Algorithm

- **Step 1 : Compute a k-nearest neighbor graph G.**
  - **For each data x_i, we identify its k nearest neighbors $\{x_j\}_{j \in N(i)}$.**
  - **Then, we compute the adjacency matrix A of the graph :**

$$A_{ij} = \begin{cases} \exp(-\frac{\text{dist}(x_i x_j)^2}{\sigma^2}) & \text{if } j \in \mathcal{N}_k(i)) \\ 0 & \text{otherwise} \end{cases}$$

where σ is the scale parameter, defined e.g.
as the mean distance of all k-th neighbors.
and $\text{dist}(\cdot, \cdot)$ is the distance between the two data vectors,
e.g. L^2 norm.



Algorithm

- **Step 2 : Compute linear patches.**
  - **Find the weights $W_{ij} \in [0,1]$ which best linearly reconstruct x_i from its neighbors :**

$$\min_{W \in \mathbb{R}^{n \times n}} \frac{1}{2} \sum_{i=1}^{n} \left\| x_i - \sum_{j=1}^{n} W_{ij} A_{ij} x_j \right\|_2^2 \text{ s.t. } \sum_{j=1}^{n} W_{ij} = 1 \ \forall i$$

Equivalently, $$\min_{W \in \mathbb{R}^{n \times n}} \frac{1}{2} \sum_{i=1}^{n} \left\| x_i - \sum_{j \in \mathcal{N}_i} W_{ij} x_j \right\|_2^2 \text{ s.t. } \sum_{j \in \mathcal{N}_i} W_{ij} = 1 \ \forall i$$

$$\min_{W \in \mathbb{R}^{n \times n}} \frac{1}{2} \left\| \sum_{j \in \mathcal{N}_i} W_{ij} x_i - \sum_{j \in \mathcal{N}_i} W_{ij} x_j \right\|_2^2 = \frac{1}{2} \left\| \sum_{j \in \mathcal{N}_i} W_{ij} (x_i - x_j) \right\|_2^2 \text{ s.t. } \sum_{j \in \mathcal{N}_i} W_{ij} = 1 \ \forall i$$

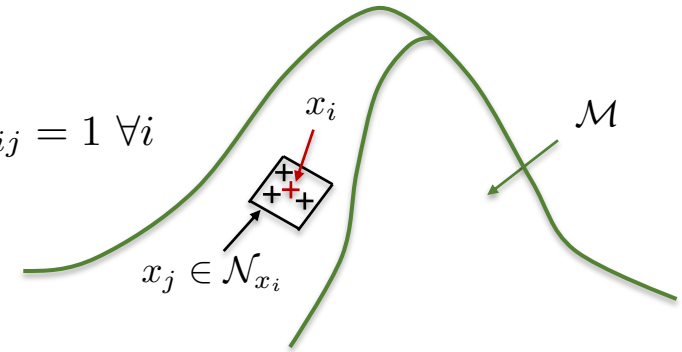
We derive the solution of this least-square problem for one data point x_i :

with $Z_{ij} = x_i - x_j \in \mathbb{R}^{d \times 1}$ and $Z_i = (x_i 1_{n_i})^T - (1_{n_i} A_{i, j \in \mathcal{N}_i})^T X \in \mathbb{R}^{n_i \times d}$

and n_i is the number of points in $\mathcal{N}_i$

We can re-write the problem as

$$\min_{W_i \in \mathbb{R}^{n_i \times 1}} \frac{1}{2} \left\| W_i^T Z_i \right\|_2^2 \text{ s.t. } W_i^T 1_{n_i} = 1$$



Algorithm

- **Step 2 : Compute linear patches.**
  - **Find the weights $W_{ij} \in [0,1]$ which best linearly reconstruct x_i from its neighbors :**

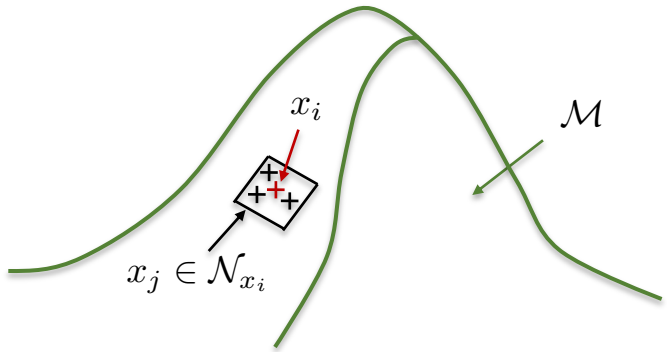
$$\min_{W_i \in \mathbb{R}^{n_i \times 1}} \frac{1}{2}\left\|W_i^T Z_i\right\|_2^2 \text{ s.t. } W_i^T 1_{n_i} = 1$$

Lagrange multiplier technique : $\min_{W_i \in \mathbb{R}^{n_i \times 1}, \lambda_i \in \mathbb{R}} \frac{1}{2}\left\|W_i^T Z_i\right\|_2^2 + \lambda_i (W_i^T 1_{n_i} - 1)$

Derivative w.r.t. W_i : $W_i^T Z_i Z_i^T + \lambda_i 1_{n_i}^T = 0 \Rightarrow W_i = (Z_i Z_i^T)^{-1} \lambda_i 1_{n_i}$

Derivative w.r.t. λ_i : $W_i^T 1_{n_i} - 1 = 0 \Rightarrow \lambda_i = \frac{1}{1_{n_i}^T (Z_i Z_i^T)^{-1} 1_{n_i}}$

Finally, the solution for data x_i is $W_i = \frac{(Z_i Z_i^T)^{-1} 1_{n_i}}{1_{n_i}^T (Z_i Z_i^T)^{-1} 1_{n_i}} \in \mathbb{R}^{n_i \times 1}$



Matrix $C = Z_i Z_i^T \in \mathbb{R}^{n_i \times n_i}$ is called the correlation or Gram matrix

In practice, a small identity matrix is added for numerical stability : $C = Z_i Z_i^T + \varepsilon \mathrm{I}_{n_i}$

Algorithm

- **Step 3 : Compute the low-dim embedding data z_i with the weights W_{ij} :**
  - **Find the coordinates $z_i \in \mathbb{R}^k$ which best linearly reconstruct z_i from its neighbors :**

$$\min_{Z=[z_1,\dots,z_n]\in\mathbb{R}^{n\times k}} \sum_{i=1}^{n} \left\| z_i - \sum_{j=1}^{n} W_{ij} z_j \right\|_2^2 \quad \text{s.t.} \quad Z^T 1_n = 0_n, Z^T Z = \mathrm{I}_n$$

The solution is given by EVD.

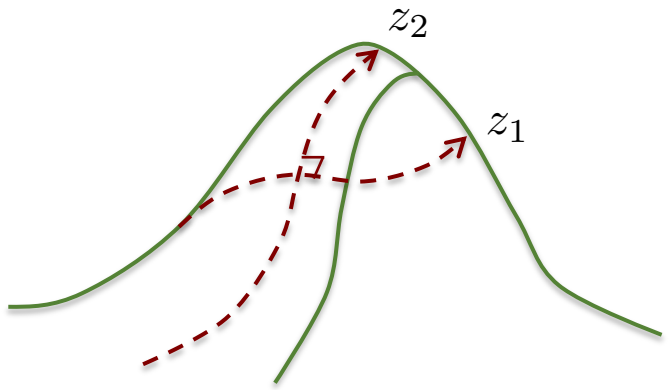
$$\min_Z \ \|Z - WZ\|_F^2 \text{ s.t. } Z^T Z = \mathrm{I}_n$$

$$\min_Z \ \mathrm{tr}((Z - WZ)^T (Z - WZ)) \text{ s.t. } Z^T Z = \mathrm{I}_n$$

$$\min_Z \ \mathrm{tr}(Z^T (\mathrm{I}_n - W^T)(\mathrm{I}_n - W) Z) \text{ s.t. } Z^T Z = \mathrm{I}_n$$

Solution is given by EVD of the matrix $M = (\mathrm{I}_n - W^T)(\mathrm{I}_n - W) \overset{\text{EVD}}{=} U \Sigma U^T$

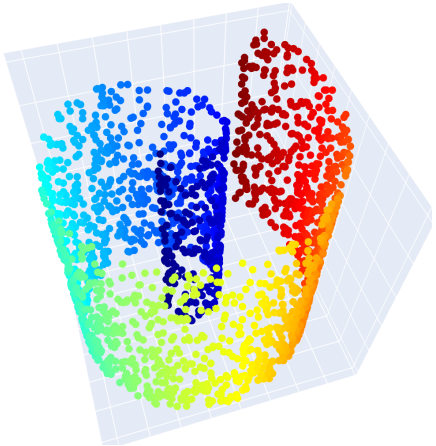
with $Z = U_{\cdot,1,\dots,k} \in \mathbb{R}^{n\times k}$



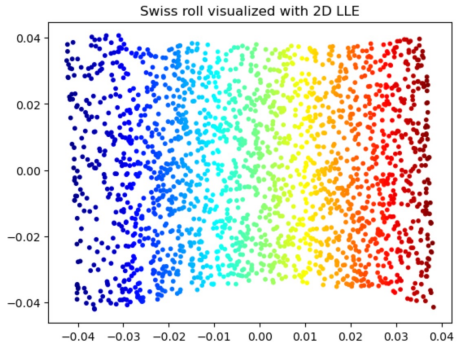
Lab 3 : LLE

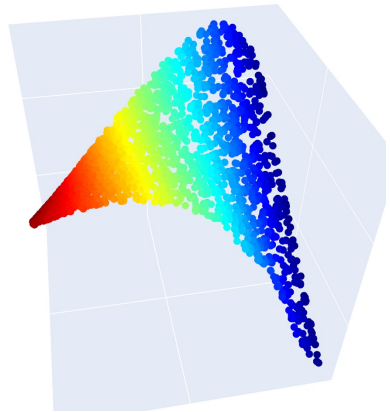
- Run code03.ipynb :
  - Compute the LLE solution for the Swiss Roll dataset.
  - Visualize the MNIST dataset with the LLE technique.

```
# Run LLE
X = Xref
X = X - np.mean(X,axis=0) # zero-centered data

# Step 1: Compute k-NN
kNN = 20
WkNN = construct_knn_graph(X,kNN,'euclidean').todense()

# Step 2: Compute locally linear patches
start = time.time()
n = X.shape[0]
W = np.zeros([n,n])
for i in range(n):
    # Find neighbors of data i
    idx_kNN = np.where(WkNN[i,:]>0.0)[1]
    K = len(idx_kNN)
    if K>kNN:
        K = kNN
        idx_kNN = idx_kNN[:K]
    XkNN = X[idx_kNN,:]
    Ones = np.ones([K,1])
    C = (X[i,:][:,None].dot(Ones.T) - XkNN.T).T
    C = C.dot(C.T)
    # if K (nb nearest neighbors) > d (data dimensionality)
    C = C + 1e-1* np.eye(K)#* trace(C)
    t,_ = scipy.sparse.linalg.cg(C, Ones)
    t = t/ (Ones.T.dot(t))
    W[i,idx_kNN] = t
print('time(min) step 2 :',(time.time()-start)/60)

# Step 3: Compute low-dim embedding coordinates
start = time.time()
M = ( np.eye(n) - W.T ).dot( np.eye(n) - W )
# EVD
lamb, U = np.linalg.eig(M)
def sortEVD(lamb, U):
    idx = lamb.argsort()
    return lamb[idx], U[:,idx]
lamb, U = sortEVD(lamb, U)
print(lamb[:4])
Xvis = U[:,1]
Yvis = U[:,2]
Zvis = U[:,3]
print('time(min) step 3 :',(time.time()-start)/60)
```

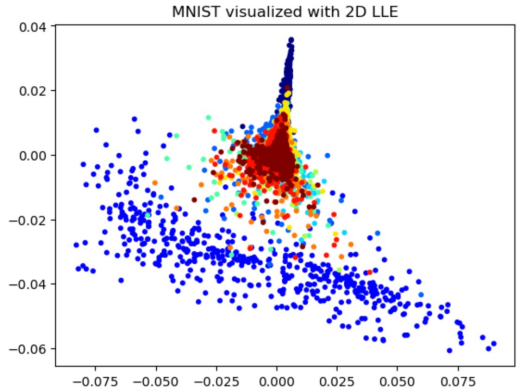


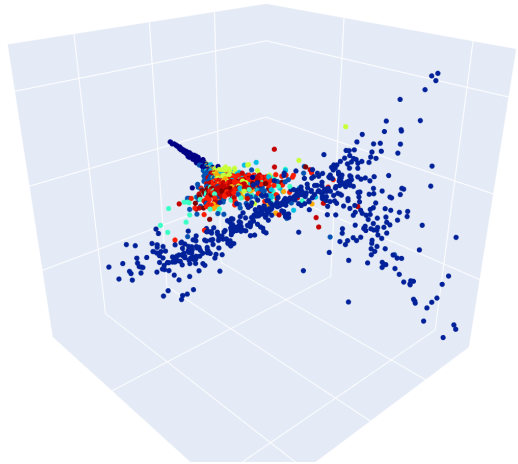
Swiss Roll dataset





2D and 3D LLE
of Swiss Roll dataset





2D and 3D LLE
of MNIST dataset

Outline

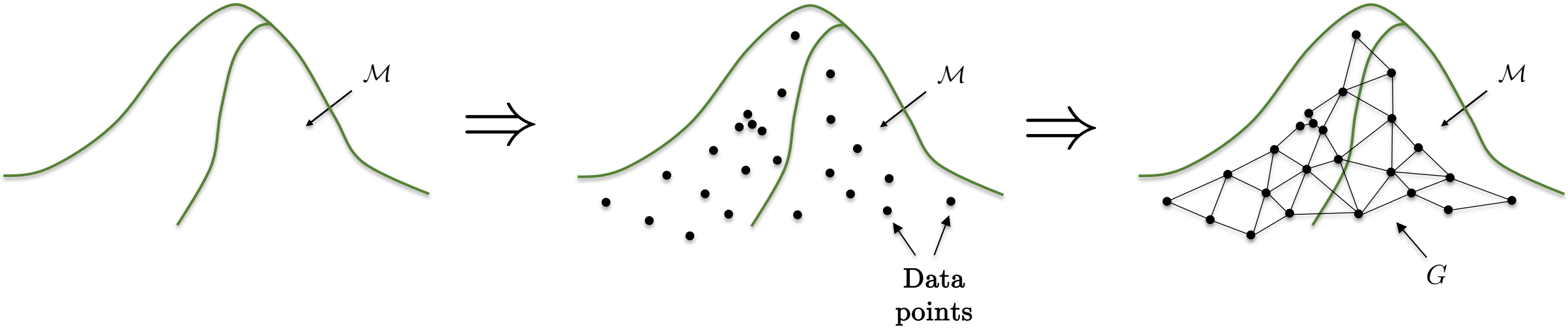
- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- **Non-linear visualization techniques**
  - LLE
  - **Laplacian eigenmaps**
  - TSNE
  - UMAP
- Conclusion

Laplacian eigenmaps

Misha Belkin

Partha Niyogi
1967-2010

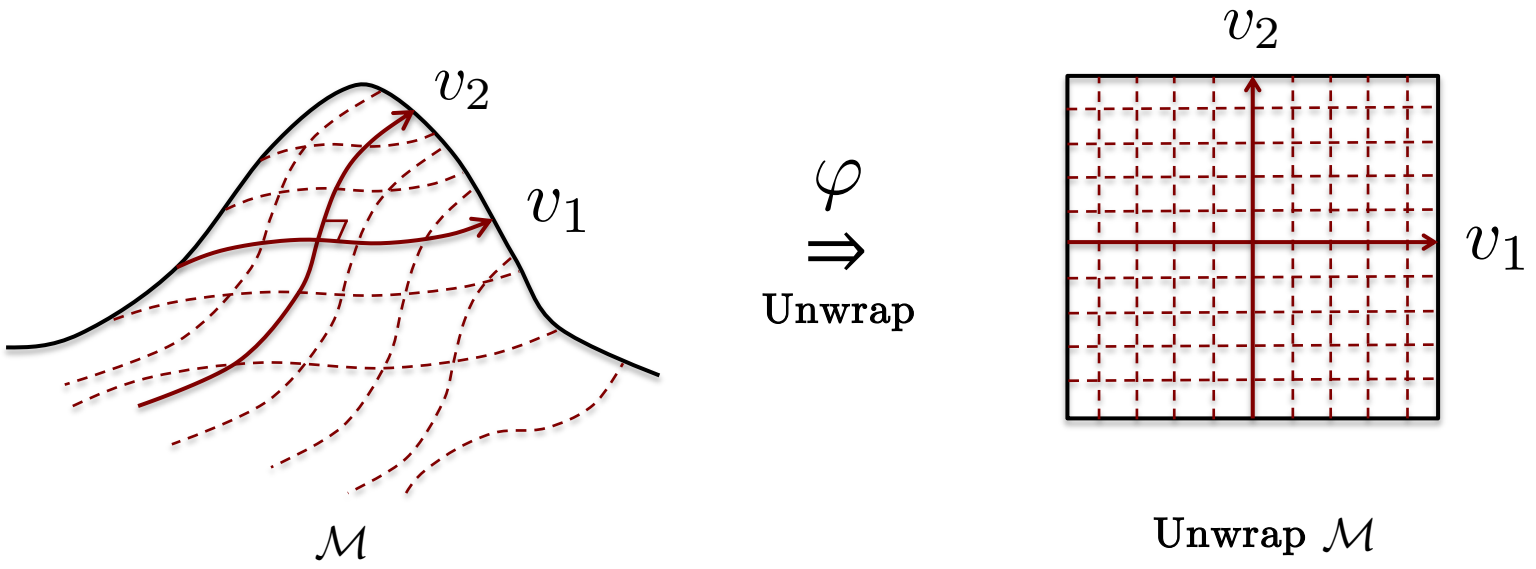
- Laplacian eigenmaps technique[1] was one of the first non-linear visualization techniques grounded in mathematical theory.
- It is based on the manifold assumption, i.e. the data distribution is sampled from a smooth and continuous manifold $\mathcal{M}$.
- Since the manifold cannot be directly observed, it is approximated using a k-nearest neighbor graph.



[1] Belkin, Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, 2003

Spectral analysis and differential geometry

- Mathematical tools have been developed to analyze smooth, continuous manifolds[1].
- The eigenfunctions $v_k \in R^d$ of the continuous Laplace-Beltrami operator $\Delta_{\mathcal{M}}$ can serve as embedding coordinates for $\mathcal{M}$.
- The discretization of $\Delta_{\mathcal{M}}$ provides the graph Laplacian L (but it is not unique, i.e. multiple definitions exist).



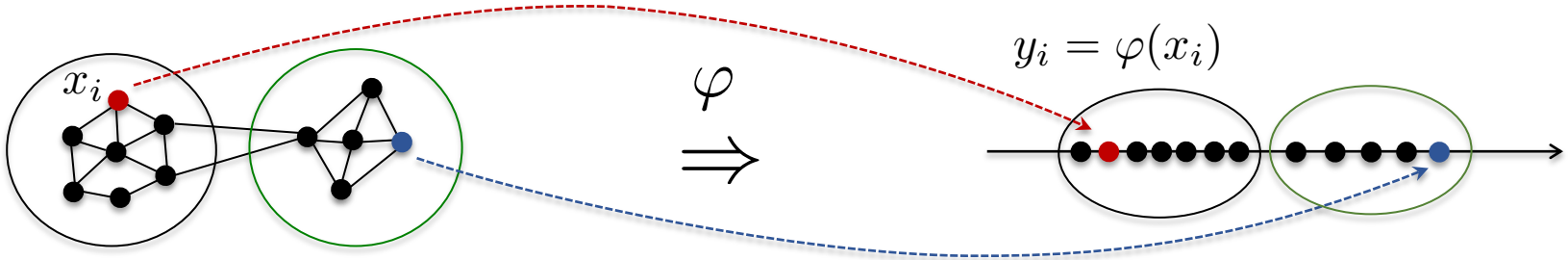
[1] Chung, Spectral graph theory, 1997 (11,000 citations)

Task formalization

- Let us begin with a simple 1D dimensionality reduction.
- The goal is to map a given graph G = (V,E,A) onto a line with the constraint that neighboring data points on G remain as close as possible on the line.
- To achieve this, we can design a loss function that computes the mapping y = φ(x) such that :

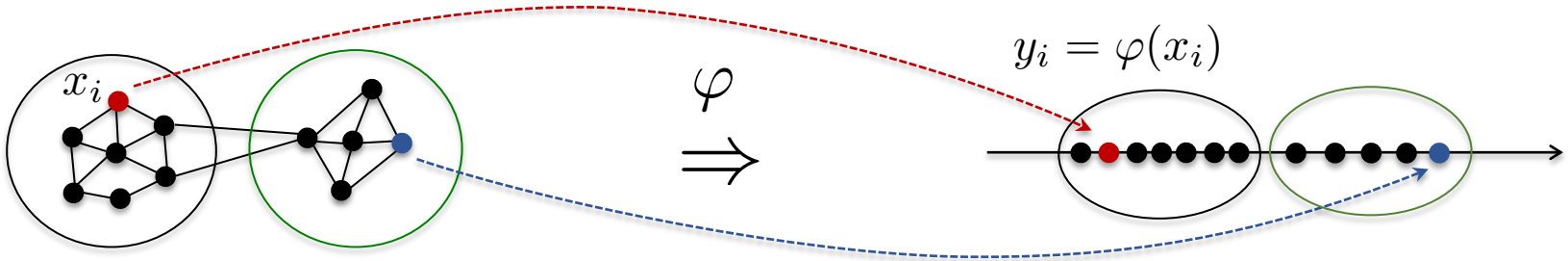
$$\min_{y \in \mathbb{R}} \sum_{ij} A_{ij}(y_i - y_j)^2, \text{ with } y_i = \varphi(x_i), y_j = \varphi(x_j), \text{ and}$$

$$A_{ij} = \begin{cases} 1 & \text{if } j \in \mathcal{N}_i \\ 0 & \text{otherwise} \end{cases} \quad \text{as a graph adjacency matrix}$$



Task formalization

- Let us analyze the loss function : $\min_{y \in \mathbb{R}^n} \sum_{ij} A_{ij}(y_i - y_j)^2, \quad \text{with } y_i, y_j \in \mathbb{R}$
  - When $A_{ij} \approx 1$, i.e. x_i is close to x_j, then minimizing the loss encourages y_i to be similar to y_j.
  - When $A_{ij} \approx 0$, i.e. x_i is far from x_j, then minimizing the loss allows y_i to differ significantly from y_j.
- In summary, minimizing this loss ensures that data points that are close in the high-dim space remain close in the low-dim space, satisfying the first key property of dimensionality reduction techniques.
- Observe that the non-linear mapping φ is never explicitly computed.



Task formalization

- **Finally, the loss function can be reformulated in terms of the Laplacian operator :**

$$\min_{y^n \in \mathbb{R}} \sum_{ij} A_{ij}(y_i - y_j)^2 = \sum_{ij} L_{ij} y_i y_j = y^T L y$$

with $L = D - A$

The unit-vector constraint, i.e. the orthogonality constraint $y^T y = 1$, avoids the trivial solution $y = 0$

The constraint $y^T 1_n = 0$, avoids a constant solution (by centering y)

In summary, we have the constrained optimization problem :

$$\min_{y^n \in \mathbb{R}} \; y^T L y \;\; \text{s.t.} \;\; y^T y = 1, \; y^T 1_n = 0$$

which solution is given by EVD of L :

$Lu_1 = \lambda_1 u_1 \in \mathbb{R}^n$, with λ_1 is the smallest non-zero eigenvalue of L, with its eigenvector u_1

Generalization to k dimensions

- **Let us extend this mapping process, i.e. from a graph to a k-dimensional Euclidean space :**

$$\min_{y^n \in \mathbb{R}} \; y^T L y \text{ s.t. } y^T y = 1, \; y^T 1_n = 0 \quad \text{(1D mapping)}$$

$$\min_{Y=[y_1,\ldots,y_n] \in \mathbb{R}^{n \times k}} \; \sum_{m=1}^{k} Y_{\cdot,m}^T L Y_{\cdot,m} = \text{tr}(Y^T L Y) \quad \text{(k-D mapping)}$$

with the generalized orthogonality constraint $Y^T Y = \text{I}_k$

Spectral Solution : Solution is given by the k non-zero smallest eigenvectors of the graph Laplacian $L = A - D$:

$$L \overset{\text{EVD}}{=} U \Lambda U^T \;\Rightarrow\; Y = U_{\cdot,1,..,k} \in \mathbb{R}^{n \times k}$$

Properties : Global solution (independent of initialization) and $O(n^2 k)$ complexity

Normalized Laplacian

- **Considering the importance of the nodes with the degree matrix D :**

$$\min_{Y \in \mathbb{R}^{n \times k}} \operatorname{tr}(Y^T L Y) = \operatorname{tr}(Y^T (D - A) Y) \quad \text{s.t.} \quad Y^T D Y = \mathrm{I}_k$$

Change of variable : $Z = D^{1/2} Y$ so we have

$$Y^T D Y = (D^{-1/2} Z)^T D^{-1/2} Z = Z^T Z$$

and

$$\min_{Z \in \mathbb{R}^{n \times k}} \operatorname{tr}(D^{-1/2} Z)^T (D - A) D^{-1/2} Z) \quad \text{s.t.} \quad Z^T Z = \mathrm{I}_k$$

$$\operatorname{tr}(Z^T D^{-1/2} (D - A) D^{-1/2} Z) \quad \text{s.t.} \quad Z^T Z = \mathrm{I}_k$$

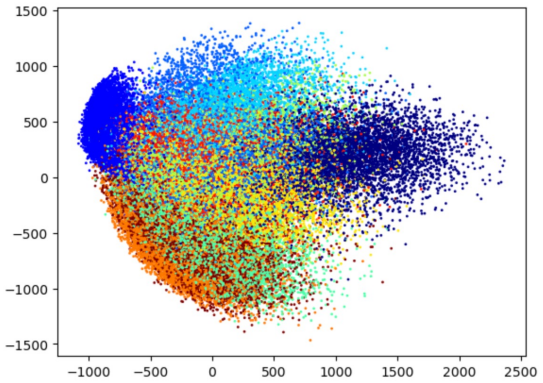
$$\operatorname{tr}(Z^T (\mathrm{I} - D^{-1/2} A D^{-1/2}) Z) = \operatorname{tr}(Z^T \mathcal{L} Z) \quad \text{s.t.} \quad Z^T Z = \mathrm{I}_k$$

where $\mathcal{L} = \mathrm{I} - D^{-1/2} A D^{-1/2}$ is the normalized graph Laplacian

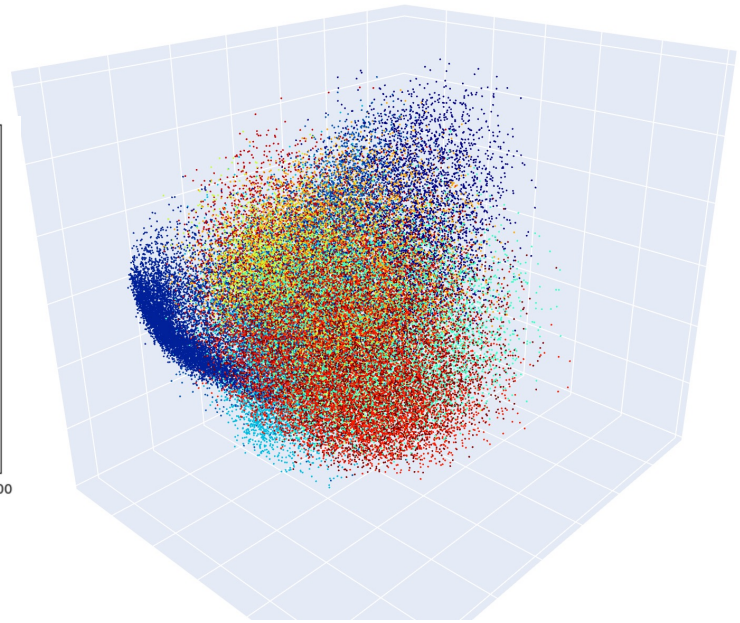
Lab 4 : Laplacian eigenmaps

- Run code04.ipynb :
  - Visualize MNIST with PCA.
  - Compare PCA with Laplacian eigenmaps.

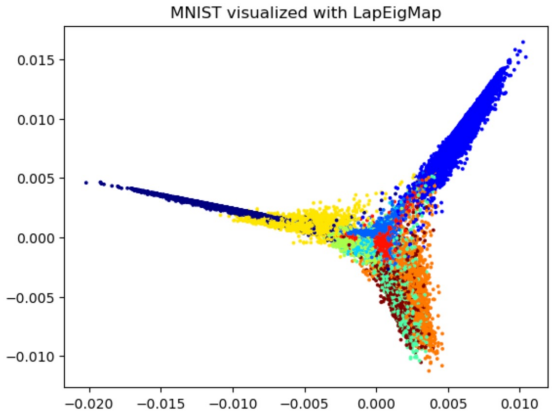
```
# Compute a k-NN graph
kNN = 10
dist = 'euclidean'
W = construct_knn_graph(X, kNN, dist)
#print(W)

# Laplacian Eigenmaps
start = time.time()
Xvis,Yvis,Zvis = nldr_visualization(W)
print('time(sec):',(time.time()-start)/1)
```



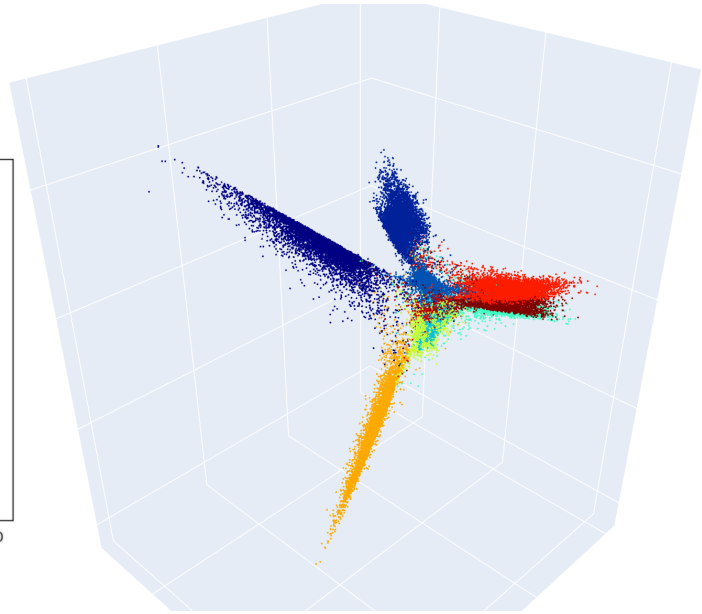
2D PCA
of MNIST dataset



3D PCA
of MNIST dataset



2D LapEigMaps
of MNIST dataset



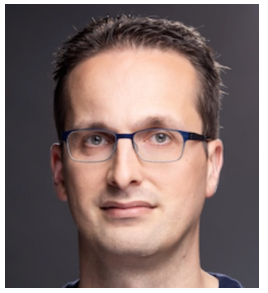
3D LapEigMaps
of MNIST dataset

Outline

- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- **Non-linear visualization techniques**
  - LLE
  - Laplacian eigenmaps
  - **TSNE**
  - UMAP
- Conclusion

TSNE


Geoffrey Hinton


Laurens van der Maaten

- T-distributed Stochastic Neighbor Embedding[1] (TSNE) has been the most successful non-linear visualization technique.
- It involves four steps :
  - Step 1: Compute a k-nearest neighbor graph G from the high-dimensional data points $\{x_i\} \in R^d$, $d \gg 1$.
  - Step 2: Represent the distribution of the high-dim points using exponential weights :

$$p_{ij} = \text{Prob}_{\text{high-dim}}(i,j) = \frac{e^{-\|x_i - x_j\|_2^2/\sigma_i^2}}{\sum_{m=1}^{n} e^{-\|x_i - x_m\|_2^2/\sigma_i^2}}$$

where σ_i is the nearest neighbour distance from data point i

[1] Van der Maaten, Hinton, Visualizing data using t-SNE, 2008
[2] Interactive demo : https://distill.pub/2016/misread-tsne

TSNE

- **Step 3: Parametrize the distribution of the low-dim points using polynomial weights :**

$$q_{ij}(y) = \text{Prob}_{\text{low-dim}}(i,j) = \frac{(1+\|y_i - y_j\|_2^2)^{-1}}{\sum_{m=1}^{n}(1+\|y_i - y_m\|_2^2)^{-1}}$$

with $y = \varphi(x) \in \mathbb{R}^{n \times k}$ are the low-dim embedding coordinates of data points.

- **Step 4: Minimize the Kullback-Leibler divergence (/distance) between the high-dim distribution P and the low-dim distribution Q parametrized by Y :**

$$\min_{Y \in \mathbb{R}^{n \times k}} \text{D}_{\text{KL}}(P, Q(Y))$$

$$\text{with } \text{D}_{\text{KL}}(P_1, P_2) = \sum_m P_1(m) \log \frac{P_1(m)}{P_2(m)}$$

[1] Van der Maaten, Hinton, Visualizing data using t-SNE, 2008

Optimization problem

- The minimization problem is a continuous non-convex problem.
- Standard gradient descent (GS) can be applied.
- However, since the problem is non-convex, the GD solution is not guaranteed to reach a global minimum.
- In practice, PCA is often used as the starting point for initialization.
- Although GD is a slow optimization process, TSNE has significant advantages :
  - TSNE does not enforce the manifold assumption, meaning there is no orthogonality constraint $Y^TY=I$, which allows for greater flexibility in representing complex structures.
  - Minimizing the KL loss ensures that local distances in the high-dimensional data distribution are preserved in the low-dimensional representation.

Algorithm

- **Optimization problem :**

$$\min_{Y \in \mathbb{R}^{n \times k}} \mathrm{D}_{\mathrm{KL}}(P, Q(Y)) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}(Y)}$$

is minimized by the standard gradient descent :

Initialization : $Y^{t=0} = \mathrm{PCA}(X) \in \mathbb{R}^{n \times k}$

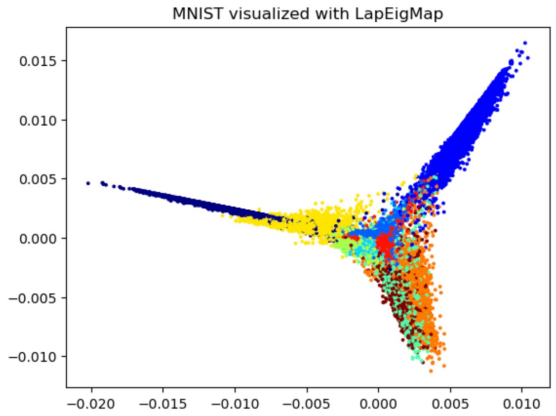
Iterate until convergence, $t = 1, 2, \ldots$:

$$y_i^{t+1} = y_i^t - \mathrm{lr} \sum_{j=1}^{n} (p_{ij} - q_{ij}^t)(1 + \|y_i^t - y_j^t\|_2^2)^{-1}(y_i^t - y_j^t) \in \mathbb{R}^k$$

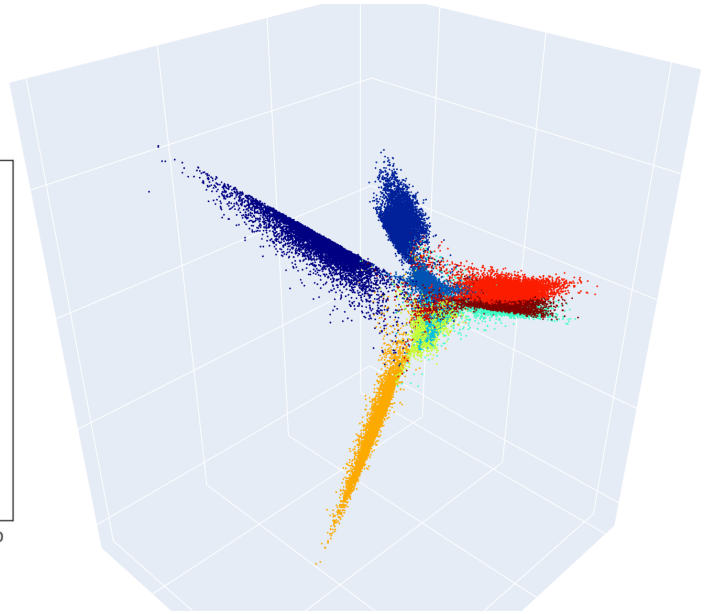
Lab 5 : TSNE

- Run code05.ipynb :
  - Compare TSNE with Laplacian Eigenmaps using MNIST.

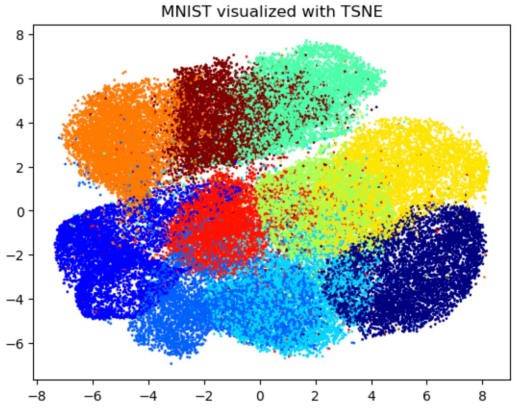
```python
# TSNE
start = time.time()
#tsne = TSNE(n_components=3, learning_rate='auto', init='random', perplexity=3)
tsne = TSNE(n_components=3, verbose=1, perplexity=40, n_iter=300)
print(X.shape)
embedding = tsne.fit_transform(X)
print('time(sec):',(time.time()-start)/1)
print(embedding.shape)
Xvis = embedding[:,0]
Yvis = embedding[:,1]
Zvis = embedding[:,2]
```



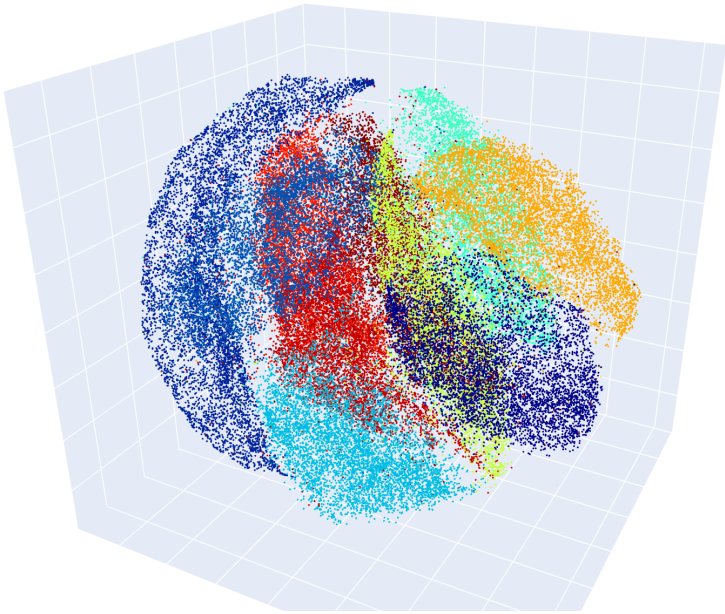
2D LapEigMaps
of MNIST dataset



3D LapEigMaps
of MNIST dataset



2D TSNE
of MNIST dataset



3D TSNE
of MNIST dataset

Outline

- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- **Non-linear visualization techniques**
  - LLE
  - Laplacian eigenmaps
  - TSNE
  - **UMAP**
- Conclusion

UMAP

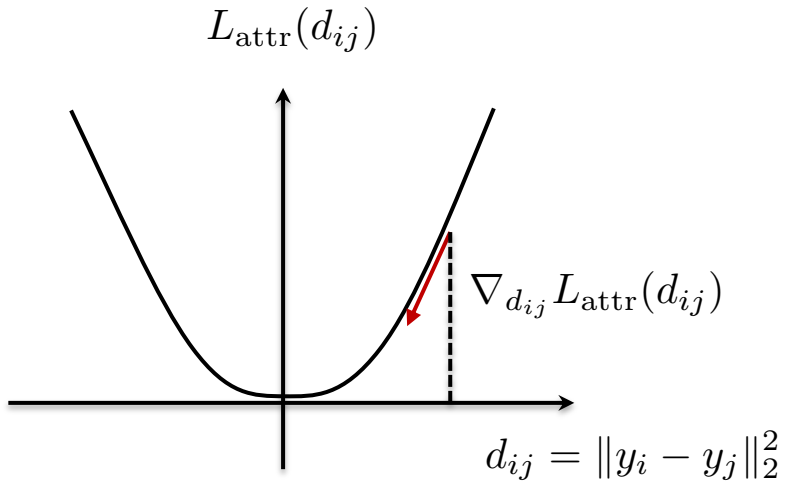

Leland McInnes

- Uniform Manifold Approximation and Projection (UMAP)[1] enhances TSNE in several key aspects.
- Notice that the TSNE gradient can be interpreted as an attractive force between data points, similar to interactions in physics.
- UMAP generalizes this concept by introducing a more flexible attractive force, controlled by two hyperparameters.
- Additionally, UMAP introduces a repulsive force by sampling a few non-neighboring data points.
- Instead of using PCA for initialization in gradient descent, UMAP uses Laplacian Eigenmaps, which offer a more effective starting point.
- To my opinion, both TSNE and UMAP excel in visualization.
- The choice boils down to the implementation with the fastest computational speed.

[1] McInnes et-al, UMAP: Uniform manifold approximation and projection for dimension reduction, 2018

Task formalization

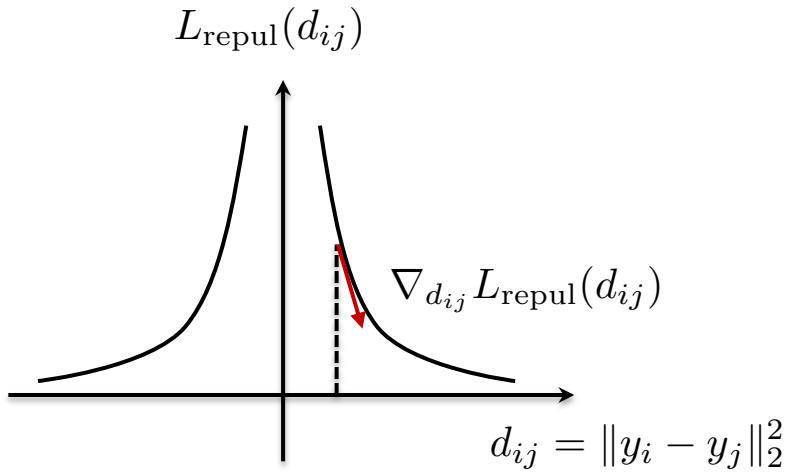
- The visualization task involves minimizing two physics-based losses, each parameterized by the low-dimensional embedding coordinates of the data points.
- One loss function generates attractive forces between closely connected data points on the graph, while the other creates repulsive forces for data points that are far apart on the graph.



Attractive loss

E.g. $L_{\text{attr}}(y) = A_{ij}(1 + a\|y_i - y_j\|_2^2)^b, \; b \geq 2$

$\nabla_y L_{\text{attr}} = 2abA_{ij}(1 + a\|y_i - y_j\|_2^2)^{b-1}(y_i - y_j)$



Repulsive loss

E.g. $L_{\text{repul}}(y) = (1 - A_{ij})(1 + a\|y_i - y_j\|_2^2)^{-b}, \; b \geq 2$

$\nabla_y L_{\text{repul}} = -2ab(1 - A_{ij})\dfrac{1}{(1 + a\|y_i - y_j\|_2^2)^{b+1}}(y_i - y_j)$

Algorithm

- **Minimization problem :**

$$\min_{Y \in \mathbb{R}^{n \times k}} \mathrm{L}_{\mathrm{attr}}(A, Y) + \mathrm{L}_{\mathrm{repul}}(1 - A, Y)$$

1. Compute graph adjacency matrix A with a $k_{\mathcal{G}}$-NN graph
2. Minimize by gradient descent

   Initialization : $Y^{t=0} = \mathrm{LapEig}(X) \in \mathbb{R}^{n \times k}$

   Iterate until convergence, $t = 1, 2, \ldots$:

   $$y_i^{t+1} = y_i^t - \mathrm{lr}\ \left(\nabla_y L_{\mathrm{attr}}(A_{ij}, y_i^t, y_j^t) + \nabla_y L_{\mathrm{repul}}(1 - A_{ij}, y_i^t, y_j^t)\right) \in \mathbb{R}^k$$

   where

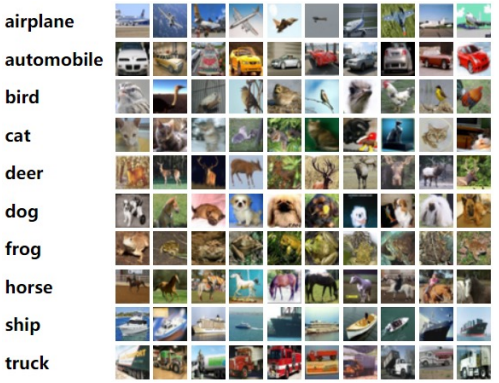
   $$\nabla_y L_{\mathrm{attr}}(A_{ij}, y_i, y_j) = \frac{-2ab\|y_i - y_j\|_2^{2(b-1)}}{1 + \|y_i - y_j\|_2^2} A_{ij}(y_i - y_j) \in \mathbb{R}^k$$

   $$\nabla_y L_{\mathrm{repul}}(1 - A_{ij}, y_i, y_j) = \frac{2b}{(1 + a\|y_i - y_j\|_2^{2b})(\varepsilon + \|y_i - y_j\|_2^2)}(1 - A_{ij})(y_i - y_j) \in \mathbb{R}^k$$

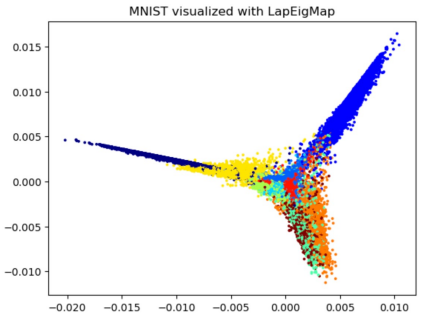
   where a, b are arbitrary hyper-parameters coming from the polynomials

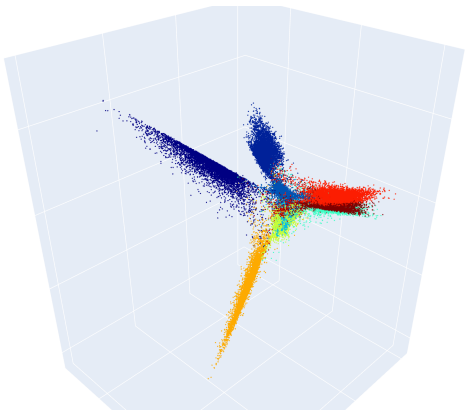
Lab 6 : UMAP

- Run code06.ipynb :
  - Compare UMAP visualization with LapEigenmaps and TSNE on MNIST.
  - Apply UMAP on CIFAR (raw) images.

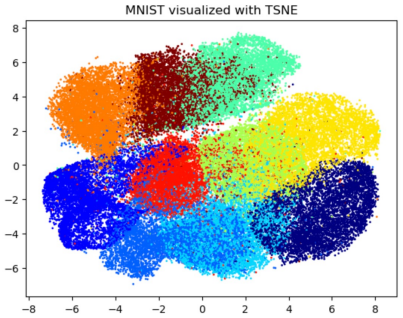


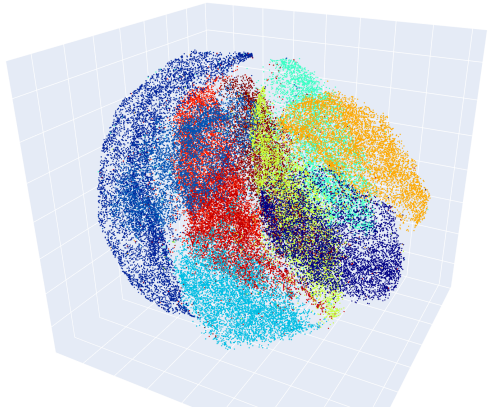
CIFAR dataset



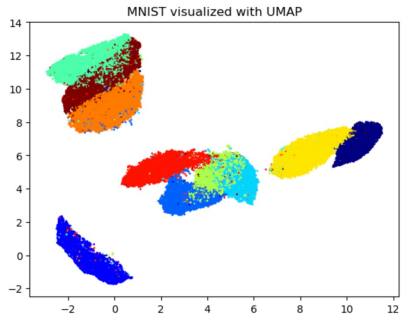


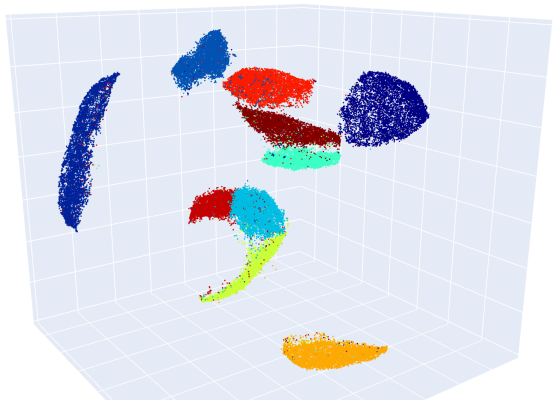
2D and 3D LapEigMaps
of MNIST dataset



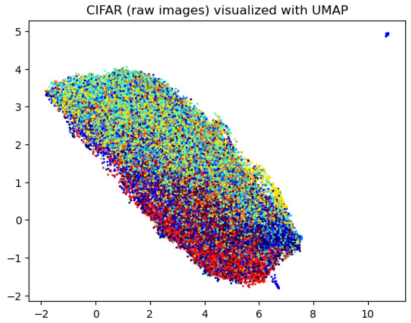


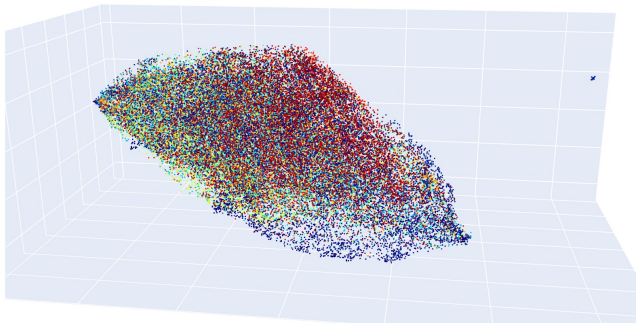
2D and 3D TSNE
of MNIST dataset





2D and 3D UMAP
of MNIST dataset

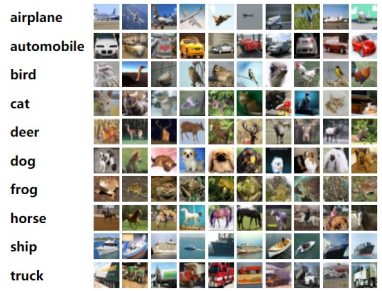




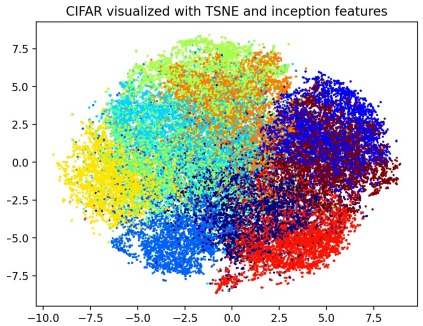
2D and 3D UMAP
of CIFAR dataset

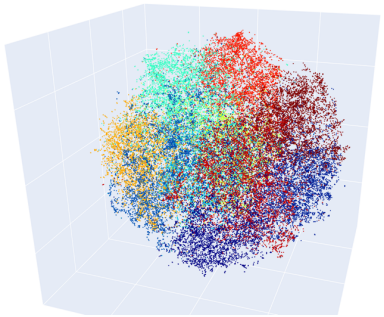
Lab 7 : Visualization with deep learning

- Run code07.ipynb :
  - Visualize CIFAR with TSNE/UMAP and InceptionV3 features.
  - Create a mosaic of CIFAR images.



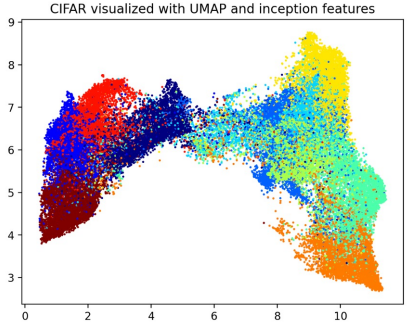
CIFAR dataset

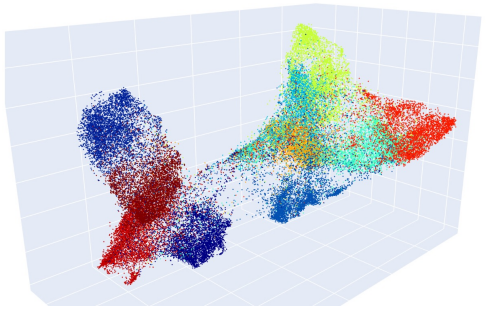




2D and 3D TSNE of CIFAR inception features

Mosaic of CIFAR images with TSNE and inception features

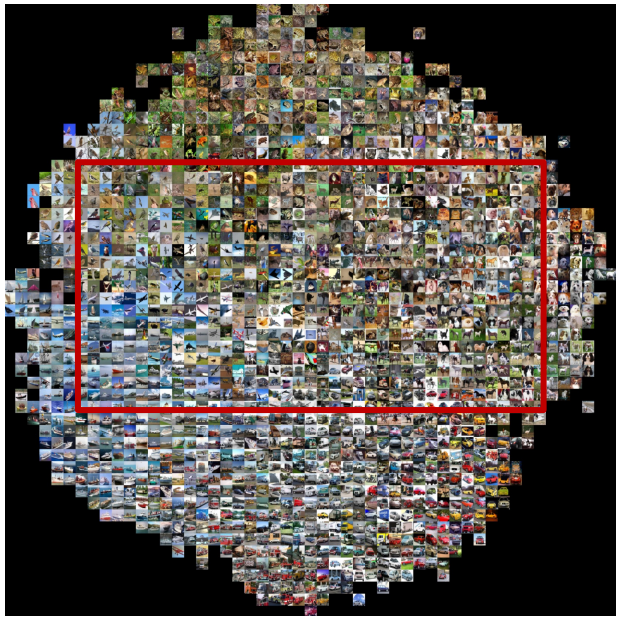




2D and 3D UMAP of CIFAR inception features

Mosaic of CIFAR images with UMAP and inception features

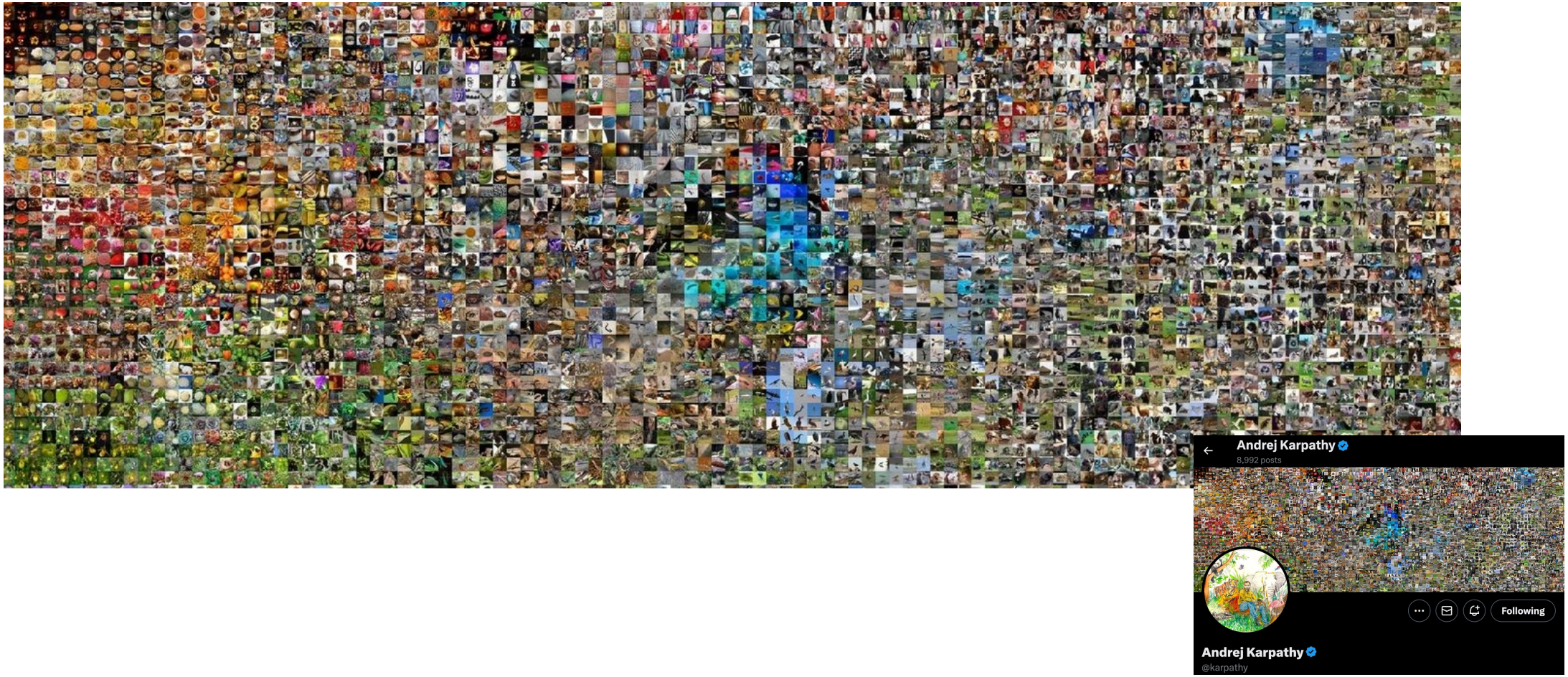
Lab 7 : Visualization with deep learning



Mosaic of CIFAR images with TSNE and inception features

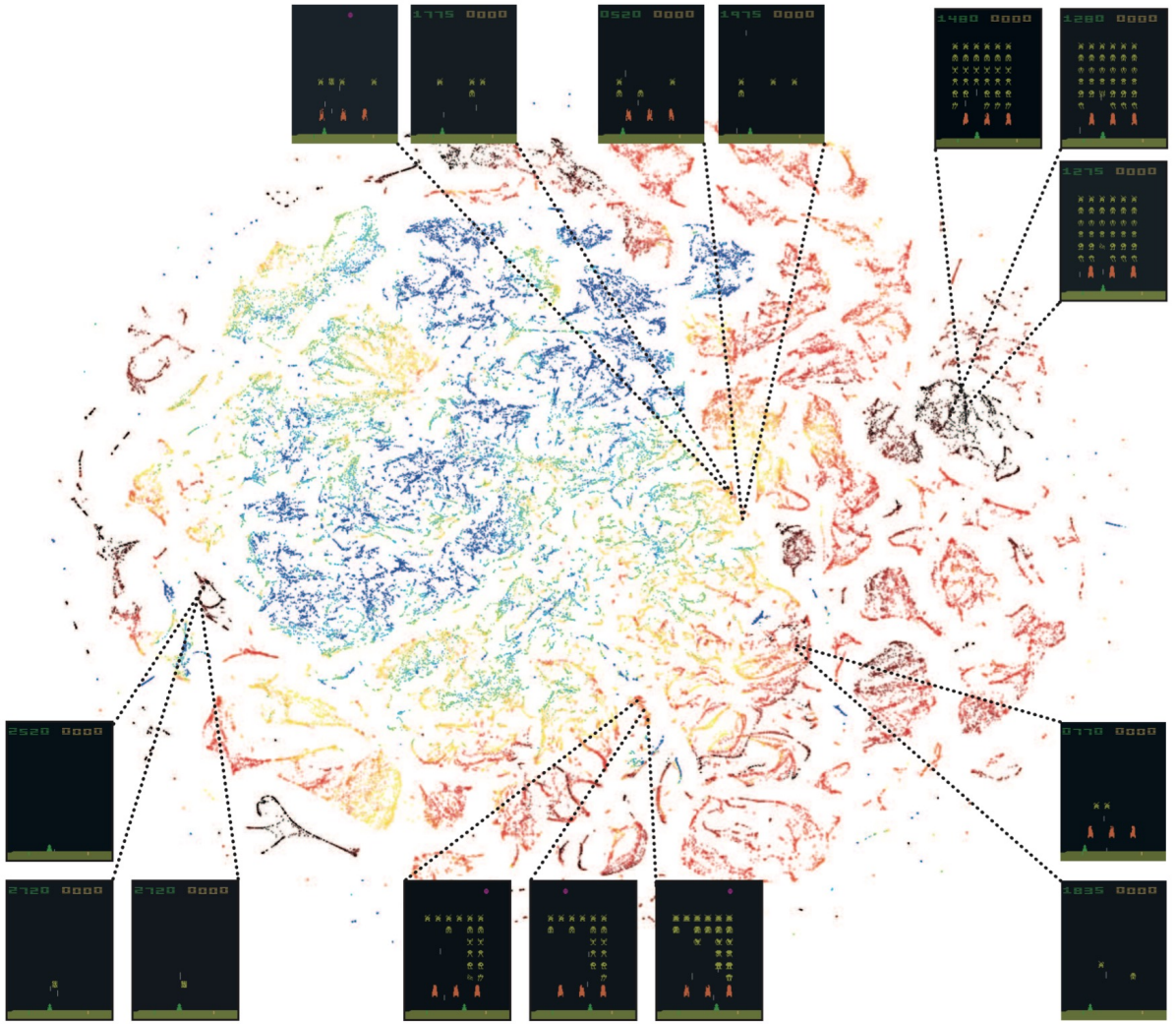
Cropped mosaic of CIFAR images with TSNE and inception features

Visualizing ImageNet dataset



[1] Andrej Karpathy's course cs231n on convolutional neural networks for image recognition

Visualizing video games

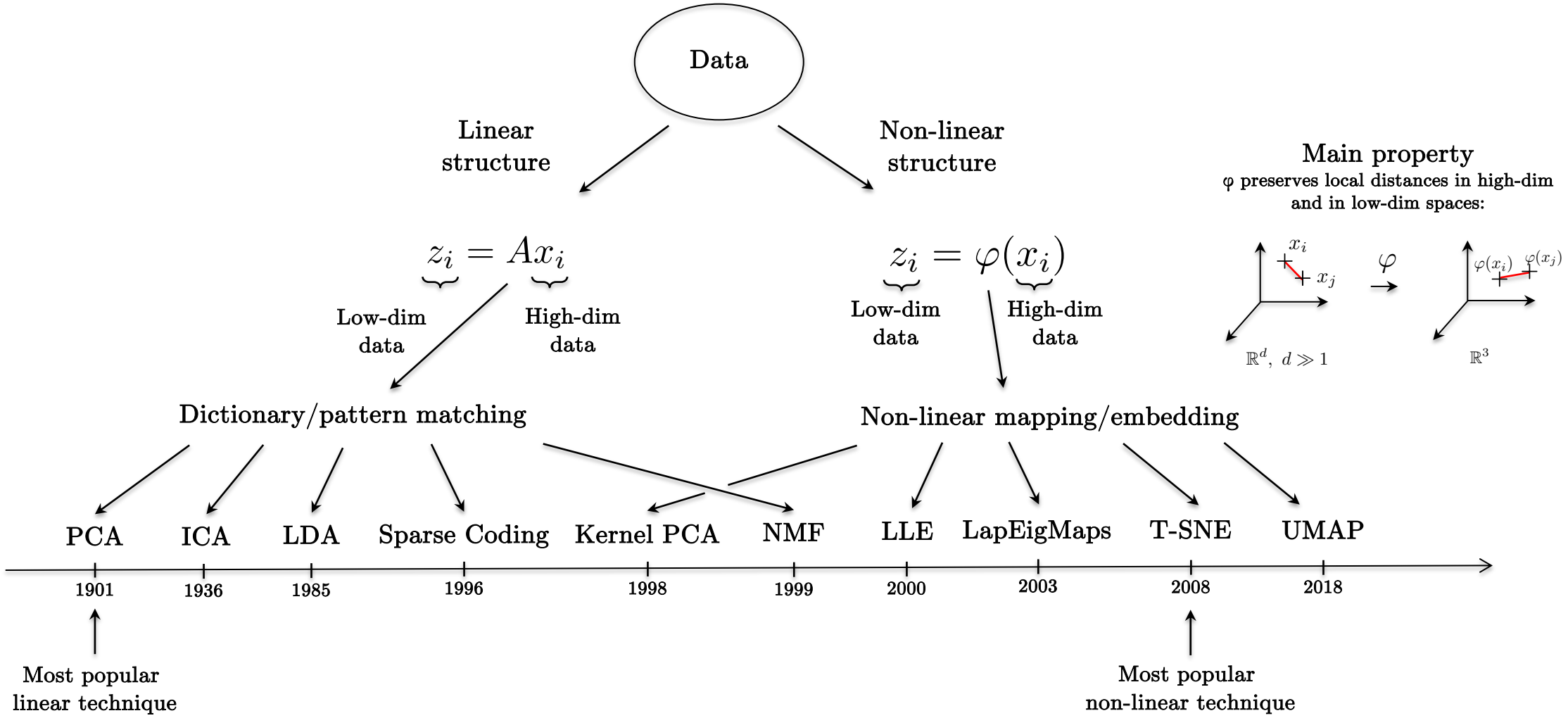


[1] Mnih, Human-level control through deep reinforcement learning, 2015

Outline

- Visualization as dimensionality reduction
- Linear visualization techniques
  - Standard PCA
  - Robust PCA
  - Graph-based PCA
- Non-linear visualization techniques
  - LLE
  - Laplacian eigenmaps
  - TSNE
  - UMAP
- **Conclusion**

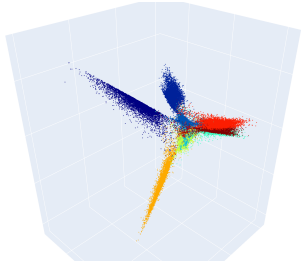
Conclusion


Data
Linear structure
Non-linear structure
$z_i = Ax_i$
Low-dim data
High-dim data
$z_i = \varphi(x_i)$
Low-dim data
High-dim data
Main property
φ preserves local distances in high-dim and in low-dim spaces:
x_i
x_j
φ
$\varphi(x_i)$
$\varphi(x_j)$
$\mathbb{R}^d,\ d \gg 1$
$\mathbb{R}^3$
Dictionary/pattern matching
Non-linear mapping/embedding
PCA
ICA
LDA
Sparse Coding
Kernel PCA
NMF
LLE
LapEigMaps
T-SNE
UMAP
1901
1936
1985
1996
1998
1999
2000
2003
2008
2018
Most popular linear technique
Most popular non-linear technique

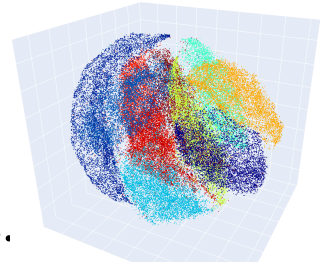
Conclusion

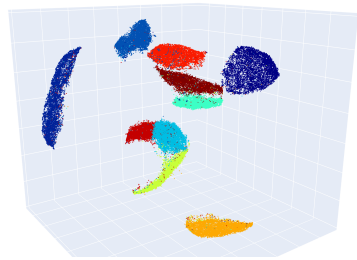
- Non-linear visualization techniques typically involve two key steps :
  - Construct a k-nearest neighbors (kNN) graph from the high-dim data points, and
  - Determine low-dim embedding coordinates, usually in 2D or 3D, that preserve the pairwise distances between the high-dim data points while maintaining a specific visualization property :
    - In spectral-based methods like LLE and Laplacian Eigenmaps, the embedding coordinates Y are orthogonal,
    - In TSNE, the low-dim distribution is optimized to match the high-dim distribution using the Kullback-Leibler (KL) distance,
    - UMAP achieves its embedding Y by balancing attractive and repulsive forces based on the high- and low-dim data representations.

Conclusion



- LLE and LapEigenmaps
  - Offer global solutions but lack flexibility in adjusting the visualization outcome.
  - The spectral orthogonality constraint, needed to avoid trivial solutions, restricts the low-dimensional embeddings.
- TSNE and UMAP
  - Produce local solutions through non-linear loss functions, offering greater flexibility and often more visually appealing results.





  - Without the orthogonality constraint, the class of possible solutions is larger, and more diverse than those provided by spectral methods.
  - UMAP includes two hyperparameters that control the visualization aspect.
  - While this flexibility is advantageous, it also raises the question: what are the optimal UMAP hyperparameter values for visualizing a new dataset?

Conclusion

- No visualization technique, whether linear or non-linear, is universally effective for high-dimensional data due to the curse of dimensionality.
- A key prerequisite for successful visualization is that the input data must be sufficiently expressive.
- In some cases, representing e.g. a text document as a bag of words (a distribution over the dictionary) may work.
- But typically, the most effective visualizations are achieved using the hidden representations from a deep neural network.
- For example, extracting the hidden vector R^{2048} of one of the last layers of an Inception or ResNet network can provide a strong representation of images.
- Similarly, using the memory state vector R^{512} from an RNN can effectively represent a time series, or a class token embedding R^{1024} from the last layer of a Transformer.

Questions?