Numerical Linear Algebra for data-related applications

Yousef Saad
Department of Computer Science and Engineering

University of Minnesota

NA2M-2019 Mohammed V University, Rabat, Morocco

April 5, 2019

Introduction: a historical perspective

In 1953, George Forsythe published a paper titled:

"Solving linear systems can be interesting".

➤ Survey of the state of the art linear algebra at that time: direct methods, iterative methods, conditioning, preconditioning, The Conjugate Gradient method, acceleration methods, ...

➤ An amazing paper in which the author was urging researchers to start looking at solving linear systems

Introduction: a historical perspective

In 1953, George Forsythe published a paper titled:
"Solving linear systems can be interesting".

➤ Survey of the state of the art linear algebra at that time: direct methods, iterative methods, conditioning, preconditioning, The Conjugate Gradient method, acceleration methods, ...

➤ An amazing paper in which the author was urging researchers to start looking at solving linear systems

➤ 66 years later – we can certainly state that:

"Linear Algebra problems in Machine Learning can be interesting"

Focus of numerical linear algebra changed many times over the years

➤ This is because linear algebra is a key tool when solving challenging new problems in various disciplines

1940s–1950s: Major issue: the flutter problem in aerospace engineering $\rightarrow$ eigenvalue problem [cf. Olga Taussky Todd]

➤ Then came the discoveries of the LR and QR algorithms. The package Eispack followed a little later

1960s: Problems related to the power grid promoted what we would call today general sparse matrix techniques

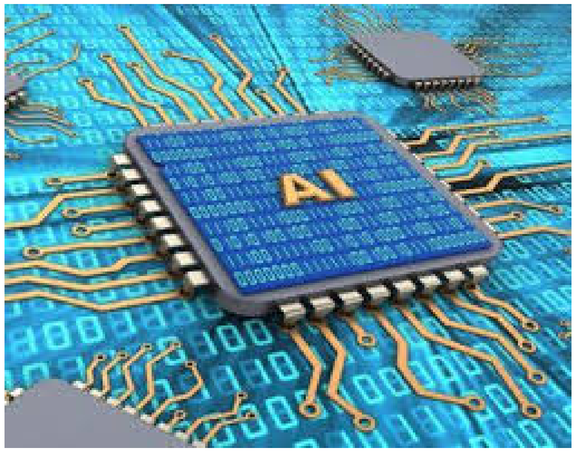
Late 1980s: Thrust on parallel matrix computations.

Late 1990s: Spur of interest in “financial computing”

Solution of PDEs (e.g., Fluid Dynamics) and problems in mechanical eng. (e.g. structures) major force behind numerical linear algebra algorithms in the past few decades.

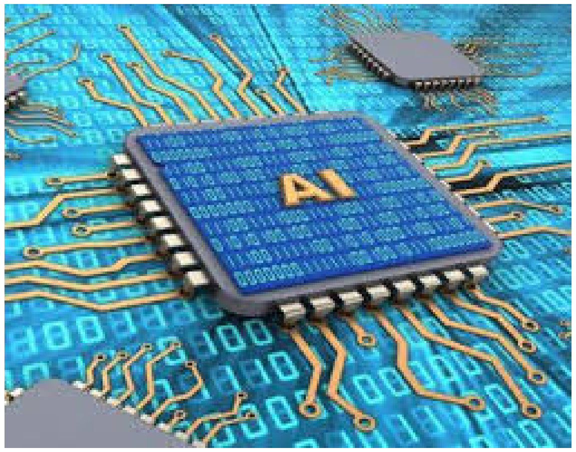
- Strong new forces are now reshaping the field today: Applications related to the use of "data"
- Machine learning is appearing in unexpected places:
  - design of materials
  - machine learning in geophysics
  - self-driving cars, ..
  - ....

Big impact on the economy



- New economy driven by Google, Facebook, Netflix, Amazon, Twitter, Ali-Baba, Tencent, ..., and even the big department stores (Walmart, ...)
- Huge impact on **Jobs**

Big impact on the economy



- ➤ New economy driven by Google, Facebook, Netflix, Amazon, Twitter, Ali-Baba, Tencent, ..., and even the big department stores (Walmart, ...)
- ➤ Huge impact on **Jobs**

- ➤ In contrast: Old economy [driven by Boeing, GM, Ford, Mining industry, US Steel, Aerospatiale, ...] does not have as much to offer...

- ➤ Look at what you are doing under new lenses: DATA

$Ax=b$

$-\Delta u = f$

Graph Partitioning

Preconditioning

Model reduction

$A x = \lambda x$

Domain Decomposition

H2 / HSS matrices

Sparse matrices

LARGE SYSTEMS

$Ax=b$

$-\Delta u = f$

Graph Partitioning

Preconditioning

Model reduction

$A x = \lambda x$

Domain Decomposition

H2 / HSS matrices

Sparse matrices

LARGE SYSTEMS

Translate

$A = U \Sigma V^T$

PCA

Clustering

Dimension Reduction

Graph Laplaceans

Semi–Supervised Learning

Divide & Conquer

Regression LASSO

Data Sparsity

BIG DATA!

Impact on what we teach...

➤ My course: *CSCI 8314: Sparse Matrix Computations*
[url: my website - follow teaching]

... Has changed substantially in past 2–4 years

Before:

—*PDEs, solving linear systems, Sparse direct solvers, Iterative methods, Krylov methods, Preconditioners, Multigrid,..*

$\longrightarrow$

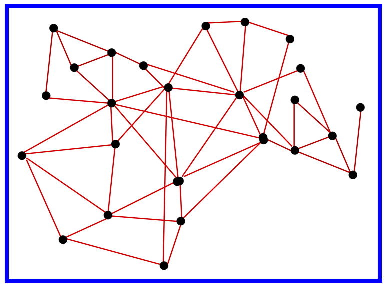
Now:

— a little of sparse direct methods + Applications of graphs, dimension reduction, Krylov methods.. Examples in: PCA, Information retrieval, Segmentation, Clustering, ...

INTRODUCTION: GRAPH LAPLACEANS

Graph Laplaceans - Definition

➤ "Laplace-type" matrices associated with general undirected graphs –



$\longrightarrow L = \left[\quad ? \quad \right]$

➤ Given a graph $G = (V, E)$ define

- A matrix W of weights w_{ij} for each edge with:

$$\boxed{w_{ij} \geq 0, \quad w_{ii} = 0, \quad \text{and} \quad w_{ij} = w_{ji} \, \forall (i, j)}$$

- The diagonal matrix $D = diag(d_i)$ with $d_i = \sum_{j \neq i} w_{ij}$

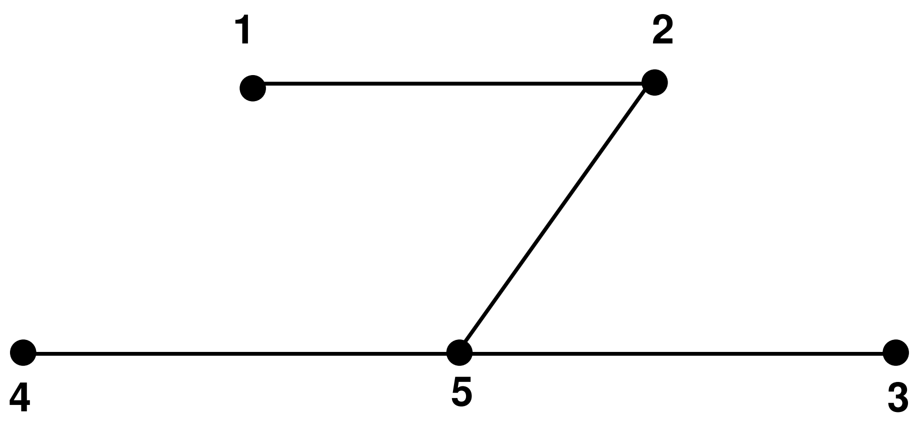
➤ Corresponding *graph Laplacean* of G is:

$$L = D - W$$

- Gershgorin's theorem $\rightarrow$ L is positive semidefinite.
- One eigenvalue equal to zero
- Simplest case:

$$w_{ij} = \begin{cases} 1 \text{ if } (i,j) \in E \ \& \ i \neq j \\ 0 \quad \text{else} \end{cases} \qquad d_i = \sum_{j \neq i} w_{ij}$$

Example: Consider the graph



$$L = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{pmatrix}$$

Bsic results on graph Laplaceans

Proposition:

(i) L is symmetric semi-positive definite.

(ii) L is singular with $\mathbb{1}$ as a null vector.

(iii) If G is connected, then $\mathbf{Null}(L) = \mathbf{span}\{\mathbb{1}\}$

(iv) If G has $k > 1$ connected components $G_1, G_2, \cdots, G_k$, then the nullity of L is k and $\mathbf{Null}(L)$ is spanned by the vectors $z^{(j)}$, $j = 1, \cdots, k$ defined by:

$$(z^{(j)})_i = \begin{cases} 1 \text{ if } i \in G_j \\ 0 \text{ if not.} \end{cases}$$

A few properties of graph Laplaceans

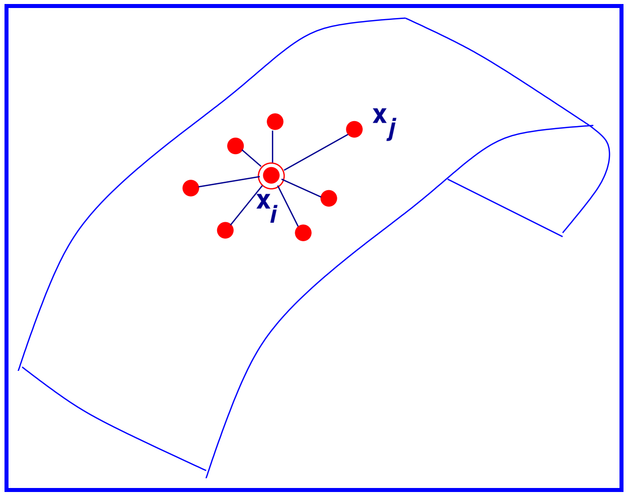
Define: oriented incidence matrix H: (1)First orient the edges $i \sim j$ into $i \to j$ or $j \to i$. (2) Rows of H indexed by vertices of G. Columns indexed by edges. (3) For each (i,j) in E, define the corresponding column in H as $\sqrt{w(i,j)}(e_i - e_j)$.

$\boxed{\text{Example:}}$ In previous example, orient $i \to j$ so that $j > i$ [lower triangular matrix representation]. Then matrix H is: $\longrightarrow$

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -1 & -1 & -1 \end{pmatrix}$$

Property 1 $\boxed{L = HH^T}$

A few properties of graph Laplaceans



Strong relation between $x^T L x$ and local distances between entries of x

➤ Let L = any graph Laplacean

Then:

Property 2: for any $x \in \mathbb{R}^n$:

$$x^\top L x = \sum_{j>i} w_{ij} |x_i - x_j|^2$$

Property 3: (generalization) for any $Y \in \mathbb{R}^{d \times n}$:

$$\text{Tr}\,[YLY^\top] = \sum_{j>i} w_{ij} \|y_i - y_j\|^2$$

➤ Note: y_j = j-th colunm of Y. Usually $d < n$. Each column can represent a data sample.

Property 4: For the particular $L = I - \frac{1}{n} \mathbb{1}\,\mathbb{1}^\top$

$$XLX^\top = \bar{X}\bar{X}^\top == n \times \text{Covariance matrix}$$

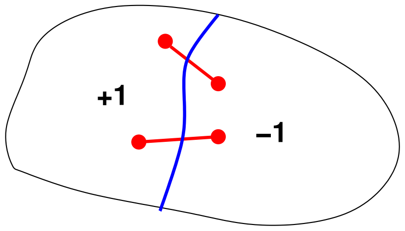
Property 5: L is singular and admits the null vector $\mathbb{1}$ =`ones(n,1)`

Property 6: (Graph partitioning) Consider situation when $w_{ij} \in \{0, 1\}$. If x is a vector of signs (± 1) then

$$x^\top L x = 4 \times \text{('number of edge cuts')}$$

... where edge-cut = pair (i, j) with $x_i \neq x_j$

- Consequence: Can be used to partition graphs....



- ...by minimizing (Lx, x) subject to $x \in \{-1, 1\}^n$ and $\mathbb{1}^T x = 0$ [balanced sets]

$$\min_{x \in \{-1,1\}^n;\ \mathbb{1}^T x = 0} \frac{(Lx, x)}{(x, x)}$$

- This problem is hard [combinatorial] $\rightarrow$

- Instead we solve a relaxed form of this problem :

$$\boxed{\min_{x\in\{-1,1\}^n;\ \mathbb{1}^T x=0} \frac{(Lx,x)}{(x,x)}} \rightarrow \boxed{\min_{x\in\mathbb{R}^n;\ \mathbb{1}^T x=0} \frac{(Lx,x)}{(x,x)}}$$

- Define $v = u_2$ then $lab = sign(v - med(v))$

Background:

- Consider any symmetric (real) matrix A with eigenvalues $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$ and eigenvectors $u_1, \cdots, u_n$

- Recall that:

(Min reached for $x = u_1$)

$$\boxed{\min_{x\in\mathbb{R}^n} \frac{(Ax,x)}{(x,x)} = \lambda_1}$$

- In addition:

(Min reached for $\boldsymbol{x} = \boldsymbol{u}_2$)

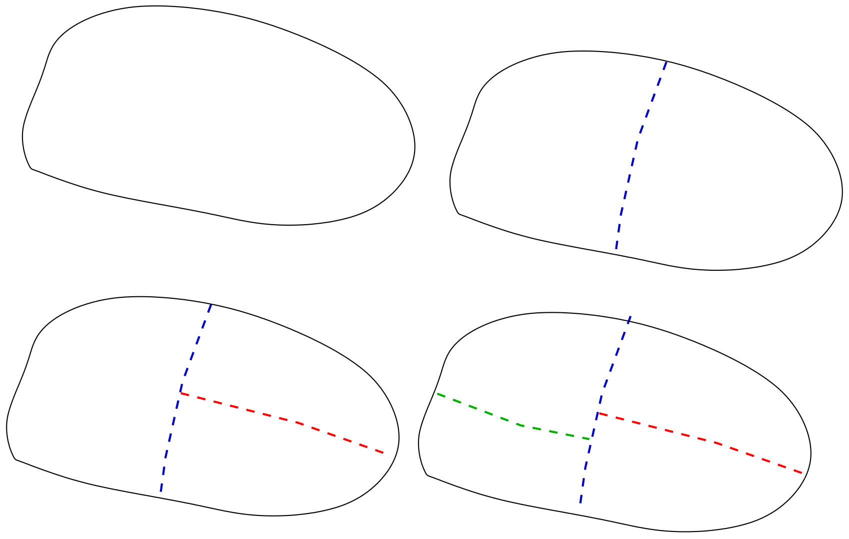
$$\boxed{\min_{x \perp u_1} \frac{(\boldsymbol{Ax}, \boldsymbol{x})}{(\boldsymbol{x}, \boldsymbol{x})} = \boldsymbol{\lambda}_2}$$

- For a graph Laplacean $\boldsymbol{u}_1 = \mathbb{1}$ = vector of all ones and

- ...vector $\boldsymbol{u}_2$ is called the Fiedler vector. It solves the relaxed optimization problem -

Recursive Spectral Bisection

1. Form graph Laplacean
2. Partition graph in 2 based on Fielder vector
3. Partition largest subgraph in two recursively ...
4. ... Until the desired number of partitions is reached

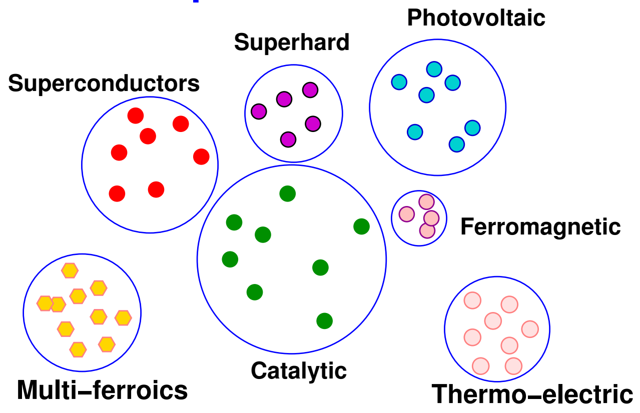


CLUSTERING

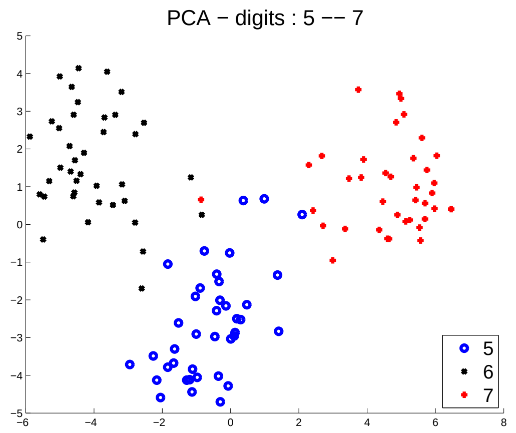
Clustering

➤ Problem: we are given n data items: $x_1, x_2, \cdots, x_n$. Would like to *'cluster'* them, i.e., group them so that each group or cluster contains items that are similar in some sense.

➤ Example: materials



➤ Example: Digits



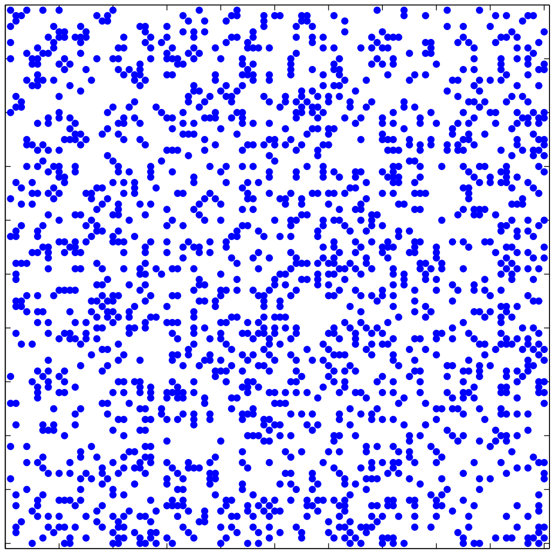
➤ Refer to each group as a 'cluster' or a 'class'

➤ 'Unsupervised learning' : Methods do not exploit labeled data

Example: Community Detection

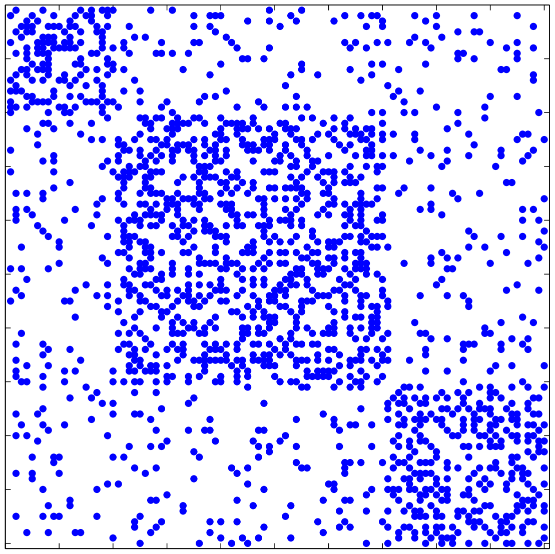
➤ Communities modeled by an ‘affinity’ graph [e.g., ’user A sends frequent e-mails to user B’]

➤ Adjacency Graph represented by a sparse matrix



$\leftarrow$ Original matrix

Goal: Find ordering so blocks are as dense as possible $\rightarrow$



➤ Use ‘blocking’ techniques for sparse matrices

➤ Advantage of this viewpoint: need not know # of clusters.

[data: `www-personal.umich.edu/~mejn/netdata/`]

Example of application

Data set from :

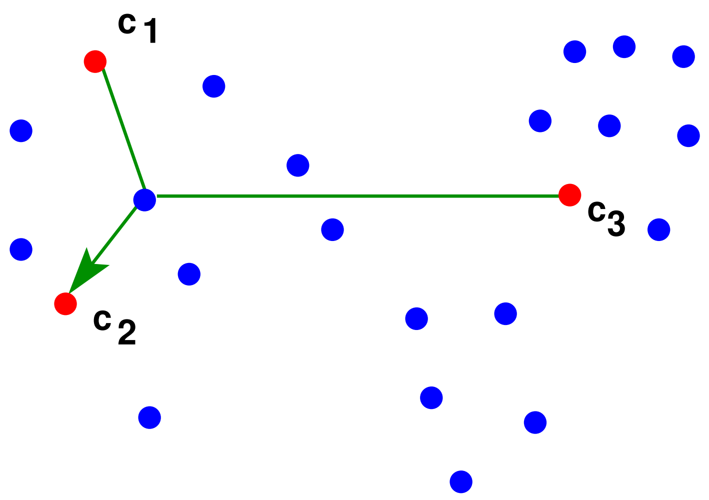
`http://www-personal.umich.edu/~mejn/netdata/`

- Network connecting bloggers of different political orientations [2004 US presidentual election]
- 'Communities': liberal vs. conservative
- Graph: $1,490$ vertices (blogs) : first 758: liberal, rest: conservative.
- Edge: $i \rightarrow j$: a citation between blogs i and j
- Blocking algorithm (Density theshold=0.4): subgraphs [note: density = $|E|/|V|^2$.]
- Smaller subgraph: conservative blogs, larger one: liberals

A basic clustering method: K-means (Background)

- A basic algorithm that uses Euclidean distance

```
1 Select $p$ initial centers: $c_1, c_2, ..., c_p$ for classes $1, 2, \cdots, p$
2 For each $x_i$ do: determine *class* of $x_i$ as $\text{argmin}_k \|x_i - c_k\|$
3 Redefine each $c_k$ to be the centroid of class $k$
4 Repeat until convergence
```



- Simple algorithm
- Works well (gives good results) but can be slow
- Performance depends on initialization

Methods based on similarity graphs

➤ Perform clustering by exploiting a graph that describes the similarities between any two items in the data.

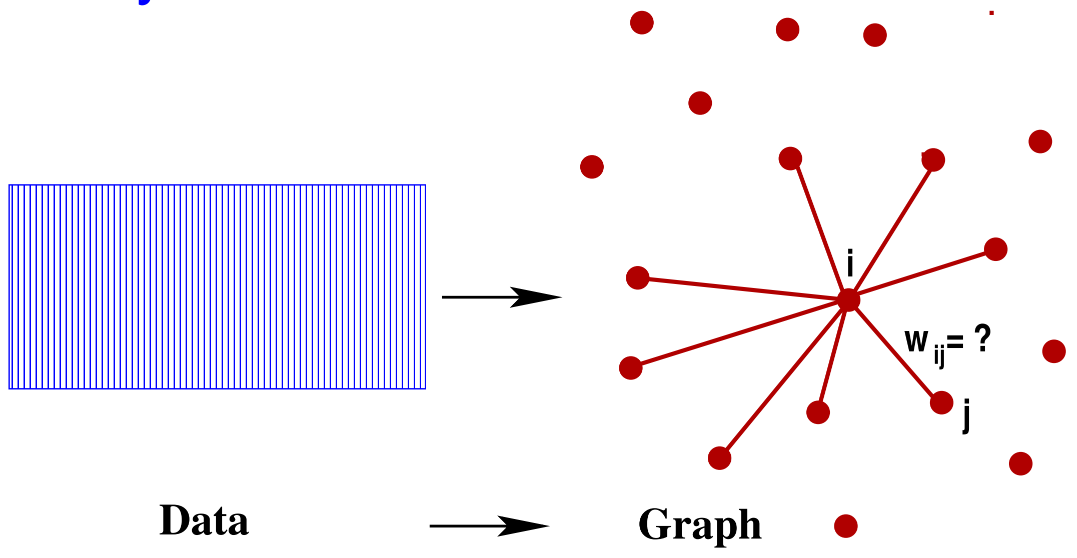
➤ Need to:

1. decide what nodes are in the neighborhood of a given node
2. quantify their similarities - by assigning a weight to any pair of nodes.

$\boxed{\textit{Example:}}$ For text data: Can decide that any columns i and j with a cosine greater than 0.95 are 'similar' and assign that cosine value to w_{ij}

First task: build a 'similarity' graph

Need: a similarity graph, i.e., a graph that captures the similarity between any two items



- For each data item find a small number of its nearest neighbors

- Two techniques are often used:

ϵ-graph: Edges consist of pairs (x_i, x_j) such that $\rho(x_i, x_j) \le \epsilon$

kNN graph: Nodes adjacent to x_i are those nodes x_ℓ with the k with smallest distances $\rho(x_i, x_\ell)$.

- ϵ-graph is undirected and is geometrically motivated. Issues: 1) may result in disconnected components 2) what ϵ?
- kNN graphs are directed in general (can be trivially fixed).
- kNN graphs especially useful in practice.

Similarity graphs: Using 'heat-kernels'

Define weight between i and j as:

$$w_{ij} = f_{ij} \times \begin{cases} e^{\frac{-\|x_i - x_j\|^2}{\sigma_X^2}} & \text{if } \|x_i - x_j\| < r \\ 0 & \text{if not} \end{cases}$$

➤ Note $\|x_i - x_j\|$ could be any measure of distance...

➤ f_{ij} = optional = some measure of similarity - other than distance

➤ Only nearby points kept.

➤ Sparsity depends on parameters

Edge cuts, ratio cuts, normalized cuts, ...

- Assume now that we have built a ‘similarity graph’
- Setting is identical with that of graph partitioning.
- Need a Graph Laplacean: $L = D - W$ with $w_{ii} = 0, w_{ij} \geq 0$ and $D = diag(W * ones(n, 1))$ [in matlab notation]
- Partition vertex set V in two sets A and B with

$$A \cup B = V, \quad A \cap B = \emptyset$$

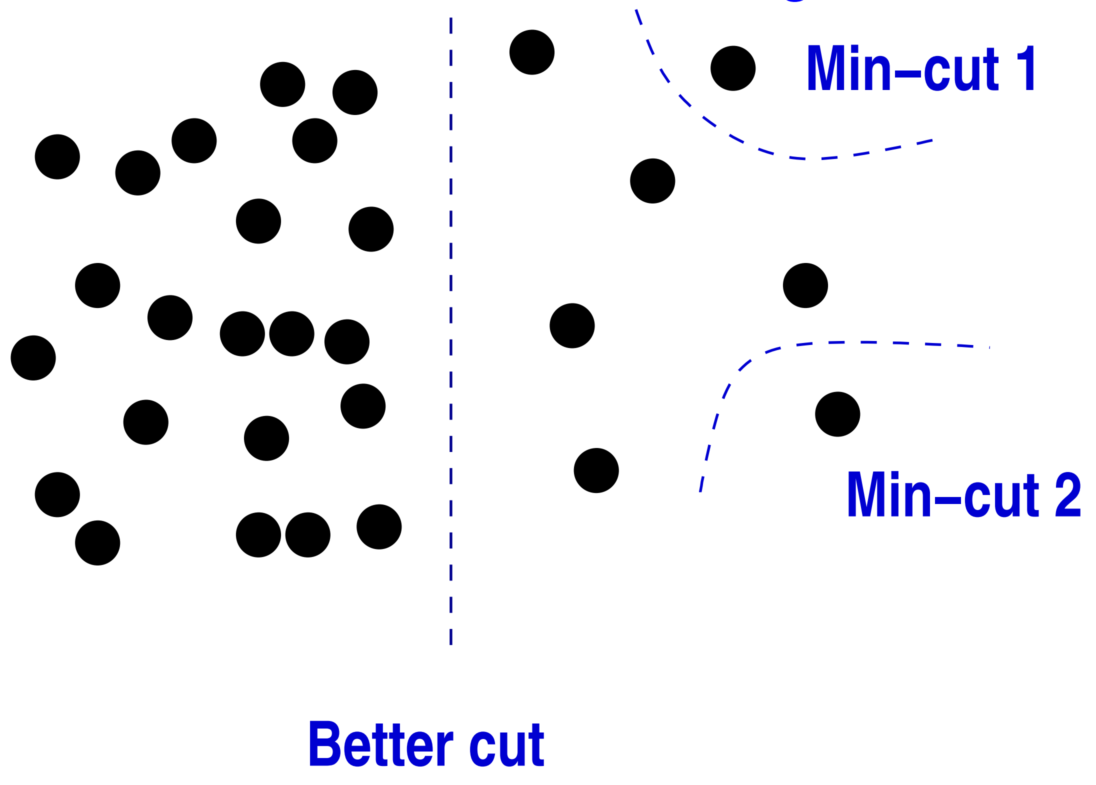
- Define

$$\boxed{cut(A, B) = \sum_{u \in A, v \in B} w(u, v)}$$

- First (naive) approach: use this measure to partition graph, i.e.,

... Find A and B that minimize $cut(A, B)$.

- Issue: Small sets, isolated nodes, big imbalances,



Ratio-cuts

- Standard Graph Partitioning: Find A, B by solving

$$\boxed{\text{Minimize} \quad cut(A,B), \text{ subject to } |A| = |B|}$$

- Condition $|A| = |B|$ not too meaningful in some applications - too restrictive in others.

- Minimum Ratio Cut approach. Find A, B by solving:

$$\boxed{\text{Minimize} \quad \frac{cut(A,B)}{|A|.|B|}}$$

- Difficult to find solution (original paper [Wei-Cheng '91] proposes several heuristics) ➤ Approximate solution : spectral

- Idea: use eigenvector associated with λ_2 to determine partition with heuristics,

Normalized cuts [Shi-Malik,2000]

- Recall notation $w(X, Y) = \sum_{x \in X, y \in Y} w(x, y)$ - then define:

$$\boxed{\text{ncut}(A, B) = \frac{cut(A,B)}{w(A,V)} + \frac{cut(A,B)}{w(B,V)}}$$

- Goal is to avoid small sets A, B
- Let x be an indicator vector:

$$x_i = \begin{cases} 1 \ if \ i \in A \\ 0 \ if \ i \in B \end{cases}$$

- Recall that: $x^T L x = \sum_{(i,j) \in E} w_{ij} |x_i - x_j|^2$ (note: each edge counted once)

- Let

$$\boxed{\begin{aligned} \beta &= \frac{w(A,V)}{w(B,V)} = \frac{x^T D \mathbb{1}}{(\mathbb{1} - x)^T D \mathbb{1}} \\ y &= x - \beta(\mathbb{1} - x) \end{aligned}}$$

- Then we need to solve:

$$\boxed{\begin{array}{ll} \min\limits_{y_i \ \{0,-\beta\}} & \dfrac{y^T L y}{y^T D y} \\ \text{Subject to} & y^T D \mathbb{1} = 0 \end{array}}$$

- + Relax $\rightarrow$ need to solve Generalized eigenvalue problem

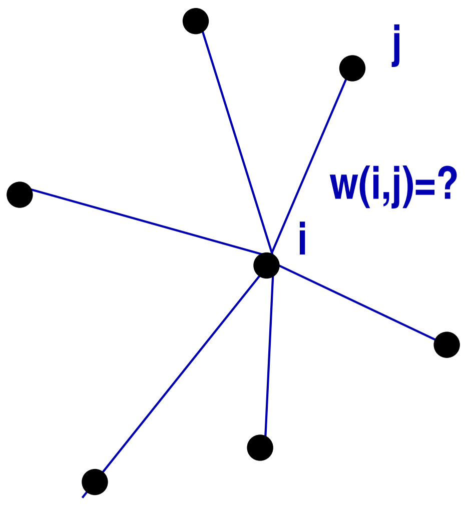
$$Ly = \lambda D y$$

- $y_1 = \mathbb{1}$ is eigenvector associated with eigenvalue $\lambda_1 = 0$
- y_2 associated with second eigenvalue solves problem.

Spectral clustering: General approach

1 Given: Collection of data samples $\{x_1, x_2, \cdots, x_n\}$

2 Build a similarity graph between items



3 Compute (smallest) eigenvector (s) of resulting graph Laplacean

4 Use k-means on eigenvector (s) of Laplacean

➤ For Normalized cuts solve generalized eigen problem.

Application: Image segmentation

- First task: obtain a weighted graph from pixels.
- Common idea: use “Heat kernels”
- Let F_j = feature value (e.g., brightness), and Let X_j = spatial position.

Then define

$$w_{ij} = e^{\frac{-\|F_i - F_j\|^2}{\sigma_I^2}} \times \begin{cases} e^{\frac{-\|X_i - X_j\|^2}{\sigma_X^2}} & \text{if} \|X_i - X_j\| < r \\ 0 & \text{else} \end{cases}$$

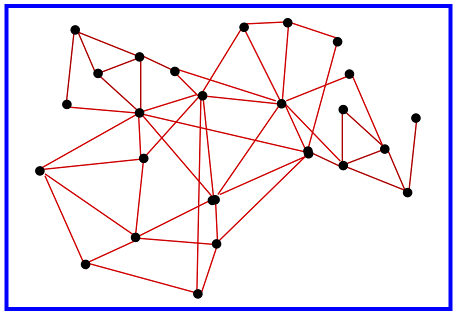
- Sparsity depends on parameters

GRAPH EMBEDDINGS

Graph embeddings

- We have seen how to build a graph to represent data
- *Graph embedding* does the opposite: maps a graph to data

Given: a graph that models some data (e.g., a kNN graph)



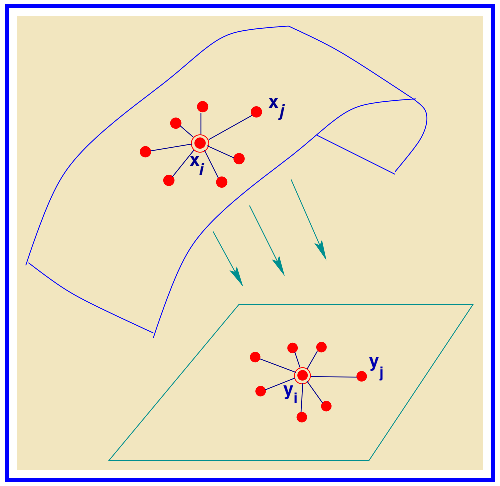
$\longrightarrow$ Data: $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_n]$ in $\mathbb{R}^d$

- Trivial use: visualize a graph ($\boldsymbol{d} = \boldsymbol{2}$)
- Wish: mapping should preserve *similarities* in graph.

➤ Many applications [clustering, finding missing link, semi-supervised learning, community detection, ...]

➤ Graph captures similarities, closeness, ..., in data

Objective: Build a mapping of each vertex $\boldsymbol{i}$ to a data point $\boldsymbol{y_i} \in \mathbb{R}^d$



➤ Many methods do this

➤ Eigenmaps and LLE are two of the best known

- Eigenmaps uses the *graph Laplacean*
- Recall: Graph Laplacean is a matrix defined by :

$$\boxed{L = D - W}$$

$$\begin{cases} w_{ij} \geq 0 \text{ if } j \in Adj(i) \\ w_{ij} = 0 \quad \text{else} \end{cases} \qquad D = \text{diag}\left[d_{ii} = \sum_{j \neq i} w_{ij} \right]$$

with $Adj(i)$ = neighborhood of i (excludes i)

- Remember that vertex i represents data item x_i. We will use i or x_i to refer to the vertex.
- We will find the y_i's by solving an optimization problem.

The Laplacean eigenmaps approach

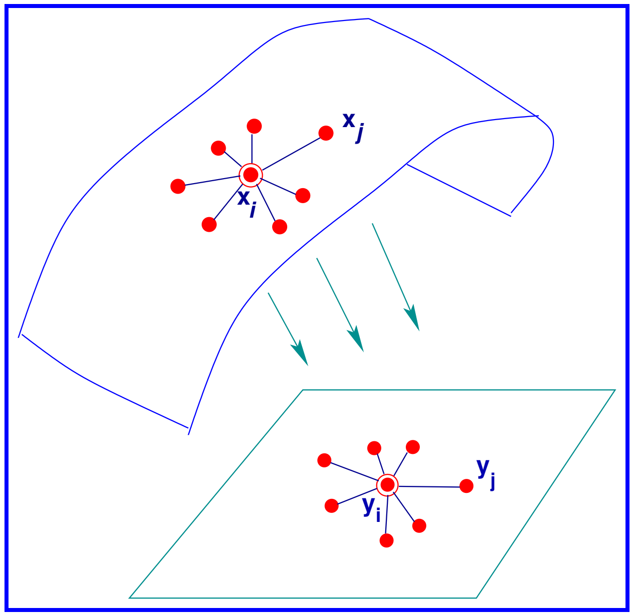
Laplacean Eigenmaps [Belkin-Niyogi '01] *minimizes*

$$\mathcal{F}(Y) = \sum_{i,j=1}^{n} w_{ij} \|y_i - y_j\|^2 \quad \text{subject to} \quad Y D Y^\top = I$$

Motivation: if $\|x_i - x_j\|$ is small (orig. data), we want $\|y_i - y_j\|$ to be also small (low-Dim. data)

➤ Data used indirectly through graph

➤ Objective function leads to a trace and yields a sparse eigenvalue problem



- Problem translates to:

$$\min_{\begin{cases} Y \in \mathbb{R}^{d\times n} \\ Y D Y^\top = I \end{cases}} \text{Tr}\left[Y(D-W)Y^\top\right] .$$

- Solution (sort eigenvalues increasingly):

$$(D-W)u_i = \lambda_i D u_i \; ; \quad y_i = u_i^\top ; \quad i = 1, \cdots, d$$

- An $n \times n$ sparse eigenvalue problem [In 'sample' space]

- Note: can assume $D = I$. Amounts to rescaling data. Problem becomes

$$\boxed{(I-W)u_i = \lambda_i u_i \; ; \quad y_i = u_i^\top ; \quad i = 1, \cdots, d}$$

Locally Linear Embedding (Roweis-Saul-00)

➤ LLE is very similar to Eigenmaps. Main differences:

1) Graph Laplacean matrix is replaced by an 'affinity' graph

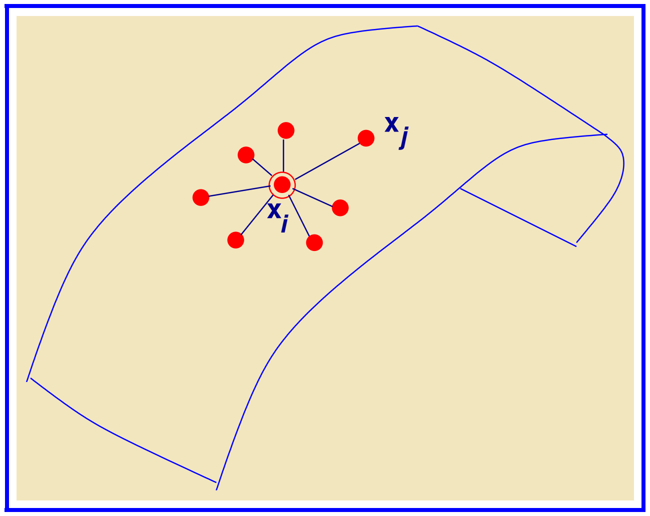
2) Objective function is changed: want to preserve graph

1. Graph: Each x_i is written as a convex combination of its k nearest neighbors:

$x_i \approx \Sigma w_{ij} x_j, \quad \sum_{j \in N_i} w_{ij} = 1$

➤ Optimal weights computed ('local calculation') by minimizing

$\|x_i - \Sigma w_{ij} x_j\| \quad$ for $\quad i = 1, \cdots, n$



2. Mapping:

The y_i's should obey the same 'affinity' as x_i's $\rightsquigarrow$

Minimize:

$$\sum_i \left\| y_i - \sum_j w_{ij} y_j \right\|^2 \quad \text{subject to:} \quad Y\mathbb{1} = 0, \quad YY^\top = I$$

Solution:

$$\boxed{(I - W^\top)(I - W)u_i = \lambda_i u_i; \qquad y_i = u_i^\top.}$$

➤ $(I - W^\top)(I - W)$ replaces the graph Laplacean of eigenmaps

More recent methods

➤ Quite a bit of recent work - methods: node2vec, DeepWalk, GraRep, ... Papers, see e.g.,:

[1] *W. L. Hamilton, R. Ying, and J. Leskovec* *Representation Learning on Graphs: Methods and Applications* arXiv: 1709. 05584v3 (2018)

[2] *S. Cao, W. Lu, and Q. Xu* *GraRep: Learning Graph Representations with Global Structural Information*, CIKM, ACM Conf. on Inform. and Knowledge Managt, 24 (2015)

[3] *A. Ahmed, N. Shervashidze, and S. Narayanamurthy*, *Distributed Large-scale Natural Graph Factorization* [Proc. WWW 2013, May 13-17, 2013, Rio de Janeiro, Brazil]

Example: Graph factorization

- Line of work in Papers [1] and [3] above + others
- Instead of minimizing $\sum w_{ij} \|y_i - y_j\|_2^2$ as before

... try to minimize
$$\boxed{\sum_{ij} |w_{ij} - y_i^T y_j|^2}$$

- In other words solve: $\min_Y \|W - Y^T Y\|_F^2$
- Referred to as *Graph factorization*
- Common in knowledge graphs

DIMENSION REDUCTION

Major tool of Data Mining: Dimension reduction

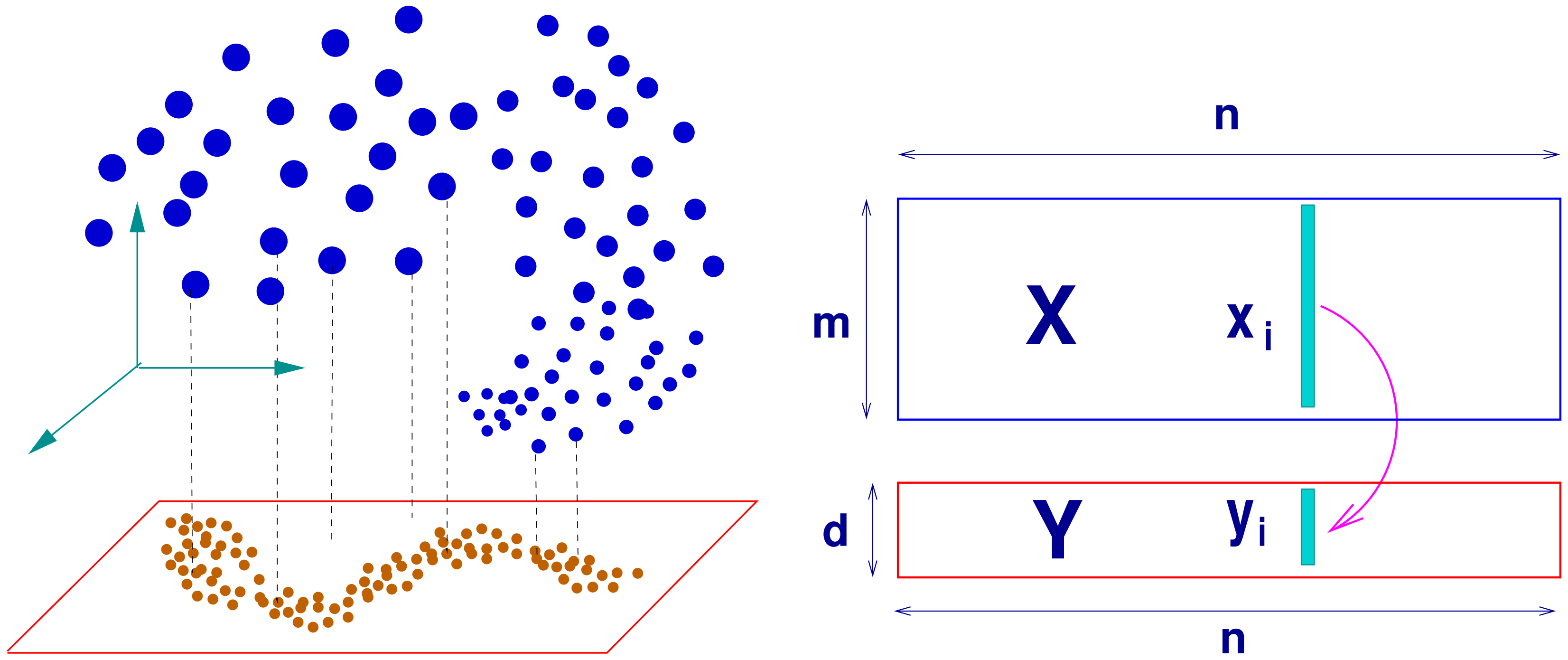
- Eigenmaps and LLE are a form of dimension reduction:

$$\text{Data in } \mathbb{R}^m \to \text{graph} \to \text{Data in } \mathbb{R}^d$$

Dimenson reduction: Given: $X = [x_1, \cdots, x_n] \in \mathbb{R}^{m \times n}$, find a low-dimens. representation $Y = [y_1, \cdots, y_n] \in \mathbb{R}^{d \times n}$ of X

- Achieved by a mapping $\Phi : x \in \mathbb{R}^m \longrightarrow y \in \mathbb{R}^d$ so:

$$\phi(x_i) = y_i, \quad i = 1, \cdots, n$$



- Φ may be linear : $\boldsymbol{y_j = W^\top x_j, \forall j}$, *or,* $\boldsymbol{Y = W^\top X}$

- ... or nonlinear (implicit).

- Mapping Φ required to: Preserve proximity? Maximize variance? Preserve a certain graph?

Basics: Principal Component Analysis (PCA)

In *Principal Component Analysis* W is computed to maximize variance of projected data:

$$\max_{W \in \mathbb{R}^{m \times d}; W^\top W = I} \quad \sum_{i=1}^{n} \left\| y_i - \frac{1}{n} \sum_{j=1}^{n} y_j \right\|_2^2, \quad y_i = W^\top x_i.$$

➤ Leads to maximizing

$$\text{Tr}\left[W^\top (X - \mu e^\top)(X - \mu e^\top)^\top W\right], \quad \mu = \tfrac{1}{n}\Sigma_{i=1}^{n} x_i$$

➤ Solution $W = \{$ dominant eigenvectors $\}$ of the covariance matrix $\equiv$ Set of left singular vectors of $\bar{X} = X - \mu e^\top$

SVD:

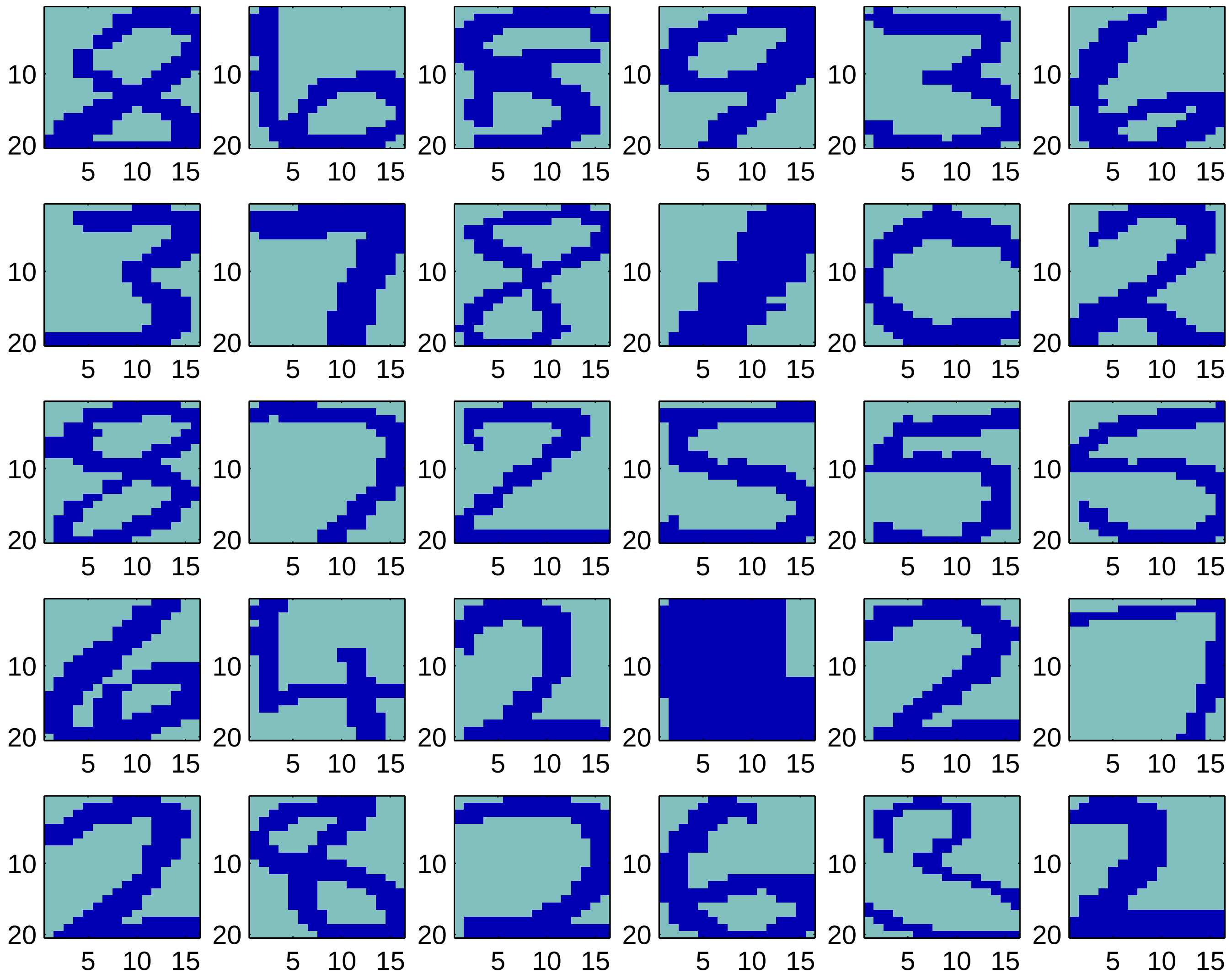
$$\bar{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top, \quad \boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{I}, \quad \boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{I}, \quad \boldsymbol{\Sigma} = \text{Diag}$$

- Optimal $\boldsymbol{W} = \boldsymbol{U}_d \equiv$ matrix of first $\boldsymbol{d}$ columns of $\boldsymbol{U}$
- Solution $\boldsymbol{W}$ also minimizes 'reconstruction error' ..

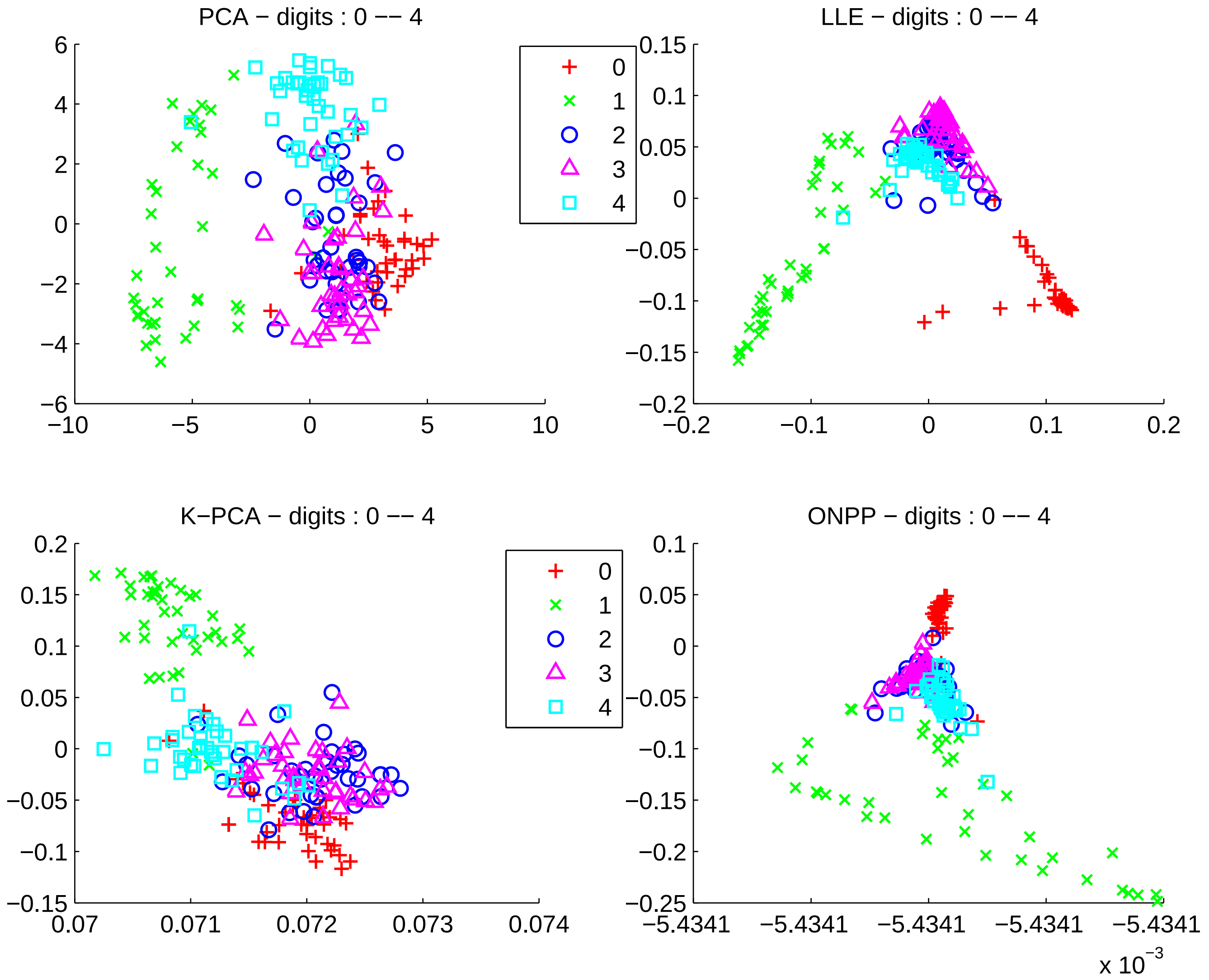
$$\sum_i \|\boldsymbol{x}_i - \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{x}_i\|^2 = \sum_i \|\boldsymbol{x}_i - \boldsymbol{W}\boldsymbol{y}_i\|^2$$

- In some methods recentering to zero is not done, i.e., $\bar{\boldsymbol{X}}$ replaced by $\boldsymbol{X}$.

Example: Digit images (a random sample of 30)



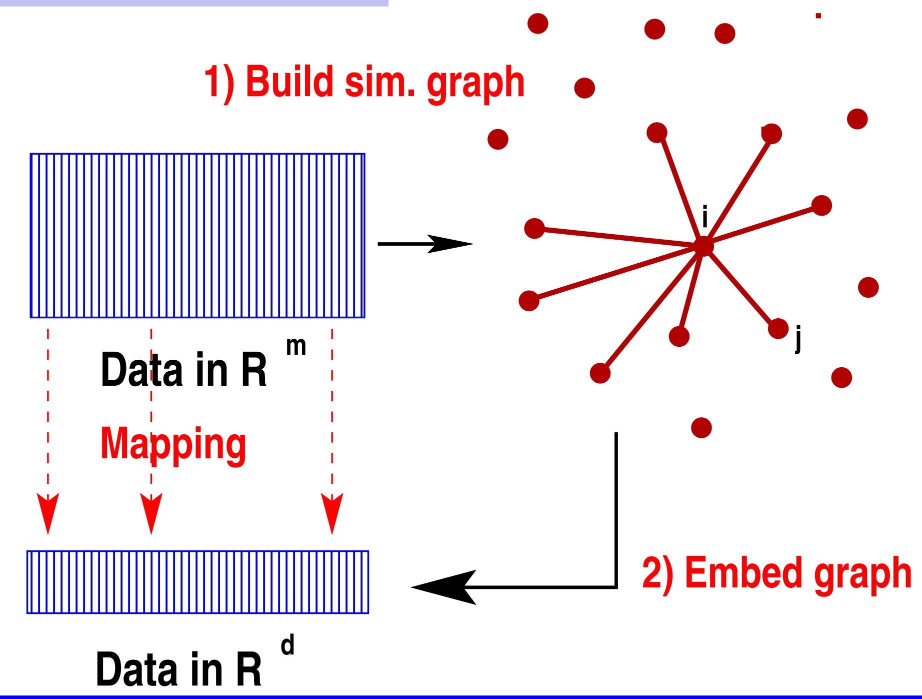
2-D 'reductions':



Graph-based dimension reduction

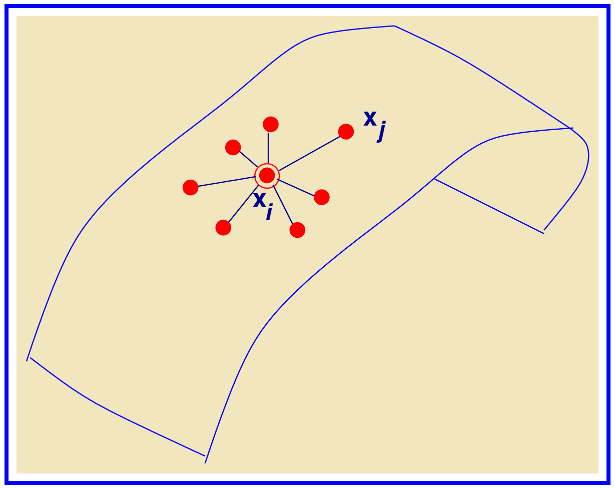
➤ A class of methods that exploit graphs to perform Dimensionality reduction

General Approach:



- Start with a graph of data. e.g.: graph of k nearest neighbors (k-NN graph)

Want: Perform a projection which preserves the graph in some sense



- Define a *graph Laplacean:*

$$L = D - W$$

e.g.,: $w_{ij} = \begin{cases} 1 & \text{if } j \in Adj(i) \\ 0 & \text{else} \end{cases}$ $\quad D = \text{diag}\left[d_{ii} = \sum_{j \neq i} w_{ij} \right]$

with $Adj(i)$ = neighborhood of i (excluding i)

- We have two methods: *Eigenmaps* and *LLE*

Explicit (linear) vs. Implicit (nonlinear) mappings:

➤ In PCA the mapping $\mathbf{\Phi}$ from high-dimensional space ($\mathbb{R}^m$) to low-dimensional space ($\mathbb{R}^d$) is explicitly known:

$$\boldsymbol{y} = \mathbf{\Phi}(\boldsymbol{x}) \equiv \boldsymbol{V}^T \boldsymbol{x}$$

➤ In Eigenmaps and LLE we only know

$$\boldsymbol{y}_i = \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{i} = \mathbf{1}, \cdots, \boldsymbol{n}$$

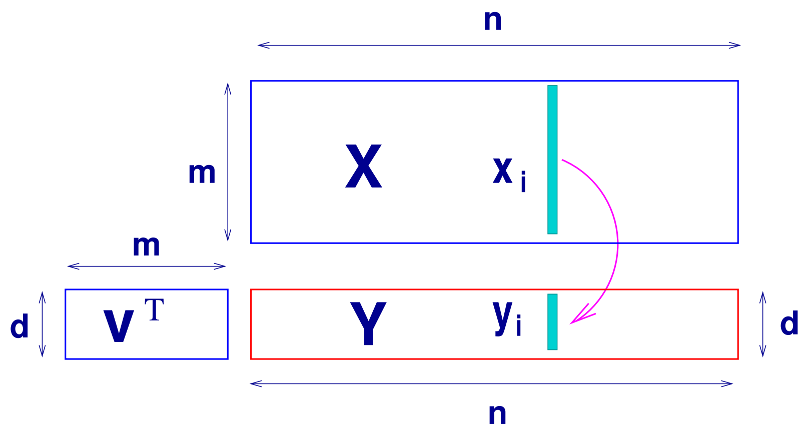
➤ Mapping ϕ is now implicit: Very difficult to compute $\phi(x)$ for an x that is not in the sample (i.e., not one of the x_i's)

➤ Inconvenient for classification. Thus is known as the "The out-of-sample extension" problem

Locally Preserving Projections (He-Niyogi-03)

- LPP is a linear dimensionality reduction technique

- Recall the setting:
Want $\boldsymbol{V} \in \mathbb{R}^{m\times d}$; $\boldsymbol{Y} = \boldsymbol{V}^\top \boldsymbol{X}$



- Starts with the same neighborhood graph as Eigenmaps:
$\boldsymbol{L} \equiv \boldsymbol{D} - \boldsymbol{W}$ = graph 'Laplacean'; with $\boldsymbol{D} \equiv \boldsymbol{diag}(\{\Sigma_i w_{ij}\})$.

➤ Optimization problem is to solve

$$\min_{Y \in \mathbb{R}^{d\times n},\ YDY^\top = I} \Sigma_{i,j} w_{ij} \|y_i - y_j\|^2, \quad Y = V^\top X.$$

➤ Difference with eigenmaps: Y is a projection of X data

➤ Solution (sort eigenvalues increasingly)

$$XLX^\top v_i = \lambda_i XDX^\top v_i \quad y_{i,:} = v_i^\top X$$

➤ Note: essentially same method in [Koren-Carmel'04] called 'weighted PCA' [viewed from the angle of improving PCA]

ONPP (Kokiopoulou and YS '05)

➤ Orthogonal Neighborhood Preserving Projections

➤ A linear (orthogonoal) version of LLE obtained by writing $\boldsymbol{Y}$ in the form $\boldsymbol{Y} = \boldsymbol{V}^\top \boldsymbol{X}$

➤ Same graph as LLE. Objective: preserve the affinity graph (as in LEE) *but* with the constraint $\boldsymbol{Y} = \boldsymbol{V}^\top \boldsymbol{X}$

➤ Problem solved to obtain mapping:

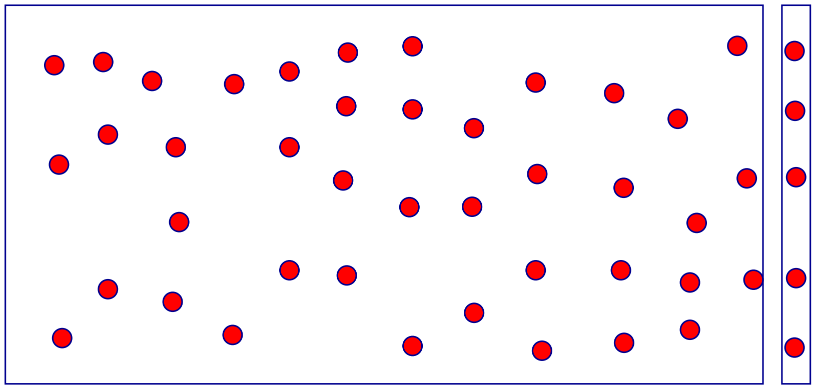
$$\min_{\boldsymbol{V}} \text{Tr}\left[\boldsymbol{V}^\top \boldsymbol{X}(\boldsymbol{I} - \boldsymbol{W}^\top)(\boldsymbol{I} - \boldsymbol{W})\boldsymbol{X}^\top \boldsymbol{V}\right]$$

s.t. $\boldsymbol{V}^T \boldsymbol{V} = \boldsymbol{I}$

➤ In LLE replace $\boldsymbol{V}^\top \boldsymbol{X}$ by $\boldsymbol{Y}$

Application: Information Retrieval

- Given: collection of documents (columns of a matrix A) and a query vector q.
- Representation: $m \times n$ term by document matrix



- A query q is a (sparse) vector in $\mathbb{R}^m$ ('pseudo-document')

Problem: find a column of A that best matches q

- *Vector space model:* use $\cos\langle(A(:,j),q), j = 1:n$
- Requires the computation of A^Tq
- Literal Matching $\rightarrow$ ineffective

Common approach: Dimension reduction (SVD)

➤ LSI: replace A by a low rank approximation [from SVD]

$$A = U\Sigma V^T \quad \rightarrow \quad A_k = U_k \Sigma_k V_k^T$$

➤ Replace similarity vector: $s = A^T q$ by $s_k = A_k^T q$

➤ Main issues: 1) computational cost 2) Updates

Idea: Replace A_k by $A\phi(A^T A)$, where ϕ == a filter function

Consider the step-function (Heaviside):

$$\phi(x) = \begin{cases} 0, & 0 \le x \le \sigma_k^2 \\ 1, & \sigma_k^2 \le x \le \sigma_1^2 \end{cases}$$

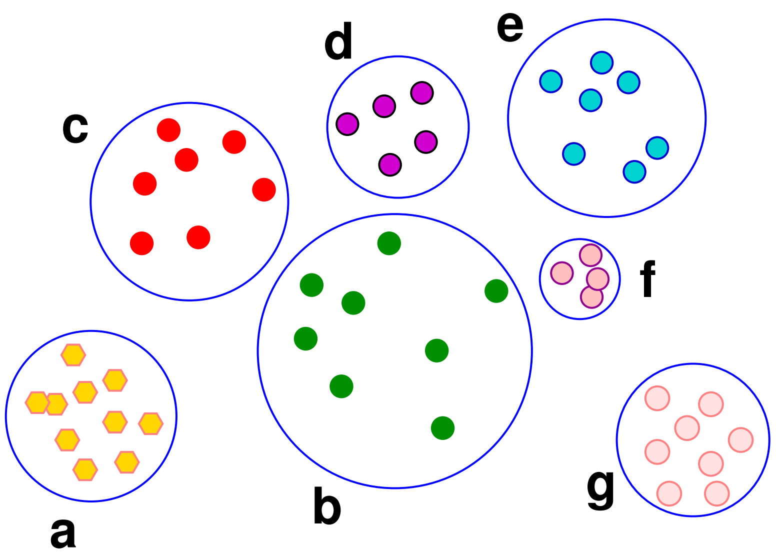
➤ Would yield the same result as TSVD but not practical

SUPERVISED LEARNING

Supervised learning

➤ We now have data that is ‘labeled’

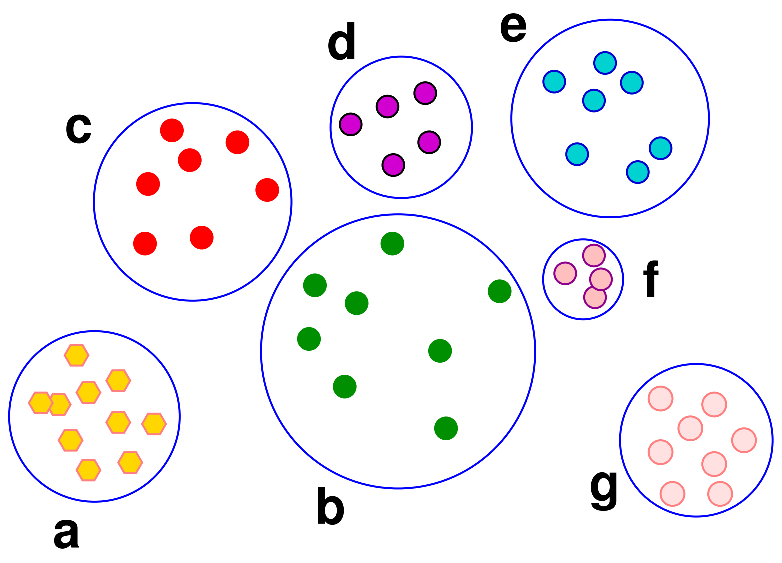
- Example: (health sciences) ‘malignant’- ’non malignant’
- Example: (materials) ’photovoltaic’, ’hard’, ’conductor’, ...
- Example: (Digit recognition) Digits ’0’, ’1’, ..., ’9’

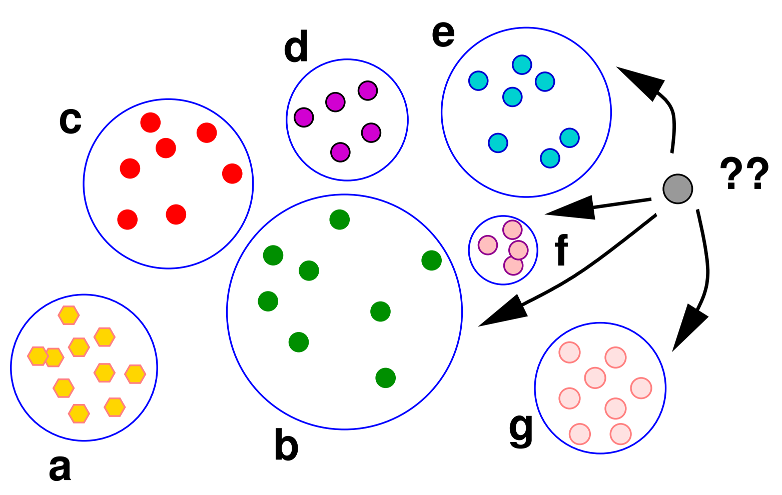


Supervised learning

We now have data that is ‘labeled’

- Example: (health sciences) ‘malignant’- ’non malignant’
- Example: (materials) ’photovoltaic’, ’hard’, ’conductor’, ...
- Example: (Digit recognition) Digits ’0’, ’1’, ..., ’9’

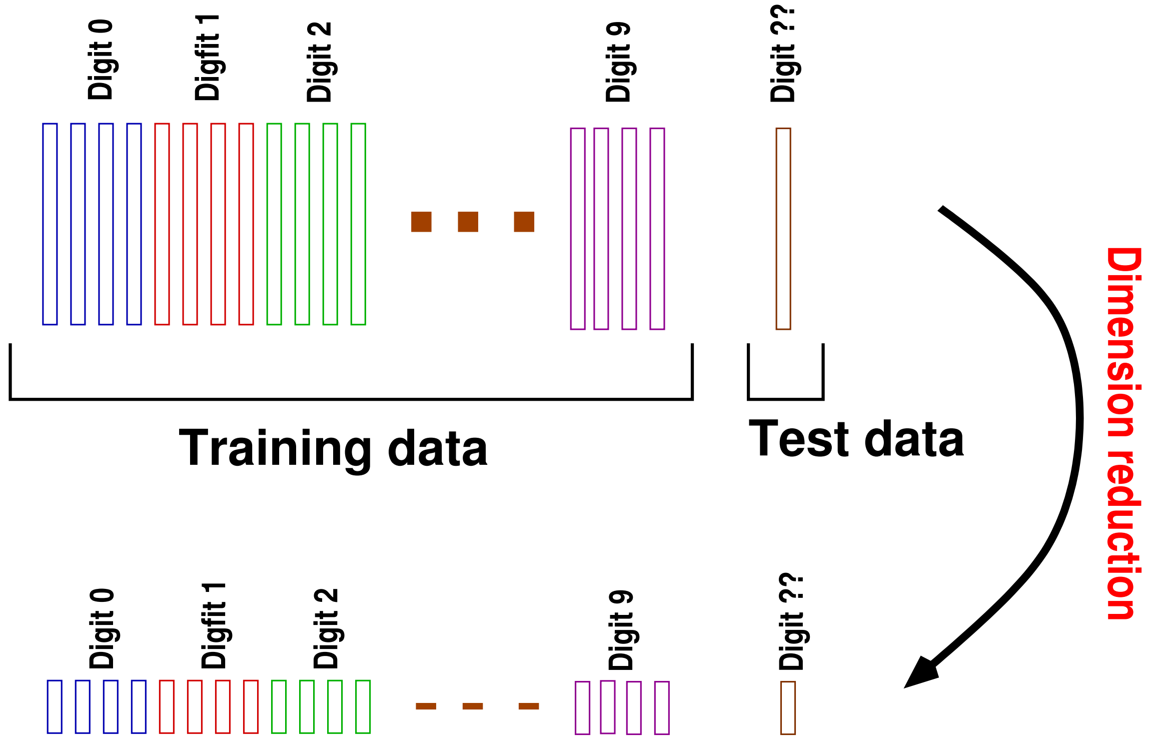




Supervised learning: classification

➤ Best illustration: written digits recognition example

Given: a set of labeled samples (training set), and an (unlabeled) test image.
Problem: find label of test image



➤ Roughly speaking: we seek dimension reduction so that recognition is 'more effective' in low-dim. space

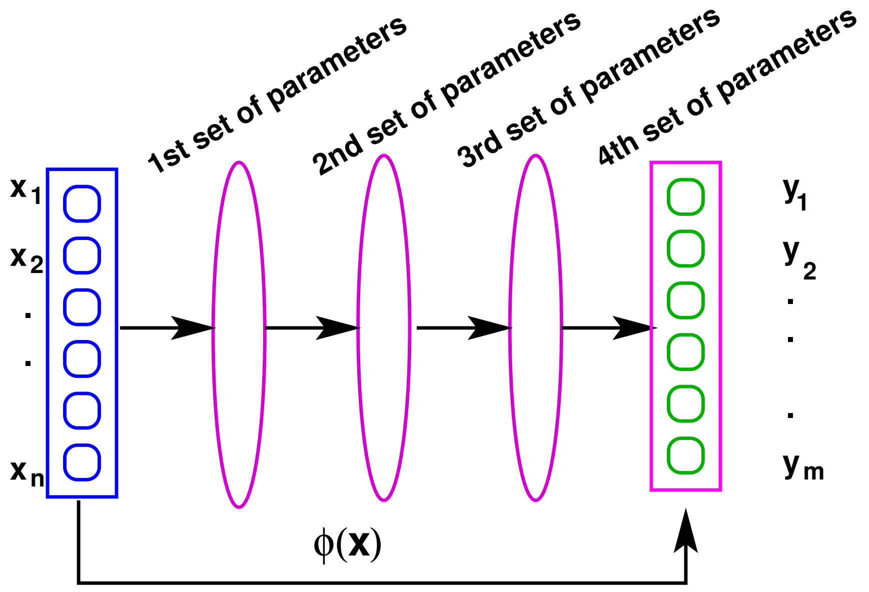
A few words on Deep Neural Networks (DNNs)

➤ Ideas of neural networks goes back to the 1960s - were popularized in early 1990s – then laid dormant until recently.

➤ Two reasons for the come-back:

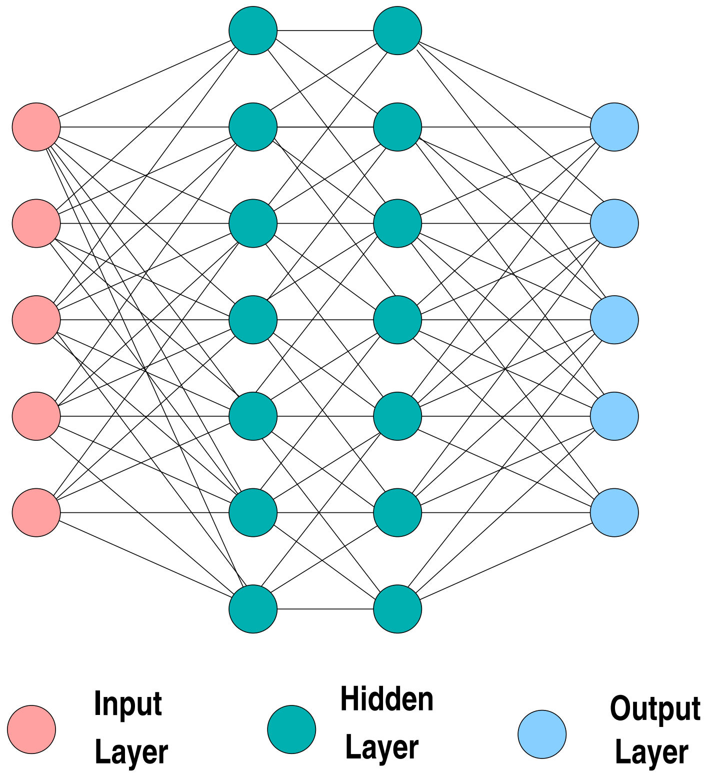
- DNN are remarkably effective in some applications
- big progress made in hardware [$\longrightarrow$ affordable ‘training cost’]

➤ Training a neural network can be viewed as a problem of approximating a function ϕ which is defined via sets of parameters:



Problem: find sets of parameters such that $\phi(x) \approx y$

```
Input: $x$, Output: $y$
Set: $z_0 = x$
For $l = 1$ : L+1 Do:
        $z_l = \sigma(W_l^T z_{l-1} + b_l)$
End
Set: $y = \phi(x) := z_{L+1}$
```



- layer # 0 = input layer
- layer # $(L+1)$ = output layer

➤ A matrix W_l is associated with layers 1,2, $L+1$.

➤ Problem: Find ϕ (i.e., matrices W_l) s.t. $\phi(x) \approx y$

DNN (continued)

- Problem is not convex, highly parameterized, ...,
- .. Main method used: Stochastic gradient descent [basic]
- It all looks like alchemy... but it works well for certain applications
- Training is still quite expensive – GPUs can help
- *Very* active area of research

Conclusion

- *Many* interesting new matrix problems in areas that involve the effective mining of data
- Among the most pressing issues is that of reducing computational cost - [SVD, SDP, ..., too costly]
- Many online resources available
- Huge potential in areas like materials science though inertia has to be overcome
- To a researcher in computational linear algebra : Tsunami of change on types or problems, algorithms, frameworks, culture,..
- But change should be welcome :

When one door closes, another opens; but we often look so long and so regretfully upon the closed door that we do not see the one which has opened for us.

Alexander Graham Bell (1847-1922)

➤ In the words of "Who Moved My Cheese?" [Spencer Johnson, 2002]

"If you do not change, you can become extinct!"

➤ In the words of Einstein:

"Life is like riding a bicycle. To keep your balance you need to keep moving"

Thank you !

➤ Visit my web-site at `www.cs.umn.edu/~saad`