Large-Scale Methods for Nonlinear Manifold Learning

Max Vladymyrov

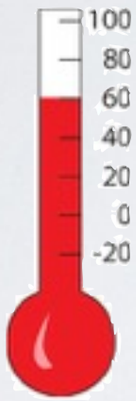
EECS, School of Engineering

University of California, Merced

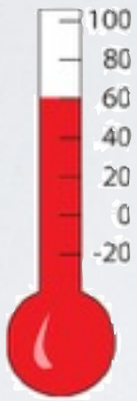
PhD defense,

Merced, CA

November 28, 2014

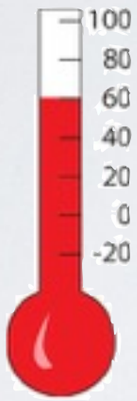

100
80
60
40
20
0
-20


STORMY
RAIN
CHANGE
FAIR
VERY DRY
29
30
28
31
27
26


100
80
60
40
20
0
-20

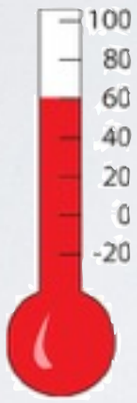

STORMY
RAIN
CHANGE
FAIR
VERY DRY
29
30
28
31
27
26


MAL
NE
MV
QTVV
HYOS
PLUG
ESLR
11.61
21.14
26.37
62.20
21.77
26.6
19.59
49.06
0.13
-0.35
0.09
-0.16


100
80
60
40
20
0
-20

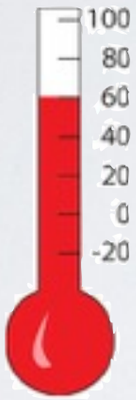

STORMY RAIN CHANGE FAIR VERY DRY
29
30
28
31
27
26


MAL
NE
MV
QTVV 11.61
HYOS 21.14
PLUG 26.37
ESLR 62.20
21.77
26.6
19.59
49.06


100
80
60
40
20
0
-20

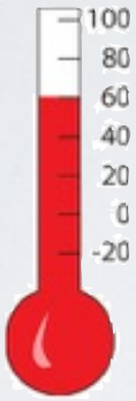

STORMY RAIN CHANGE FAIR VERY DRY
29
30
28
31
27
26


MAL
NE
MV
QTVV
HYOS
PLUG
ESLR
11.61
21.14
26.37
62.20
21.77
26.6
19.59
49.06
-0.11
-0.04
-0.01
0.53
0.13
-0.35
0.09
-0.16


100
80
60
40
20
0
-20


STORMY RAIN CHANGE FAIR VERY DRY
29
30
28
31
27
26

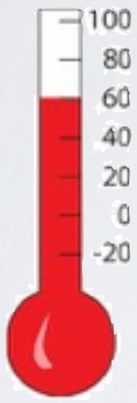

MAL
NE
MV
QTVV
HYOS
PLUG
ESLR
11.61
21.14
26.37
62.20
21.77
26.6
19.59
49.06
-0.11
-0.04
-0.01
0.53
0.13
-0.35
0.09
-0.16


100
80
60
40
20
0
-20


DUE:
$

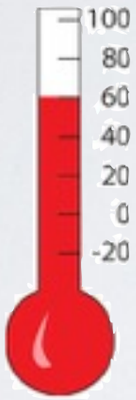

MAL
NE
MV
QTVV 11.61
HYOS 21.14
PLUG 26.37
ESLR 62.20
21.77
26.6
19.59
49.06

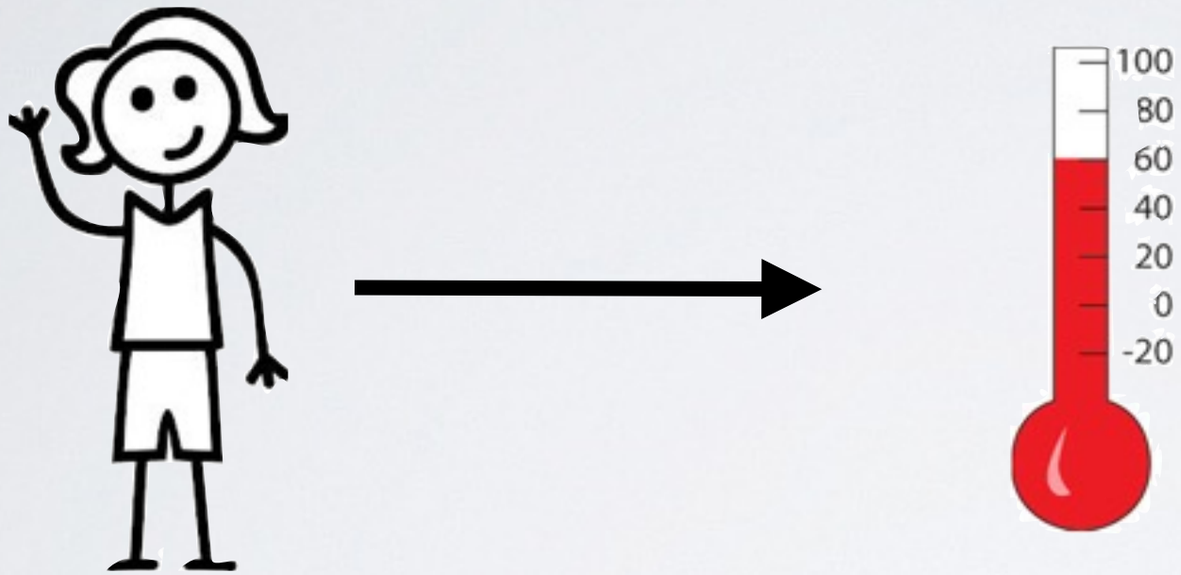

STORMY RAIN CHANGE FAIR VERY DRY
29
30
28
31
27
26

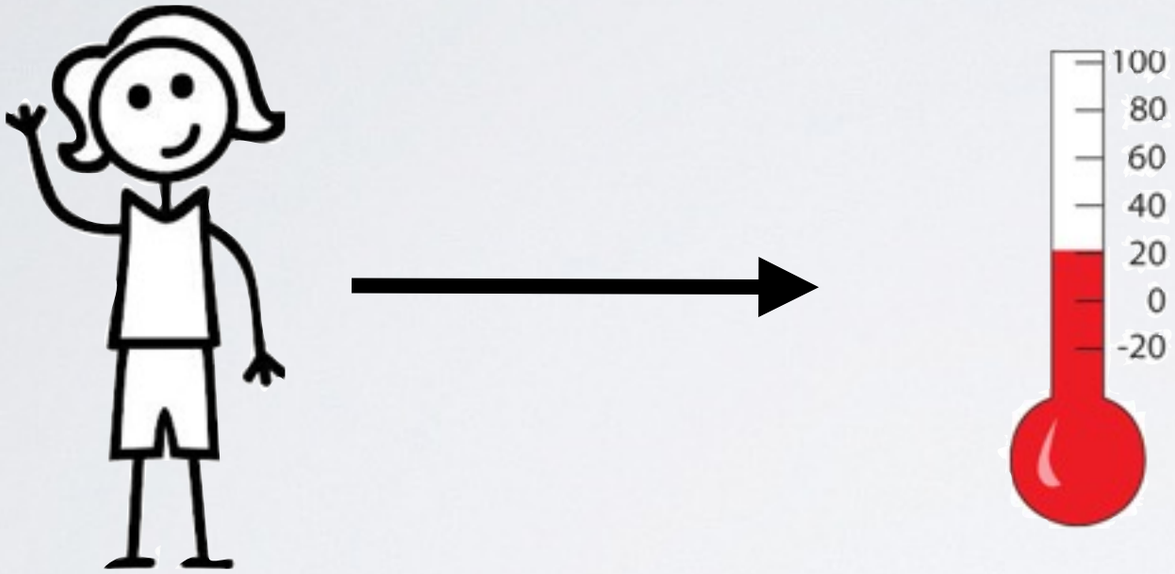

100
80
60
40
20
0
-20

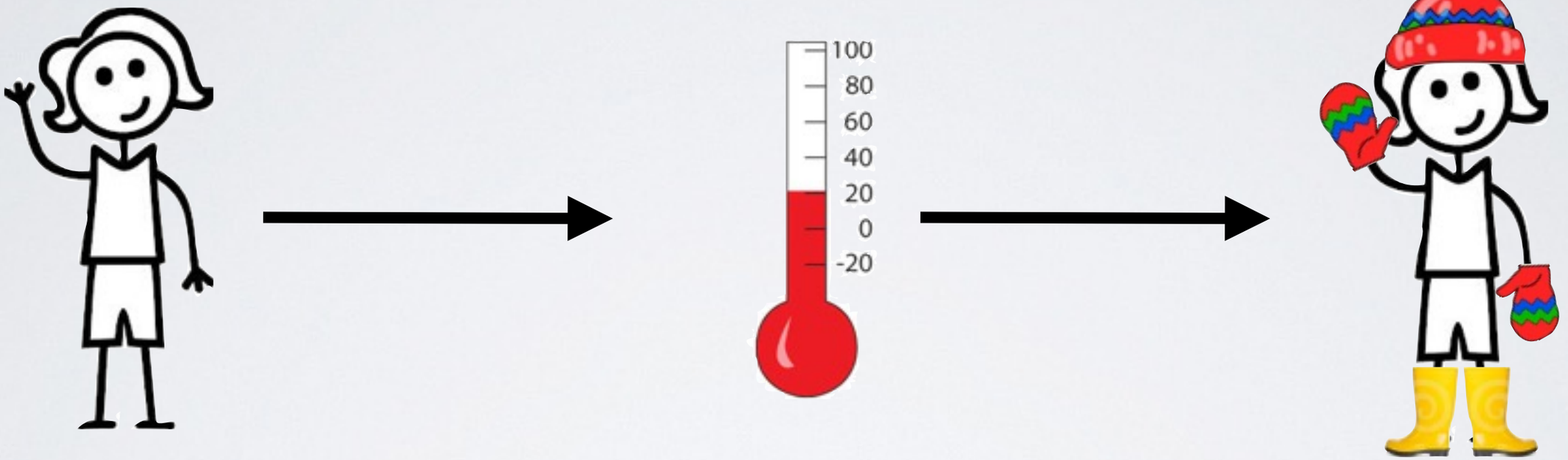

DUE:
$

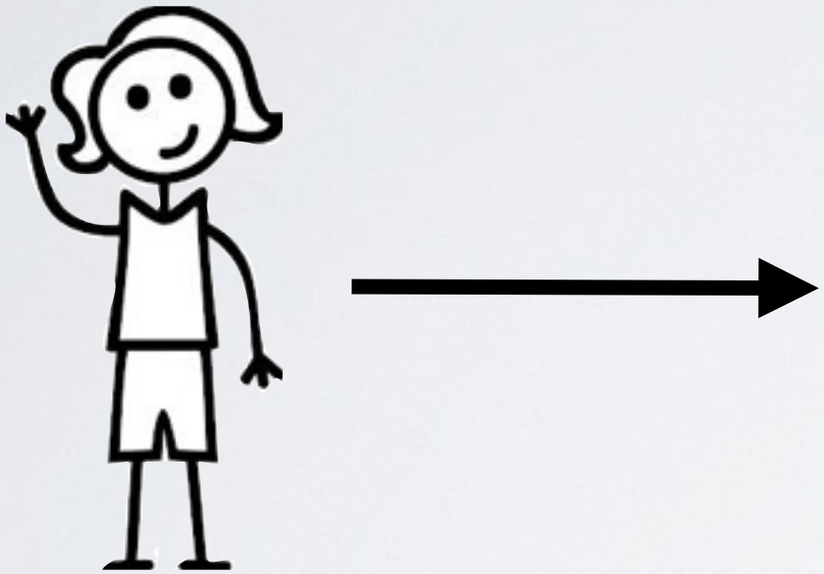
Data is **everywhere!**

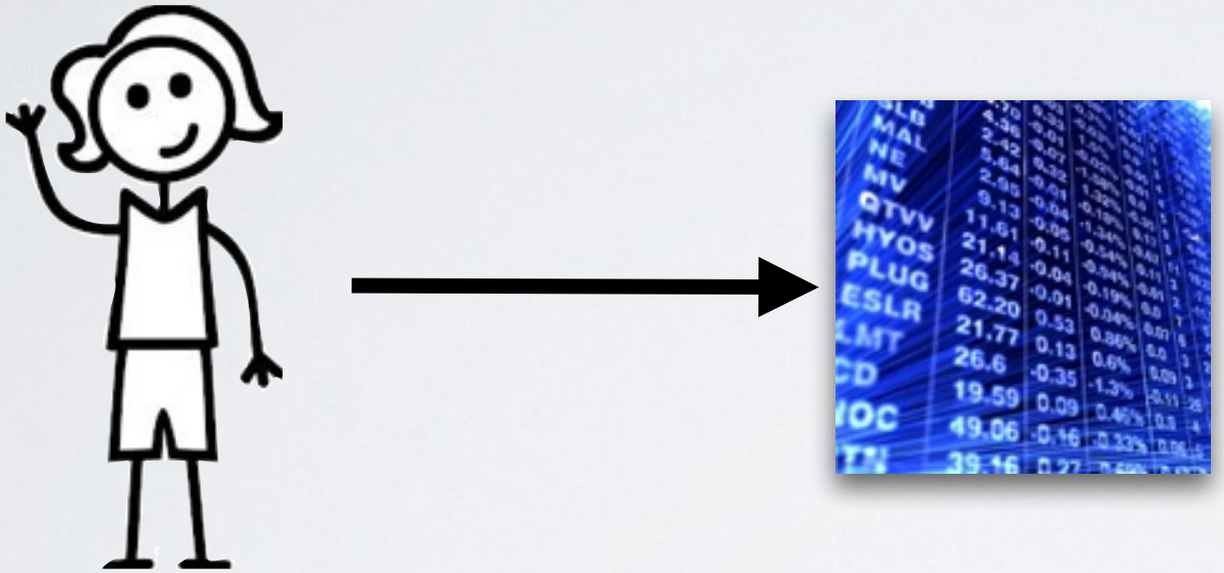


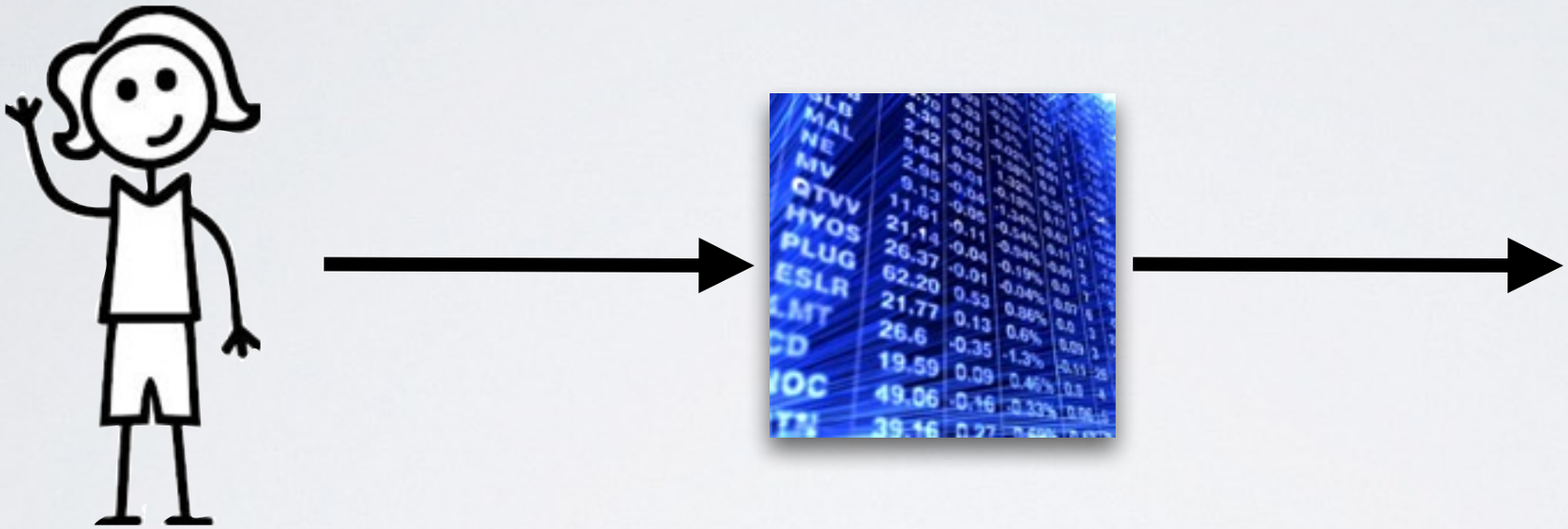

100
80
60
40
20
0
-20


100
80
60
40
20
0
-20


100
80
60
40
20
0
-20




MAL
NE
MV
QTVV
HYOS
PLUG
ESLR
LMT
CD
OC


MAL
NE
MV
HYOS
PLUG
ESLR
LMT
CD
IOC
49.06
39.16


MAL
NE
MV
QTVV
HYOS
PLUG
ESLR
LMT
CD
VOC

- historical data

- historical data
- market condition

- historical data
- market condition
- other players' actions

- historical data
- market condition
- other players' actions
- governmental regulations

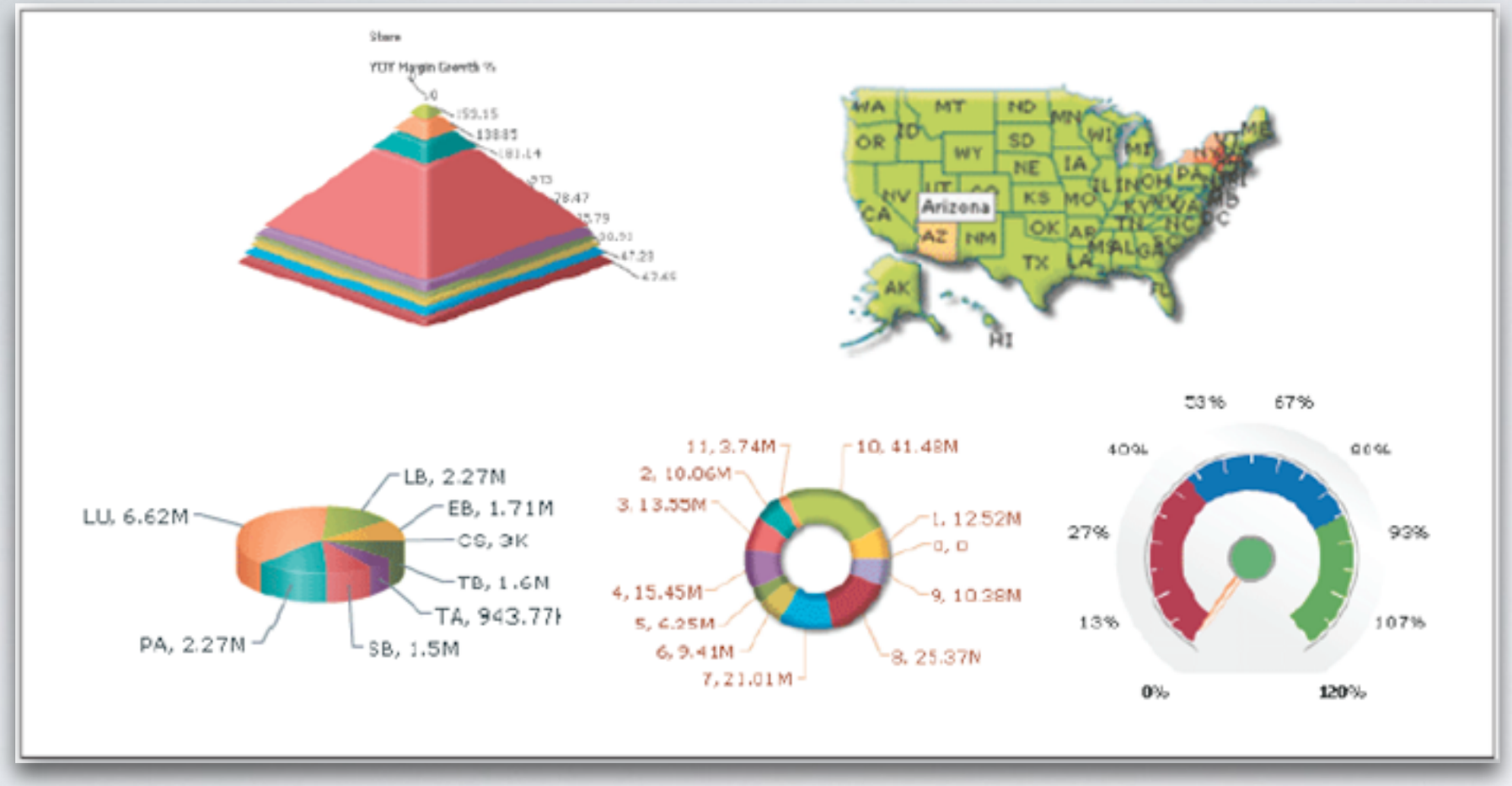
- historical data
- market condition
- other players' actions
- governmental regulations
- other factors

Data is
multidimensional!

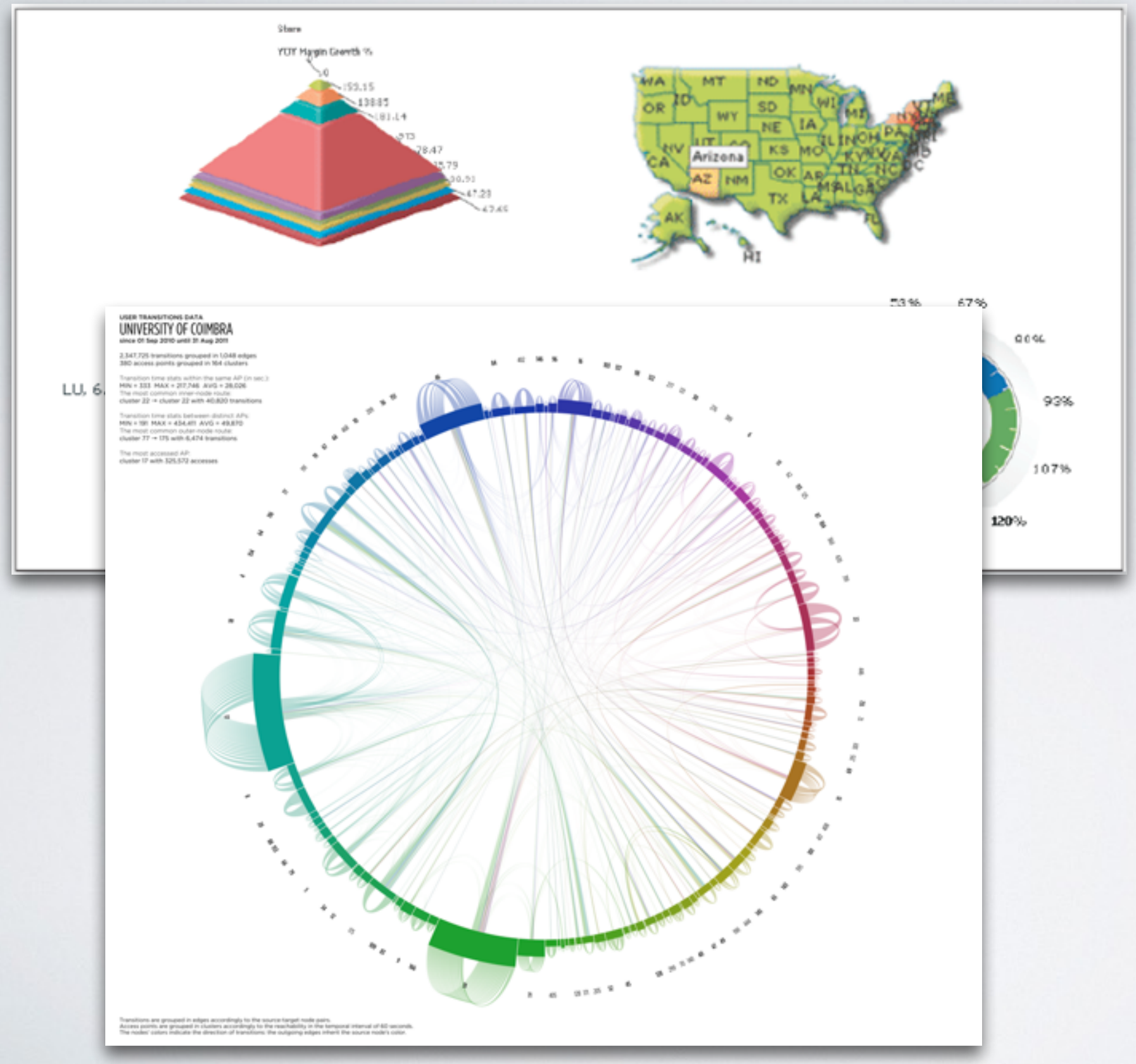
?

- historical data
- market condition
- other players' actions
- governmental regulations
- other factors

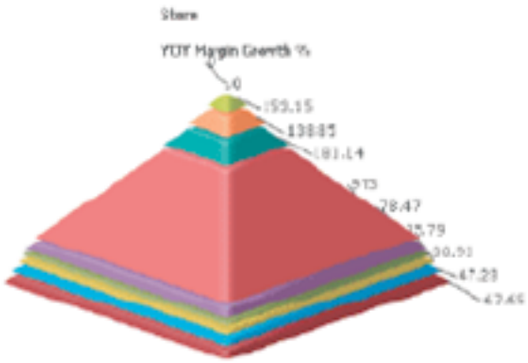
Possible ways to analyze high-dimensional data

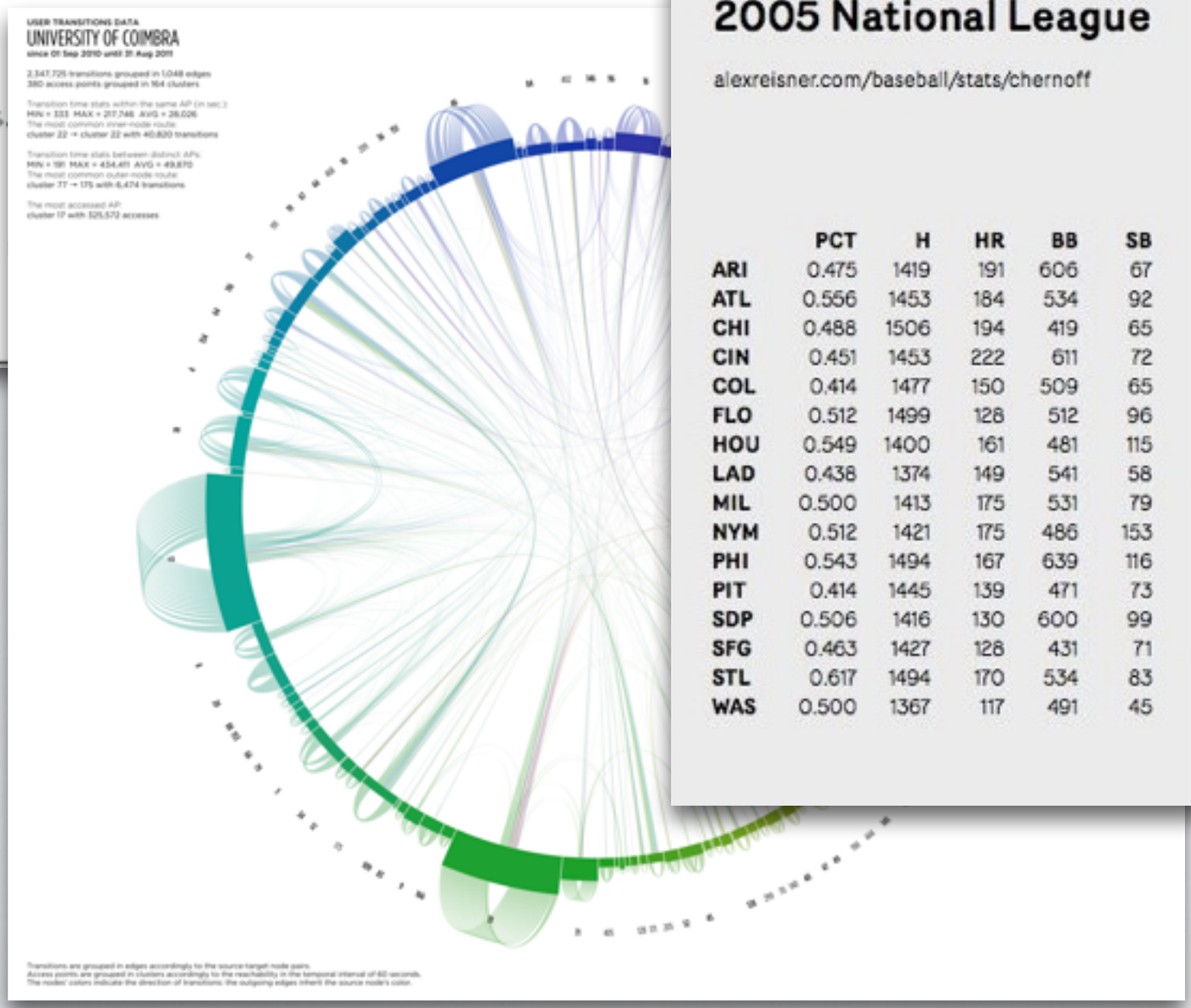


Possible ways to analyze high-dimensional data



Possible ways to analyze high-dimensional data



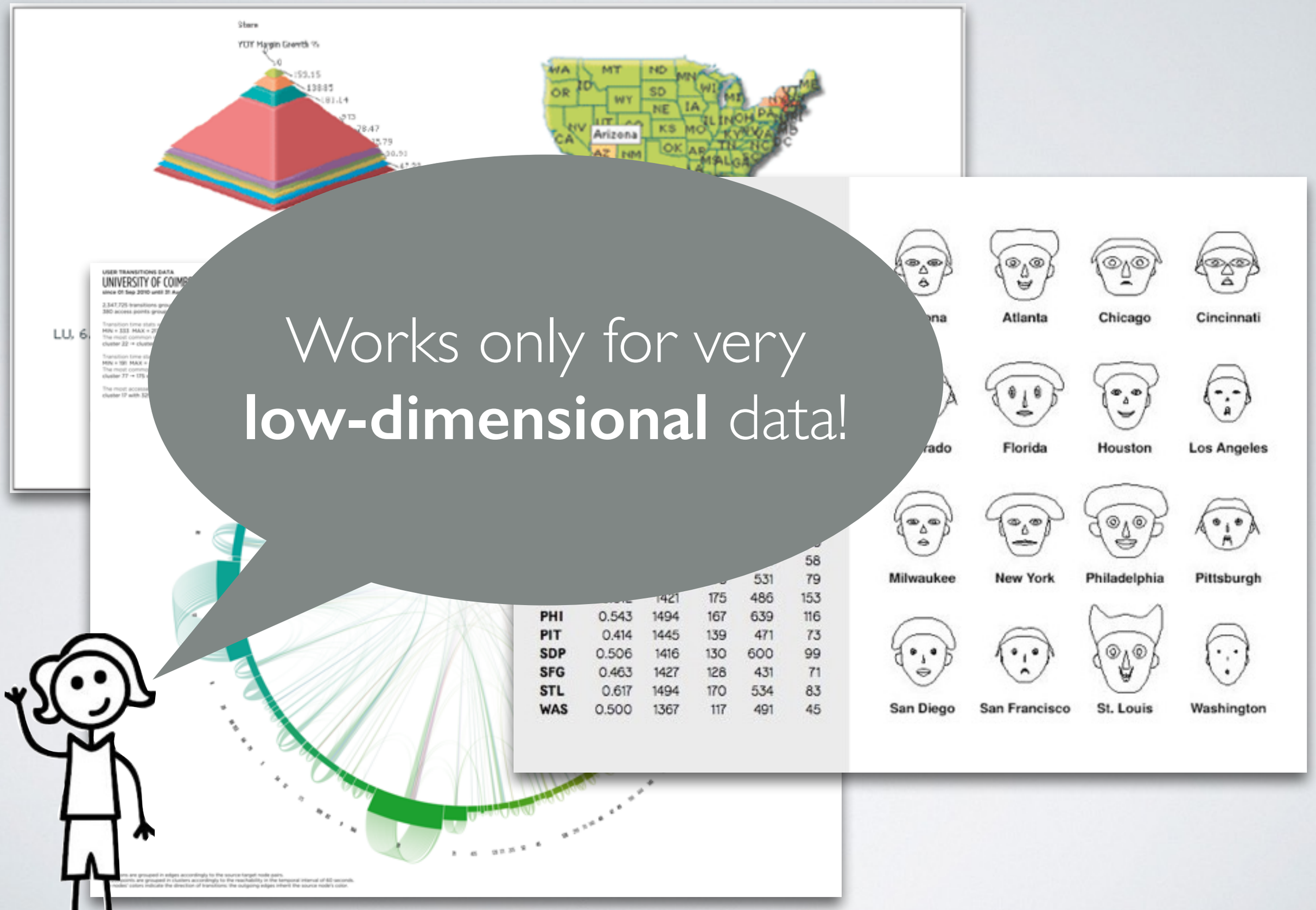


Chernoff Faces
2005 National League

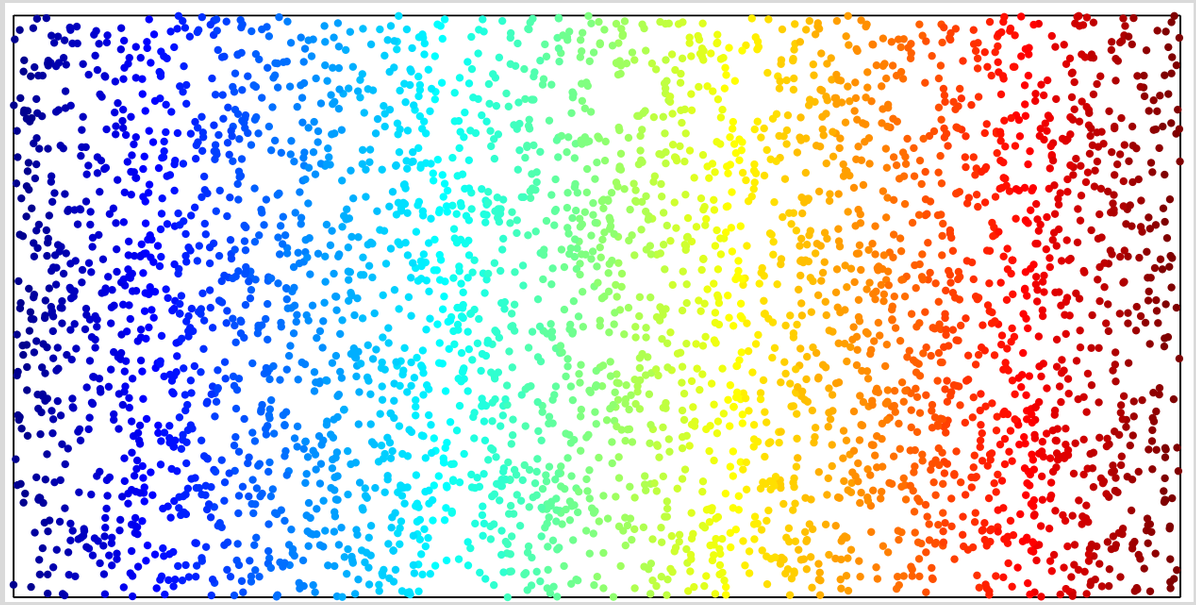
alexreisner.com/baseball/stats/chernoff

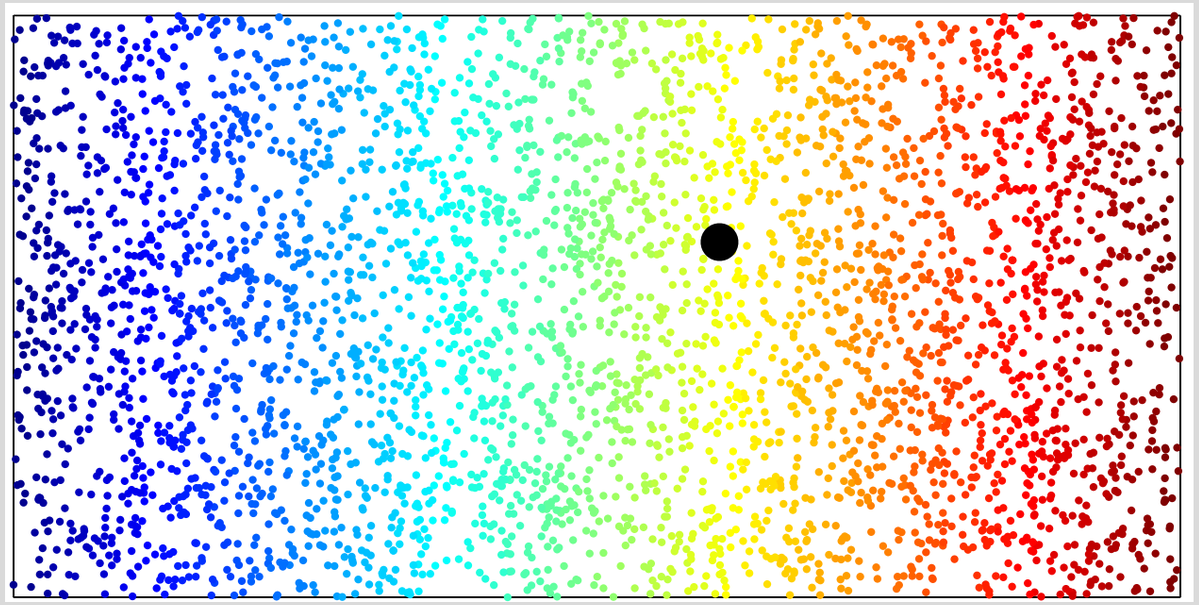
	PCT	H	HR	BB	SB
ARI	0.475	1419	191	606	67
ATL	0.556	1453	184	534	92
CHI	0.488	1506	194	419	65
CIN	0.451	1453	222	611	72
COL	0.414	1477	150	509	65
FLO	0.512	1499	128	512	96
HOU	0.549	1400	161	481	115
LAD	0.438	1374	149	541	58
MIL	0.500	1413	175	531	79
NYM	0.512	1421	175	486	153
PHI	0.543	1494	167	639	116
PIT	0.414	1445	139	471	73
SDP	0.506	1416	130	600	99
SFG	0.463	1427	128	431	71
STL	0.617	1494	170	534	83
WAS	0.500	1367	117	491	45

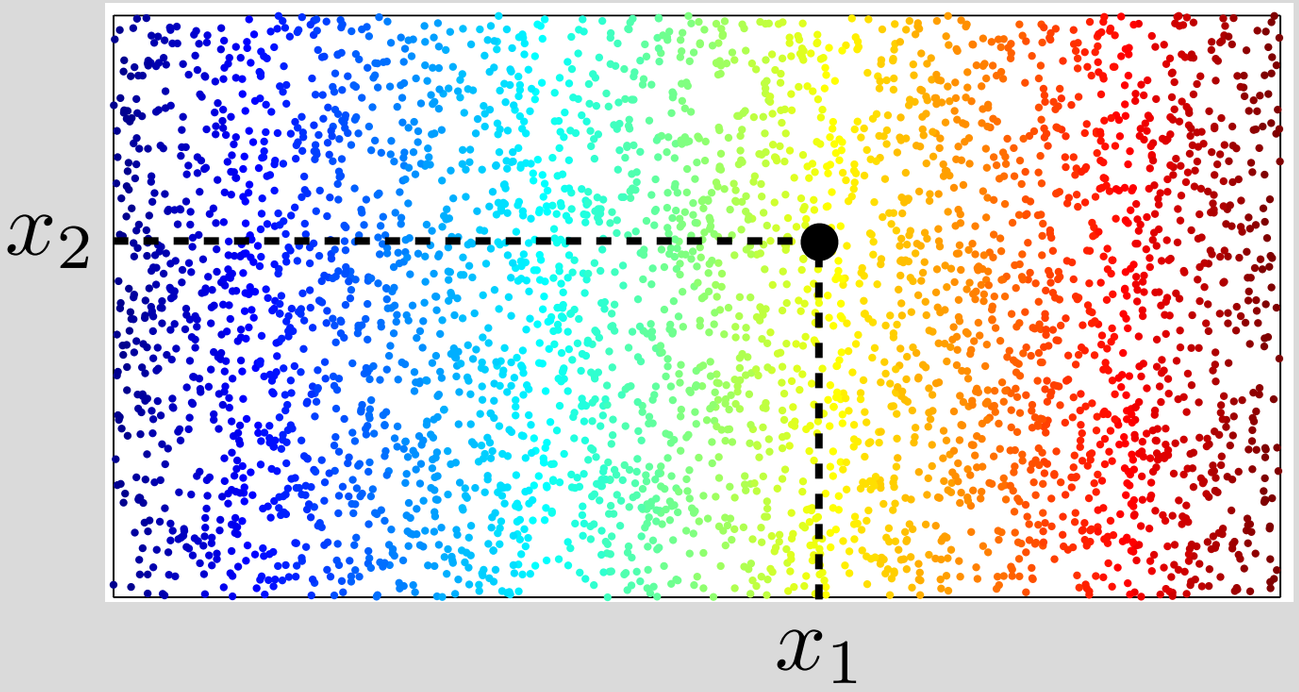
Possible ways to analyze high-dimensional data



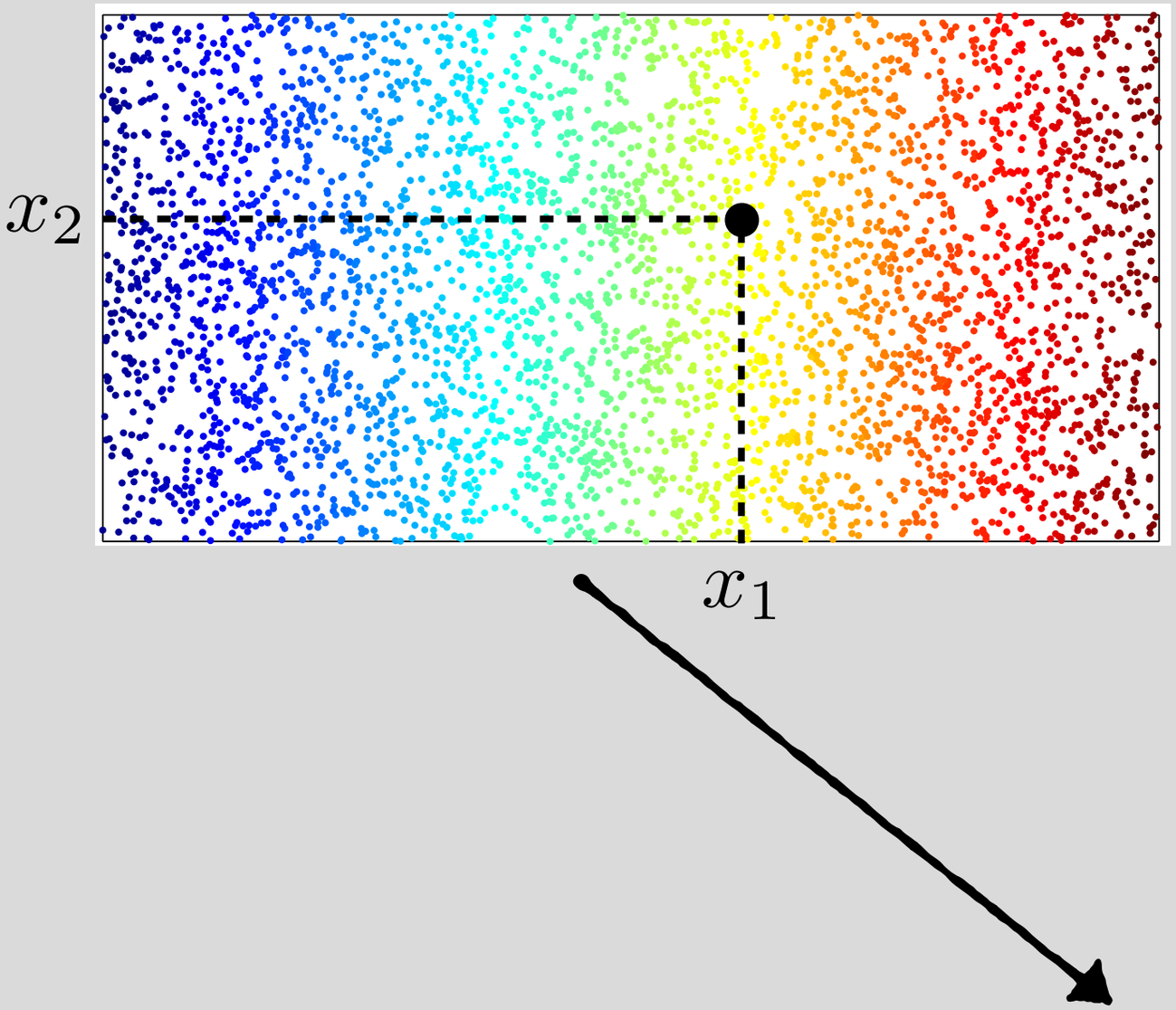
Works only for very **low-dimensional** data!



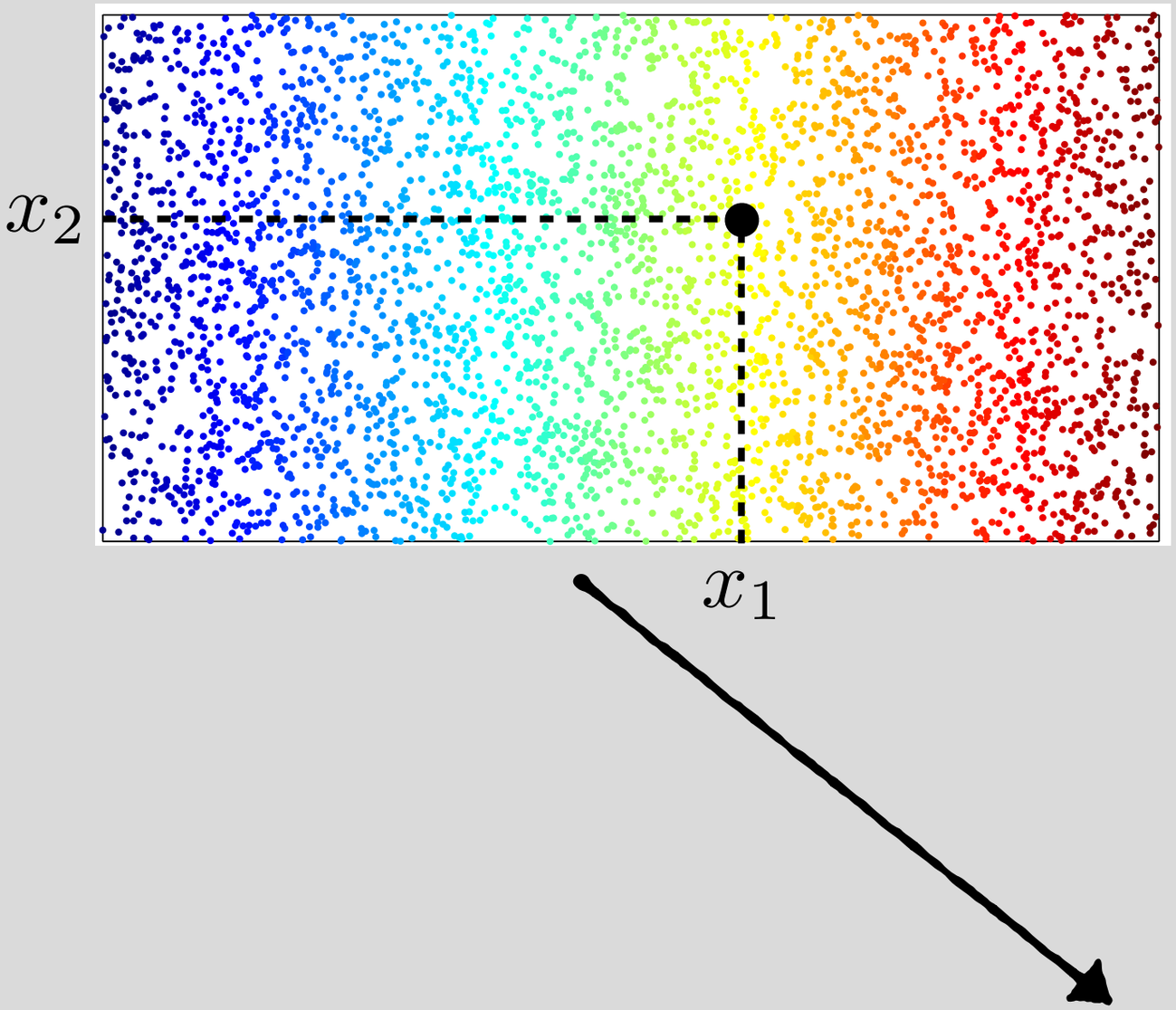




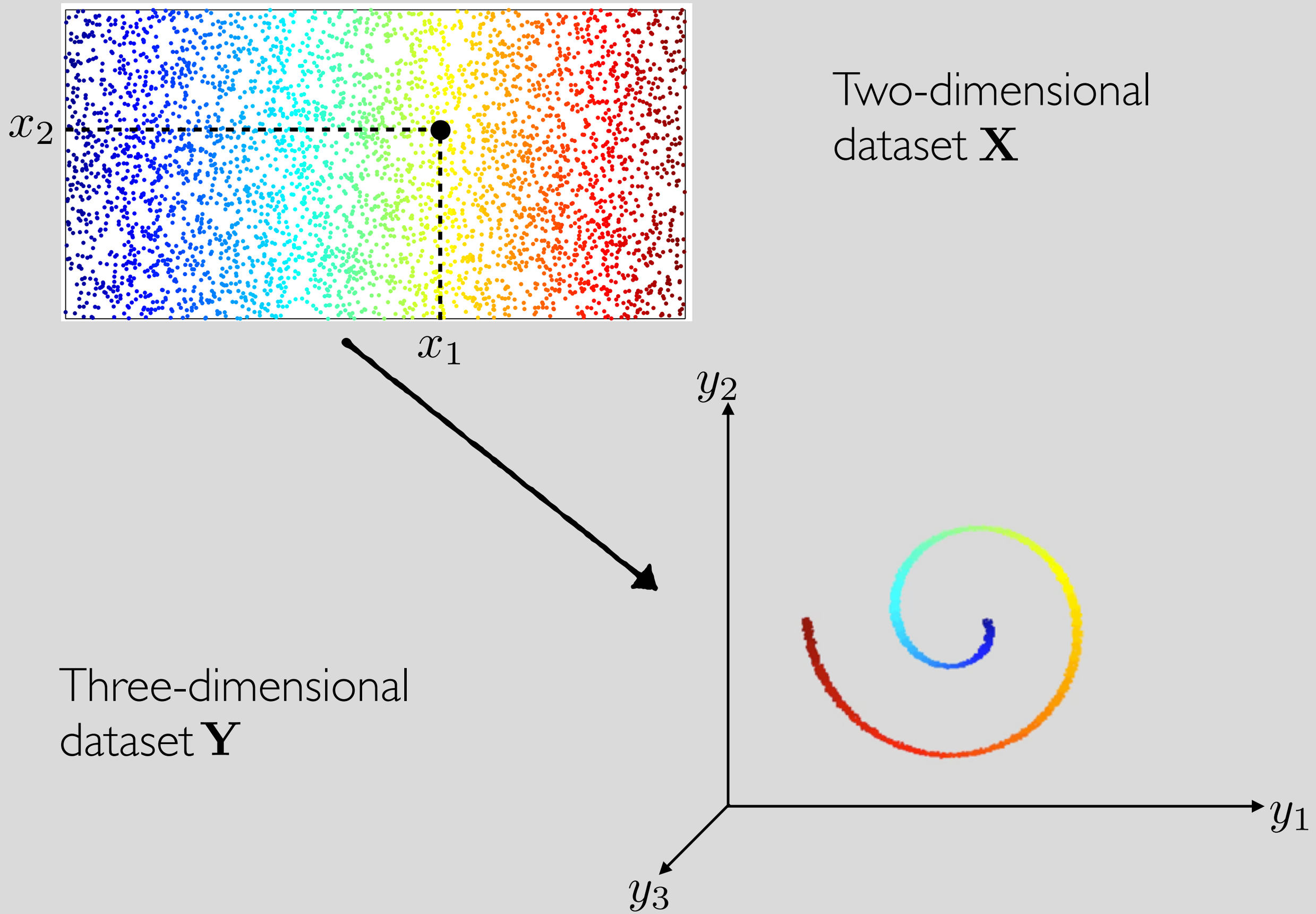
Two-dimensional dataset $\mathbf{X}$



Two-dimensional dataset $\mathbf{X}$

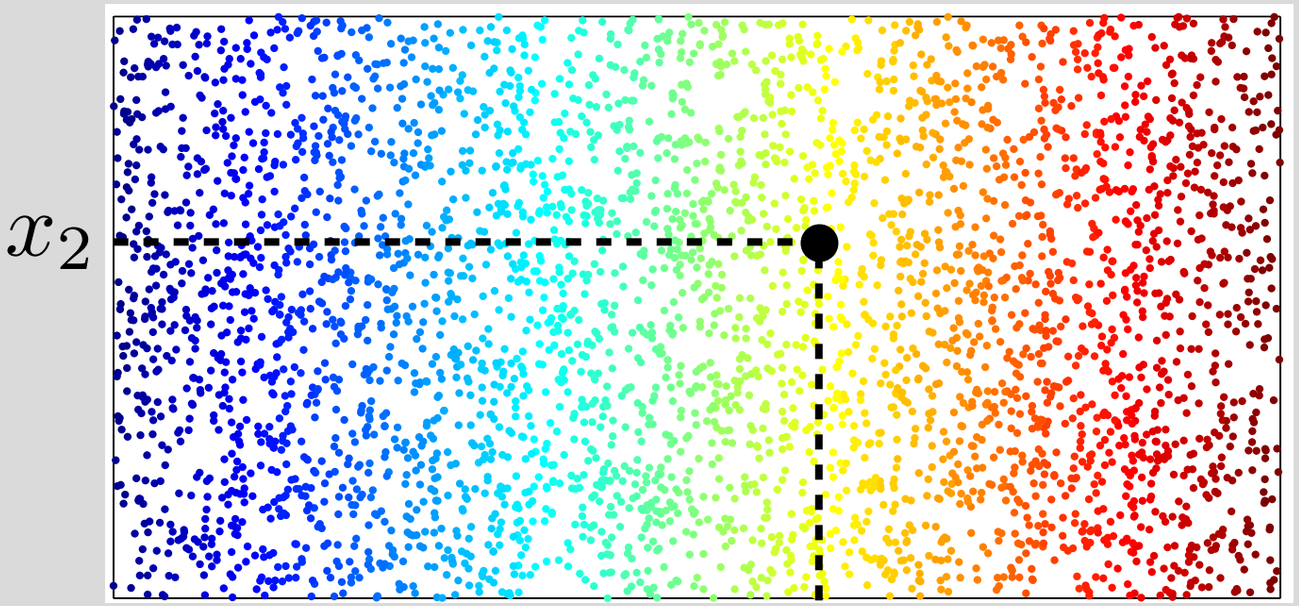


Two-dimensional dataset $\mathbf{X}$



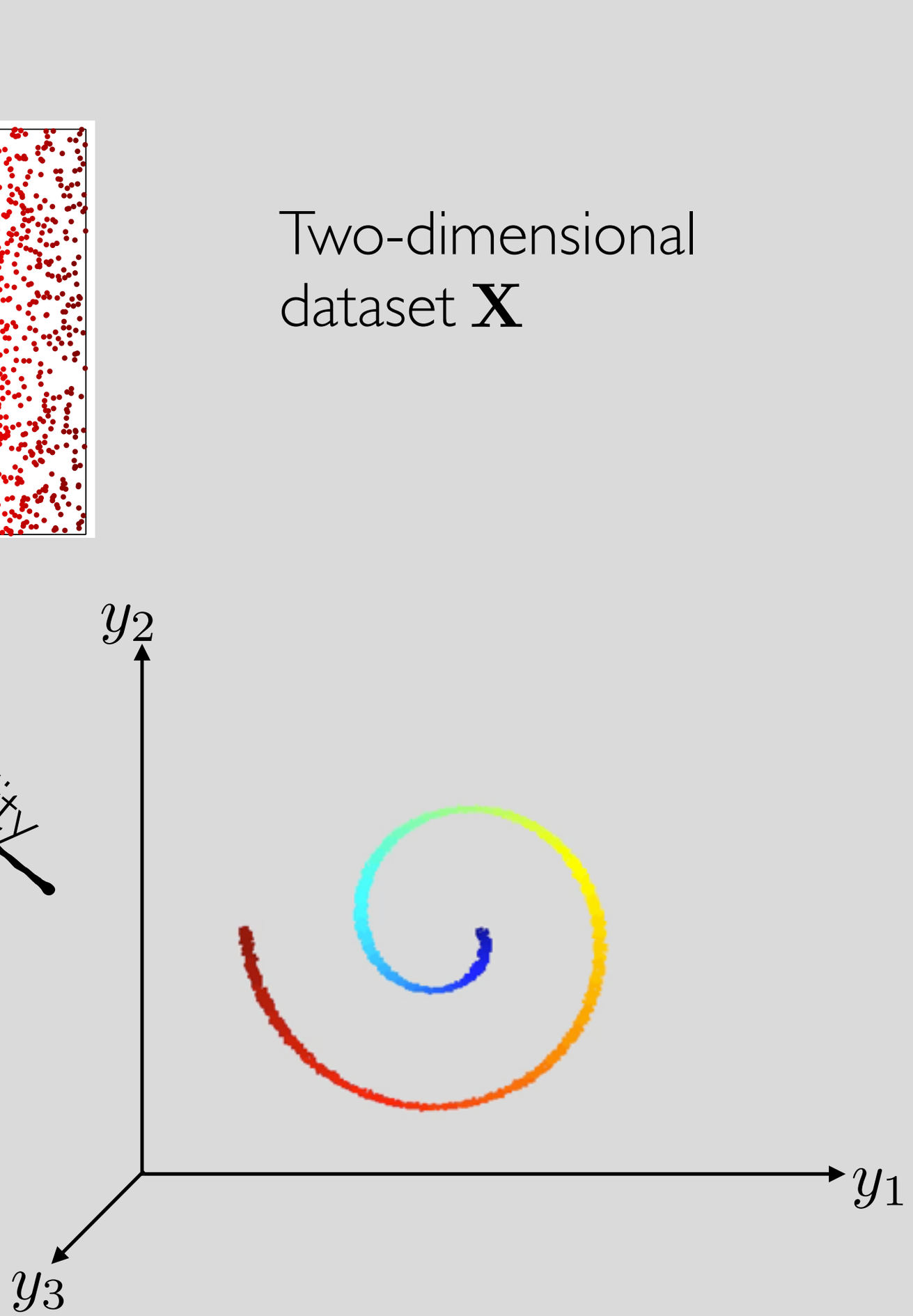
Two-dimensional dataset $\mathbf{X}$

Three-dimensional dataset $\mathbf{Y}$

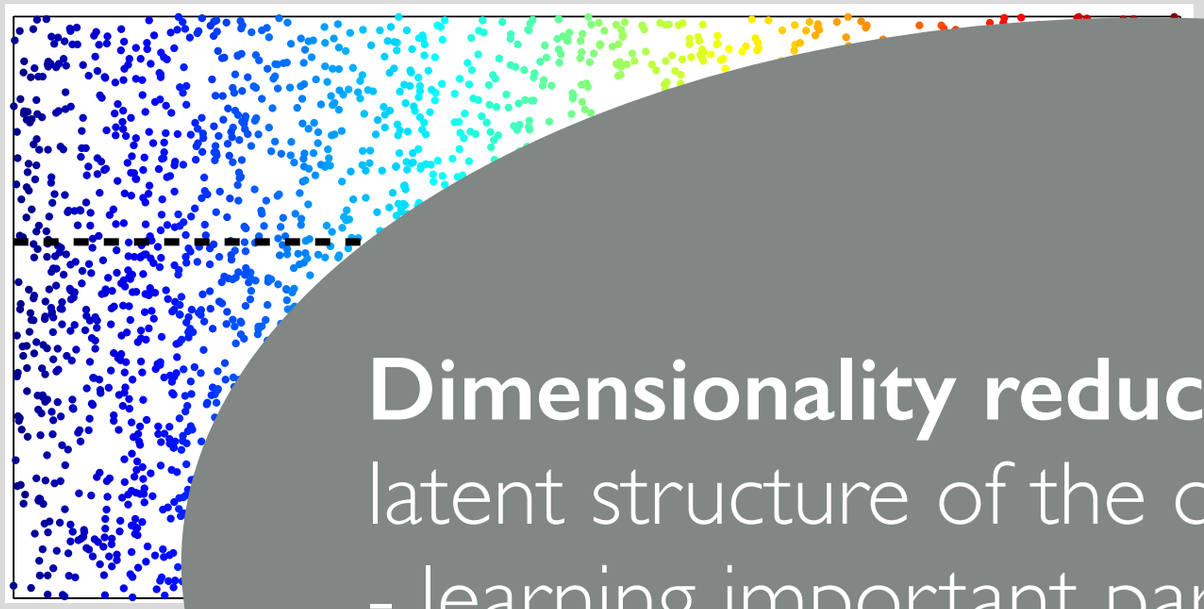


Two-dimensional dataset $\mathbf{X}$

Dimensionality reduction



Three-dimensional dataset $\mathbf{Y}$



nsional

Dimensionality reduction tries to find latent structure of the data by
- learning important parameters,
- removing unnecessary dimensions (noise).

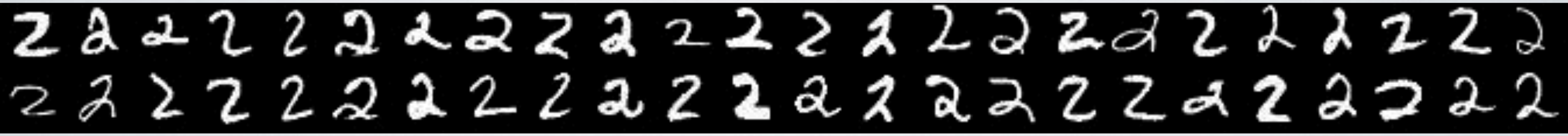
ee-dimensional
set $\mathbf{Y}$

Other use of dimensionality reduction

- Preprocessing before other task e.g. classification or regression:
  - denoising,
  - decreasing the complexity with respect to dimensionality D.
- Extracting latent structure of the data:
  - feature learning,
  - cluster information,
  - deep networks with autoencoders.
- etc.

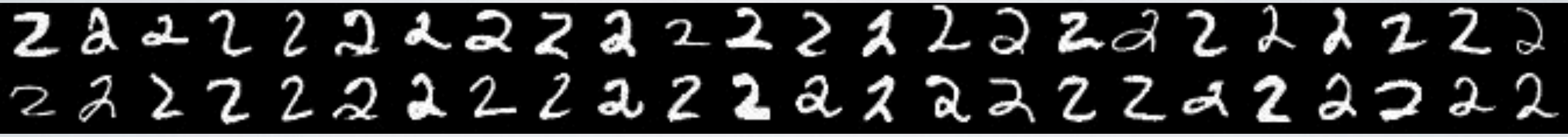
MNIST Handwritten digits

Consider a dataset with $1\,000$ handwritten digits 2 :



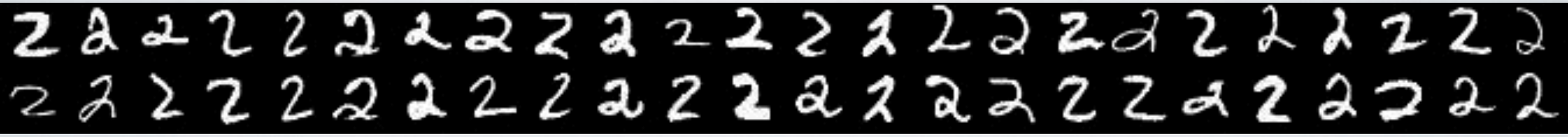
MNIST Handwritten digits

Consider a dataset with $1\,000$ handwritten digits 2 :



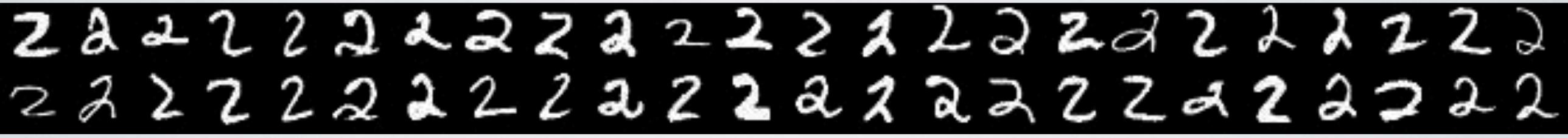
MNIST Handwritten digits

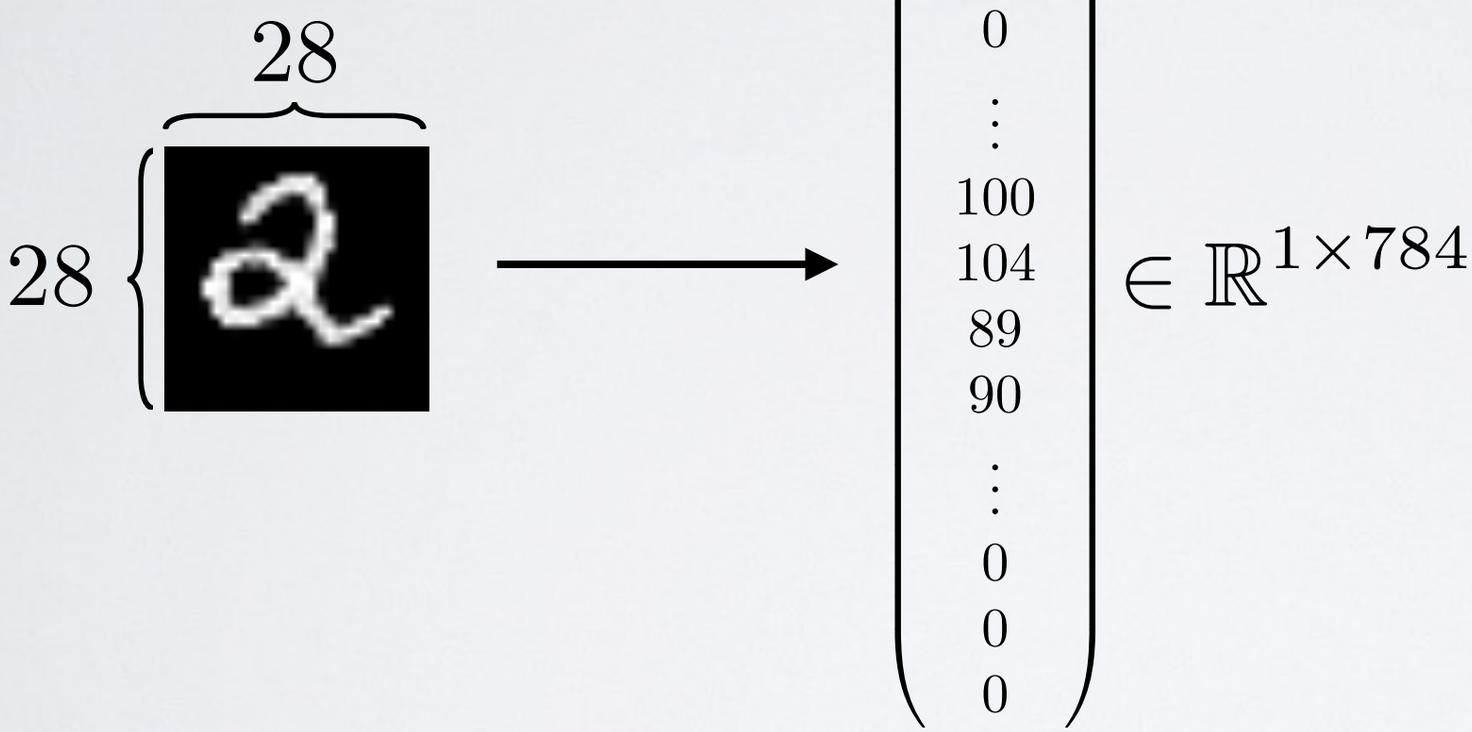
Consider a dataset with $1\,000$ handwritten digits 2 :



MNIST Handwritten digits

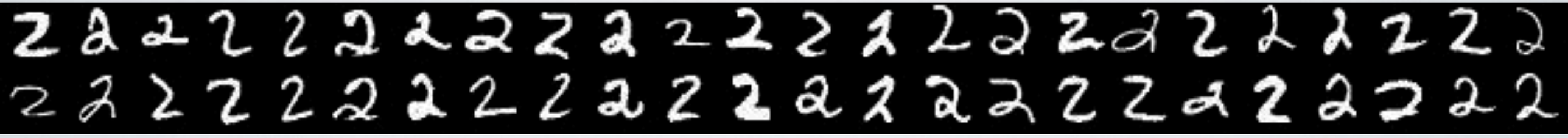
Consider a dataset with $1\,000$ handwritten digits 2 :





MNIST Handwritten digits

Consider a dataset with $1\,000$ handwritten digits 2 :



$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 100 \\ 104 \\ 89 \\ 90 \\ \vdots \\ 0 \\ 0 \\ 0 \end{pmatrix} \in \mathbb{R}^{1\times 784}$$

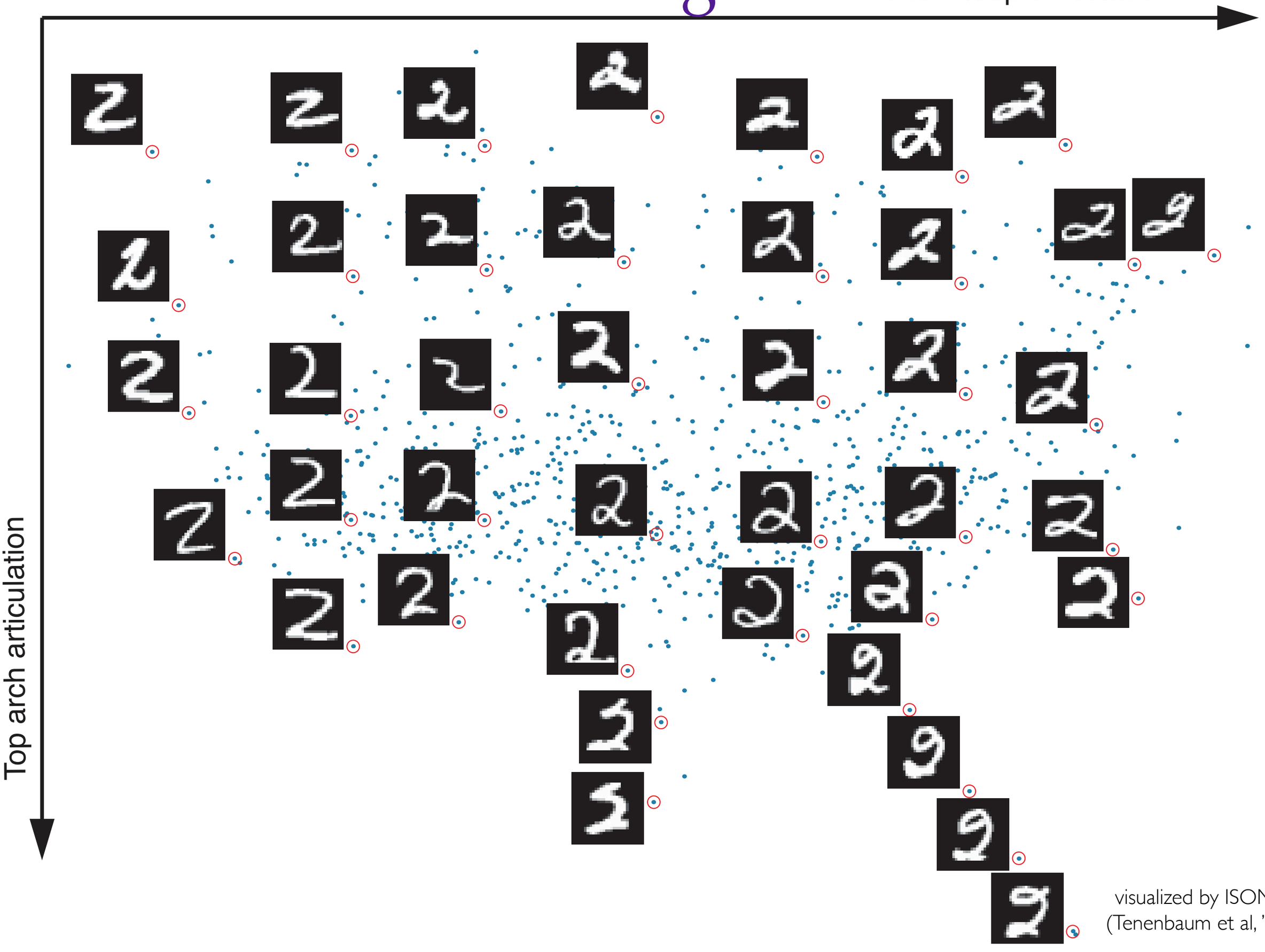
High-dimensional dataset: $\mathbf{Y} \in \mathbb{R}^{1\,000\times 784}$

Number of points: $N = 1\,000$

Number of dimensions: $D = 784$

Reduction space: $d = 2$

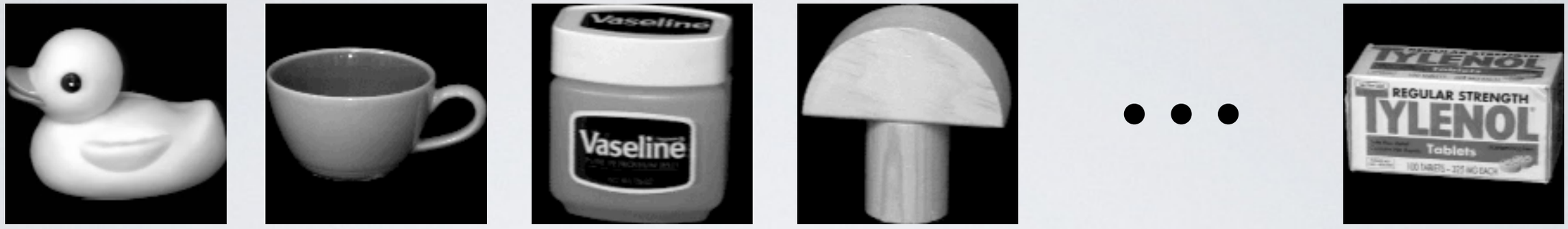
MNIST Handwritten digits



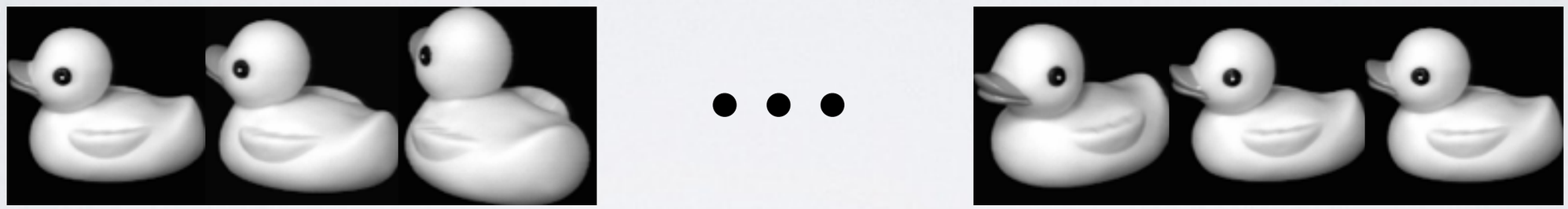
visualized by ISOMAP
(Tenenbaum et al, '00)

COIL-20 Rotational sequences

10 objects:



72 images per object:



128

128

$\downarrow$

$\mathbb{R}^{1\times 16\,384}$

High-dimensional dataset: $\mathbf{Y} \in \mathbb{R}^{720\times 16\,384}$

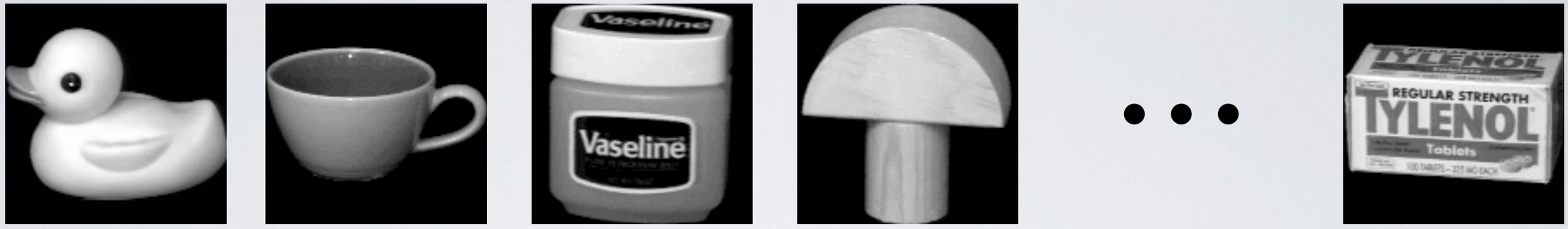
Number of points: $N = 720$

Number of dimensions: $D = 16\,384$

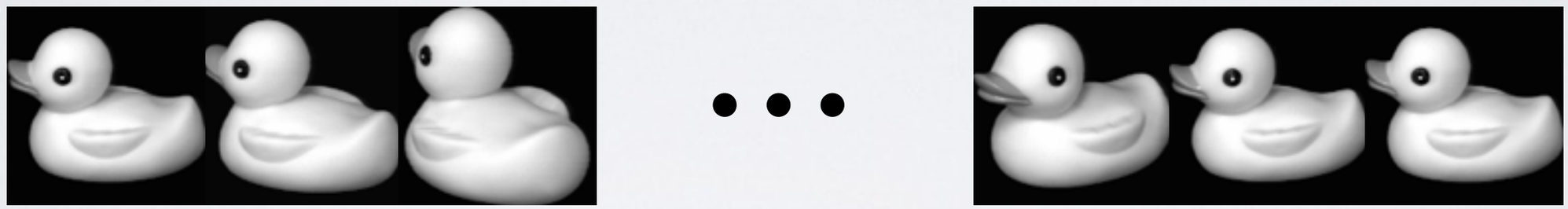
Reduction space: $d = 2$

COIL-20 Rotational sequences

10 objects:



72 images per object:



128

128

$\downarrow$

$\mathbb{R}^{1\times 16\,384}$

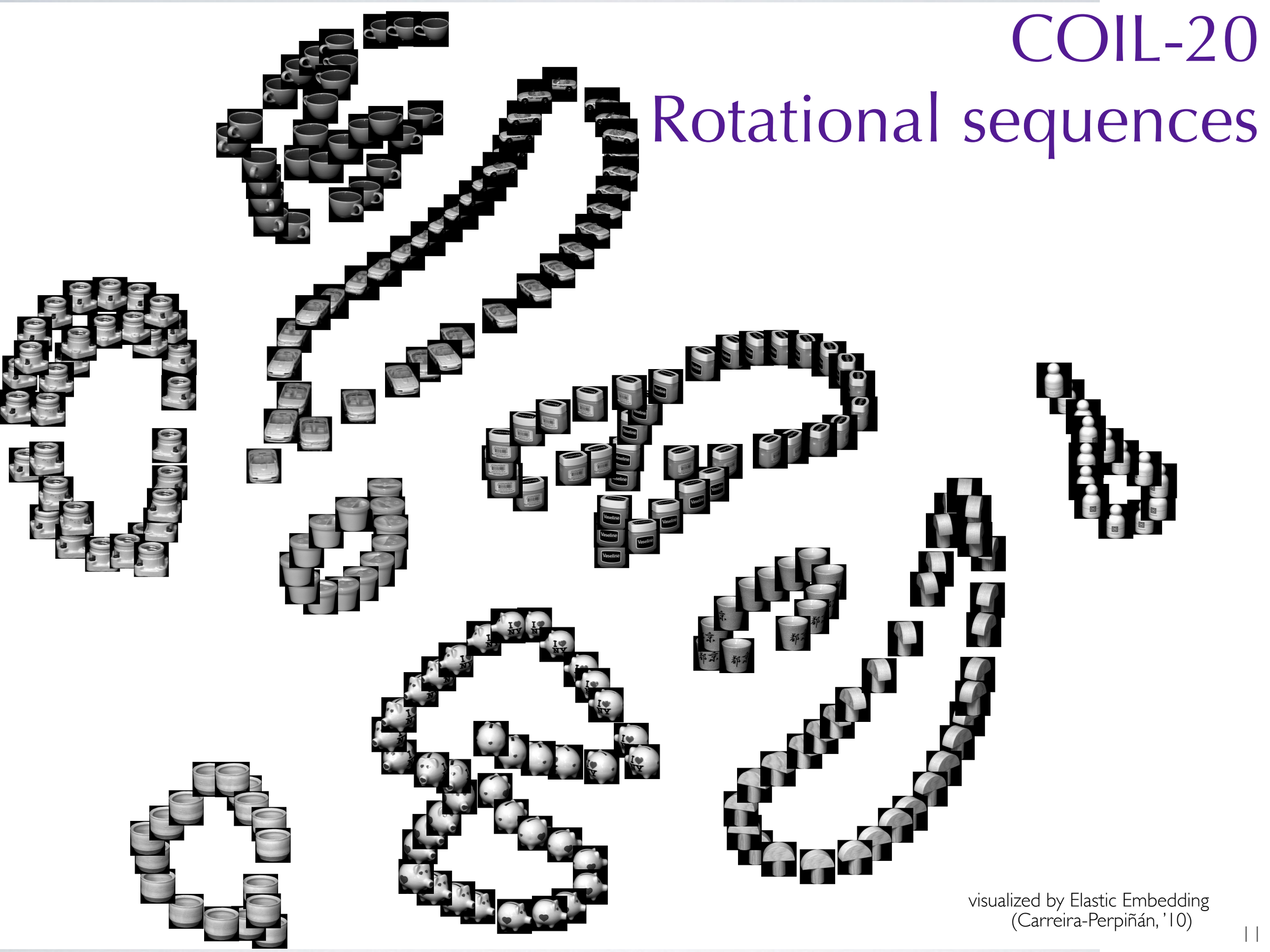
High-dimensional dataset: $\mathbf{Y} \in \mathbb{R}^{720\times 16\,384}$

Number of points: $N = 720$

Number of dimensions: $D = 16\,384$

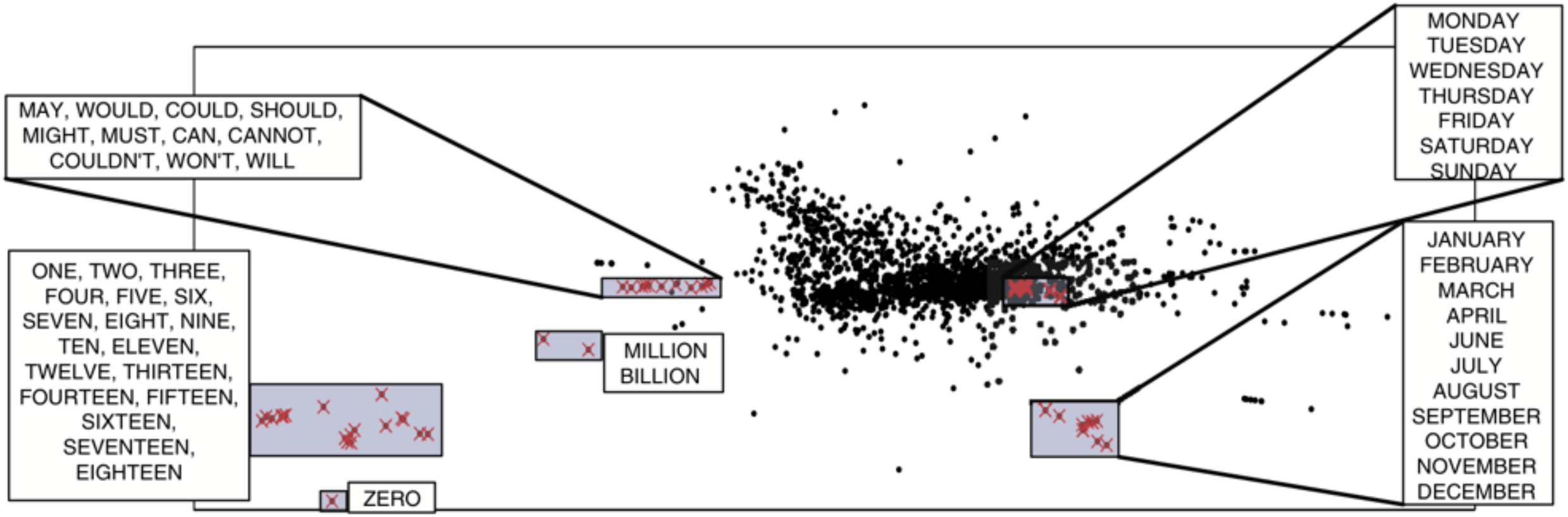
Reduction space: $d = 2$

COIL-20
Rotational sequences



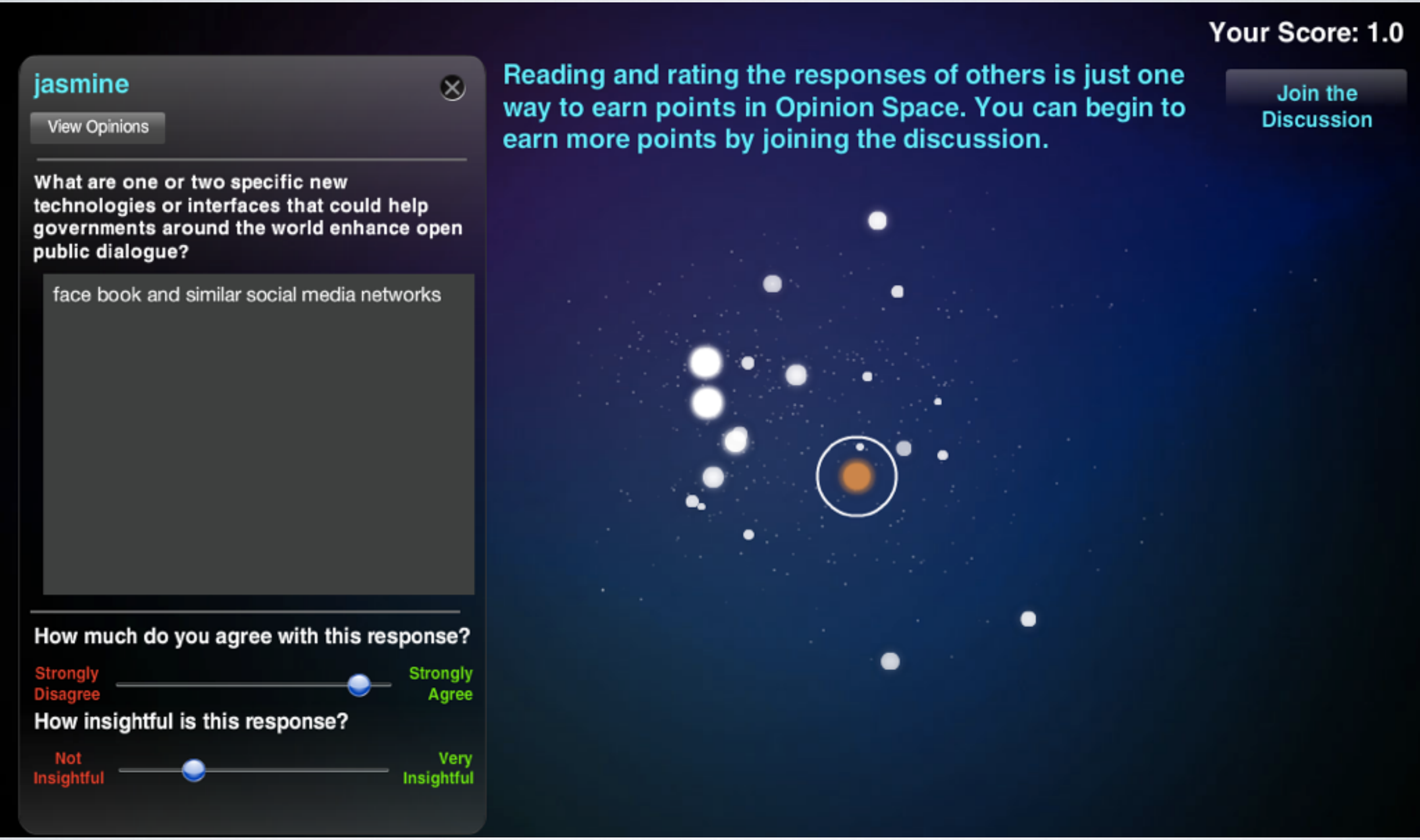
visualized by Elastic Embedding
(Carreira-Perpiñán, '10)

Text corpus



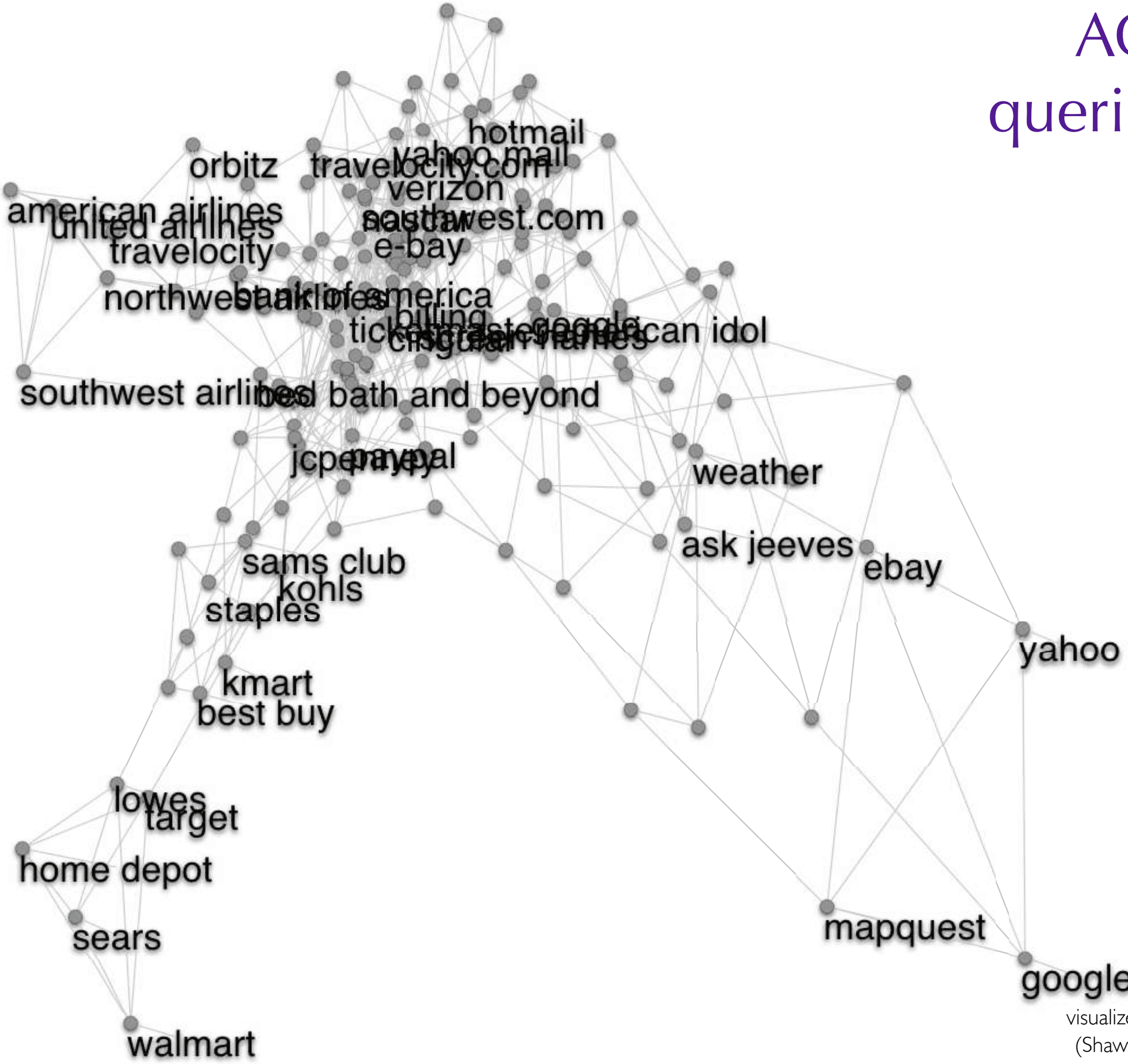
visualized using MVU
(Weinberger and Saul, '06)

Opinions of users on political issues



http://www.state.gov/opinionspace/

AOL search queries dataset



visualized using MVE w/ *b* matching (Shaw, '11)

Part I. Nonlinear dimensionality reduction

Nonlinear dimensionality reduction

Entropic affinities

Spectral methods

Stochastic Neighbor Embedding

s-SNE

t-SNE

Elastic Embedding

Nonlinear Embedding Methods (NLE)

Part II

Optimization using partial-Hessian

Part III

Locally Linear Landmarks

Large-scale approx. using *N*-Body methods

Barnes-Hut method

Fast Multipole Methods

Part I. Nonlinear dimensionality reduction

Nonlinear dimensionality reduction

Entropic affinities

Spectral methods

Stochastic Neighbor Embedding

s-SNE

t-SNE

Elastic Embedding

Nonlinear Embedding Methods (NLE)

Optimization using partial-Hessian

Part II

Part III

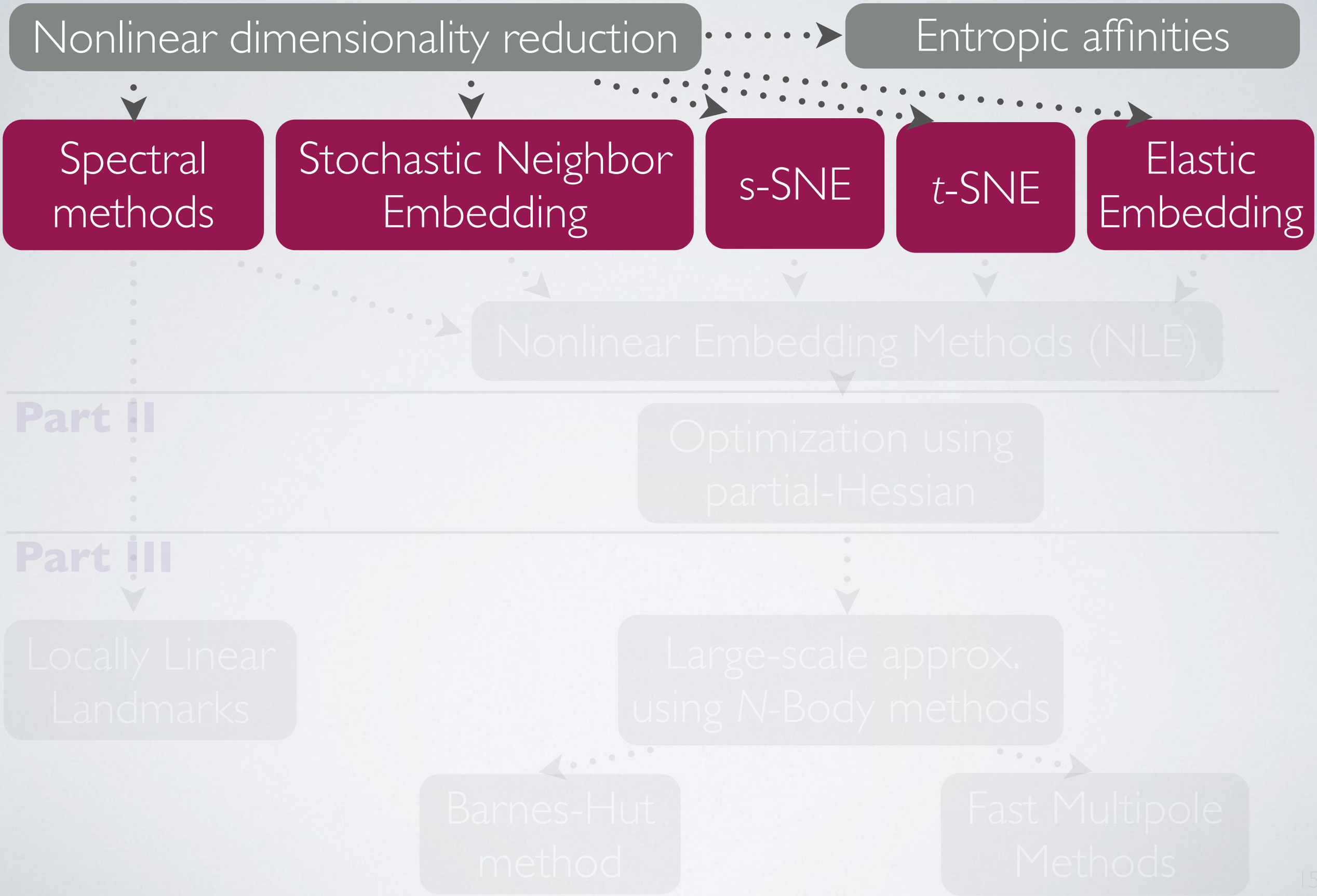
Locally Linear Landmarks

Large-scale approx. using *N*-Body methods

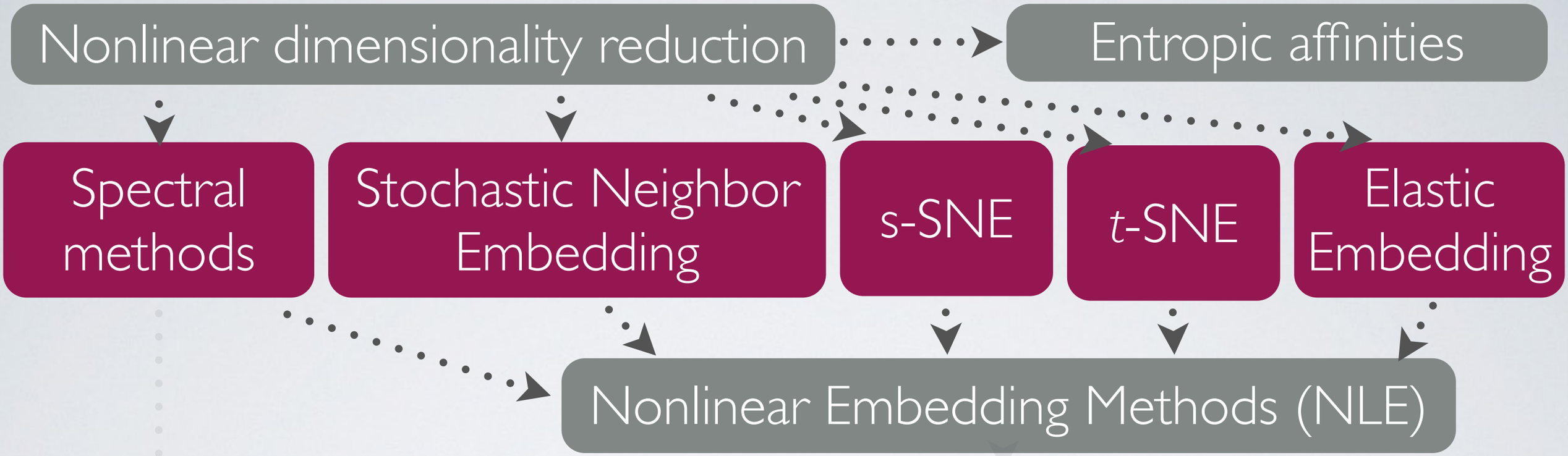
Barnes-Hut method

Fast Multipole Methods

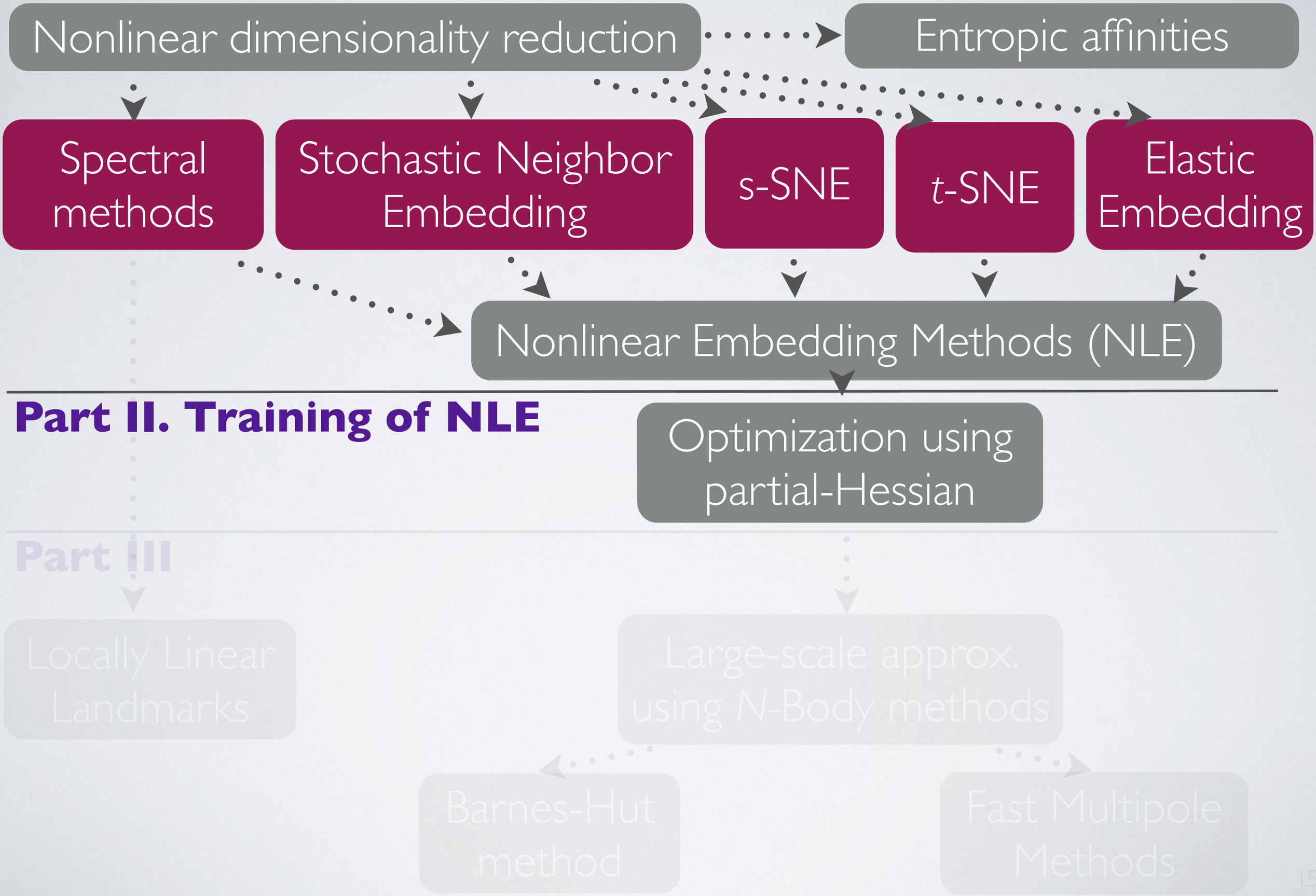
Part I. Nonlinear dimensionality reduction



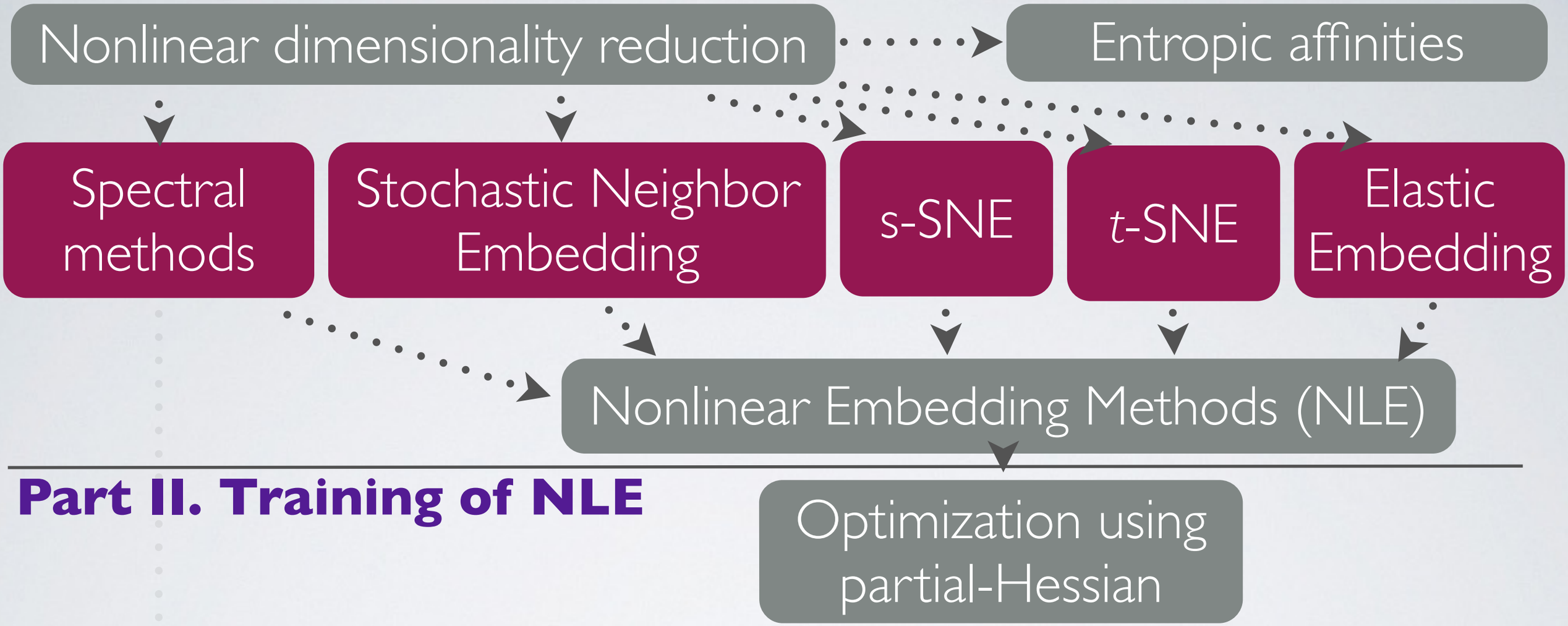
Part I. Nonlinear dimensionality reduction



Part I. Nonlinear dimensionality reduction

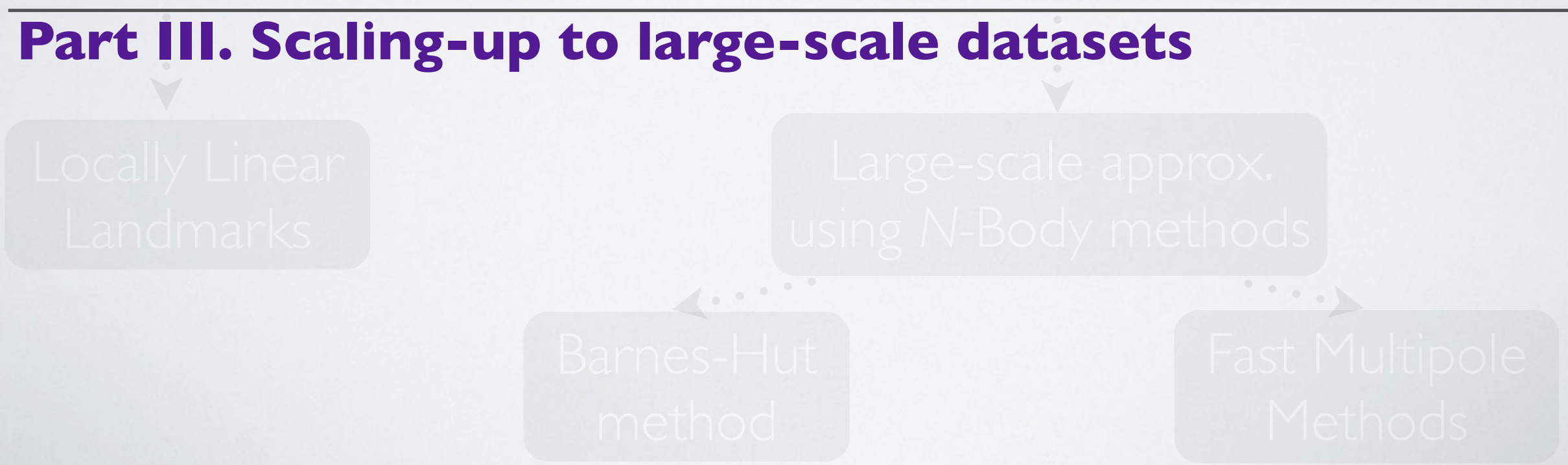


Part I. Nonlinear dimensionality reduction

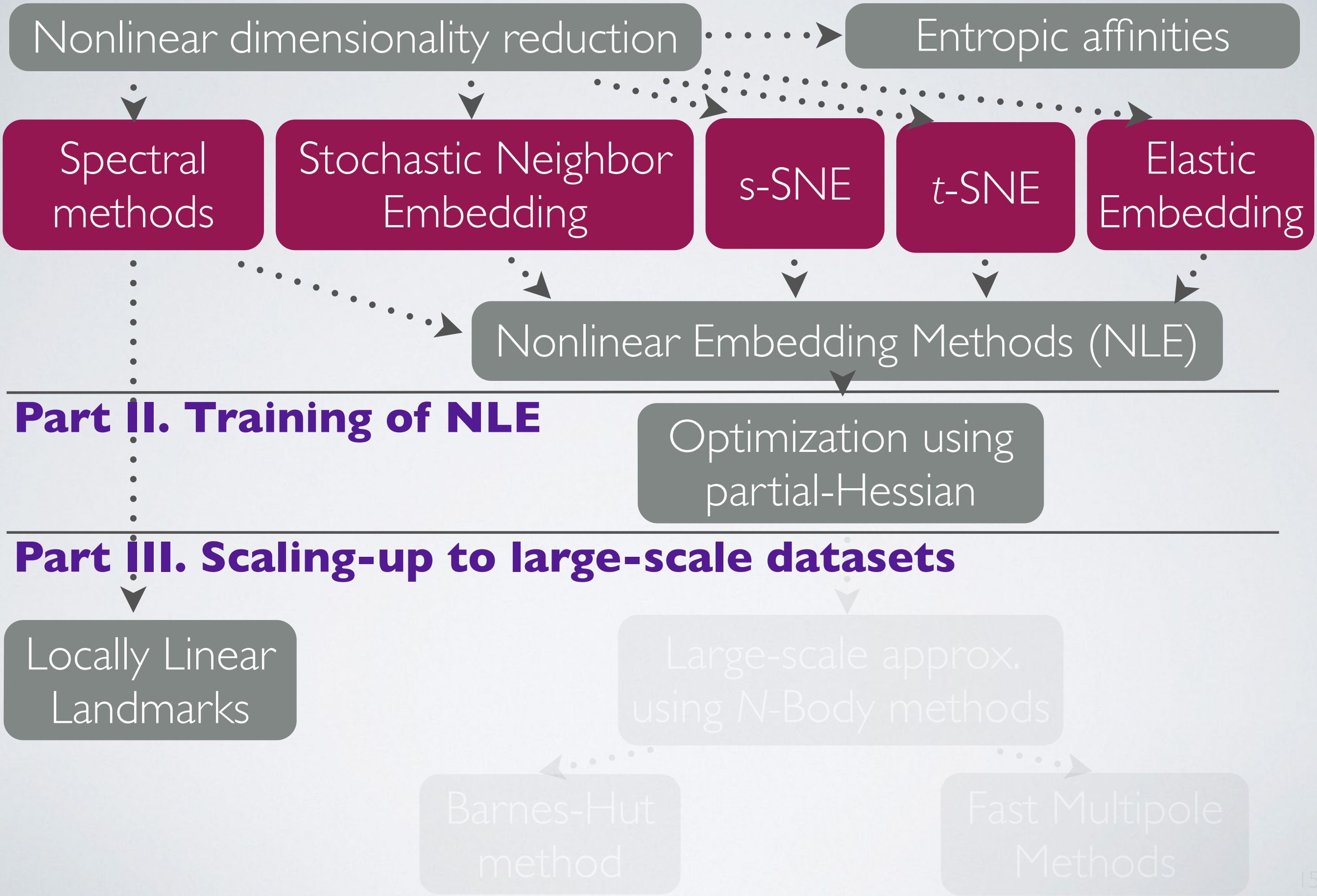


Part II. Training of NLE

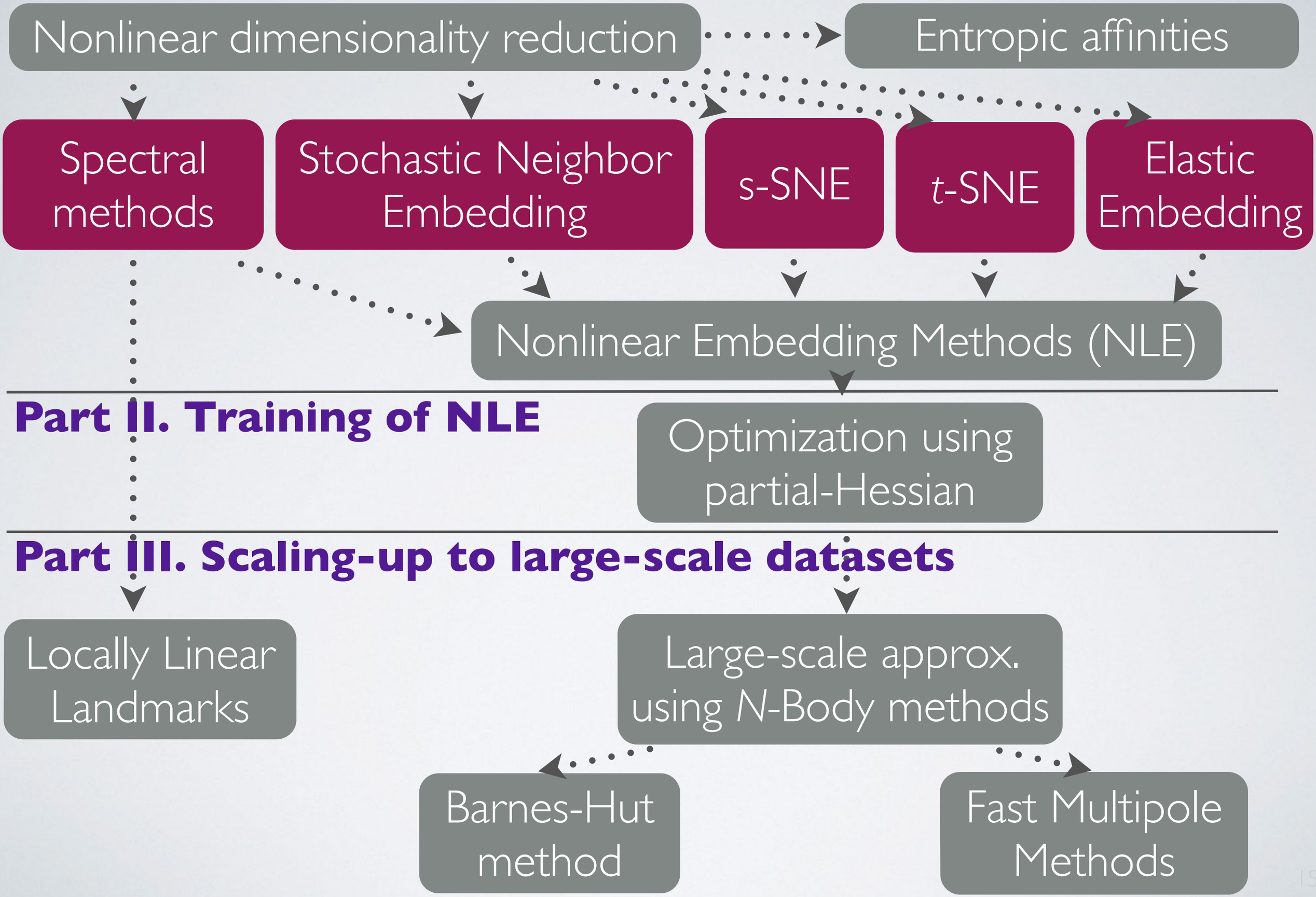
Part III. Scaling-up to large-scale datasets



Part I. Nonlinear dimensionality reduction



Part I. Nonlinear dimensionality reduction

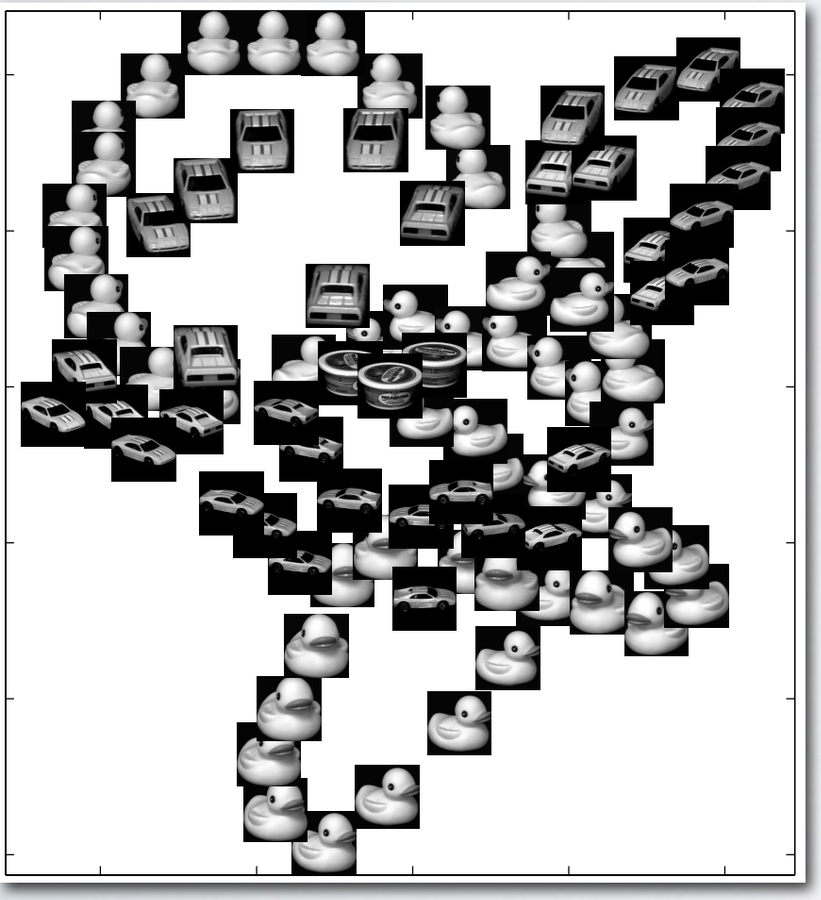


Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.

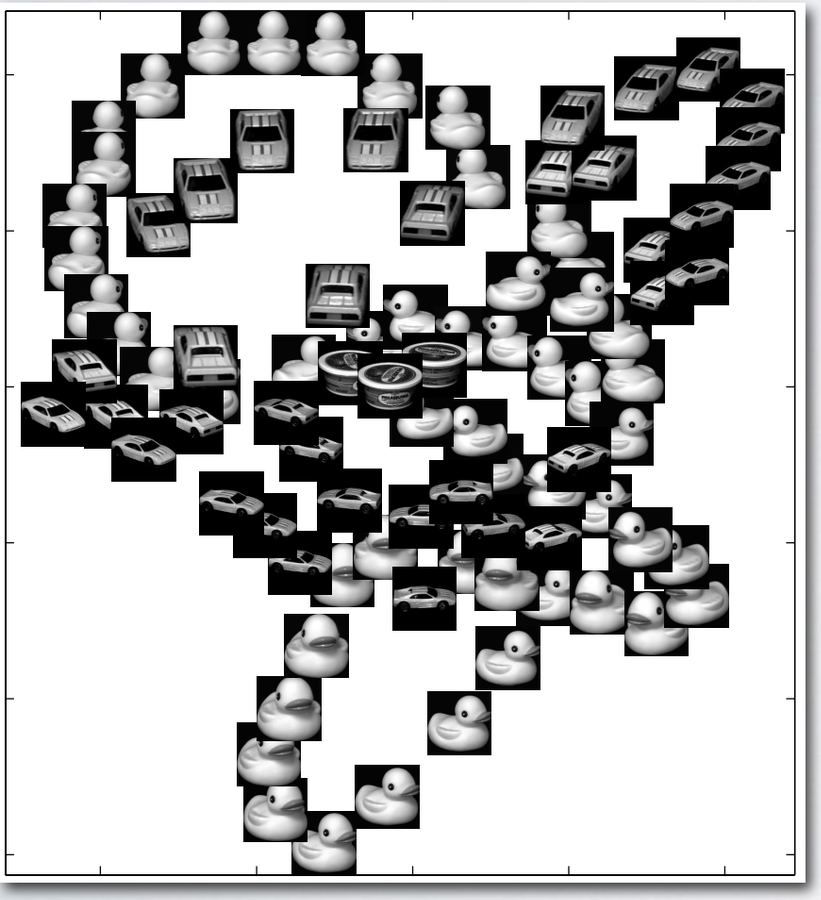
Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.



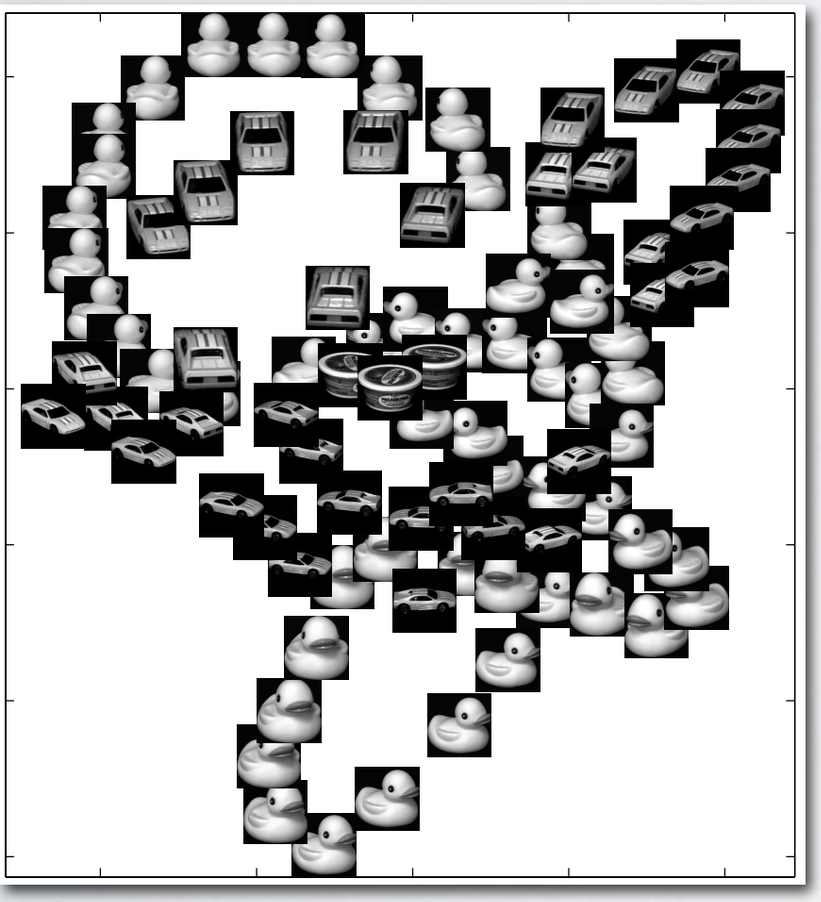
Classification of dimensionality reduction

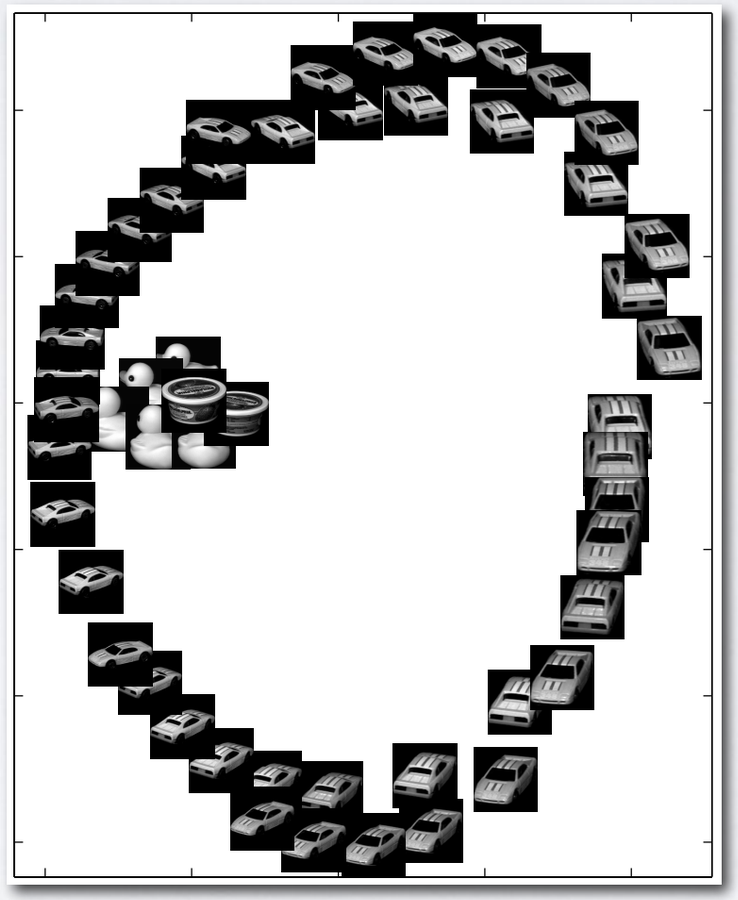
- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.



Classification of dimensionality reduction

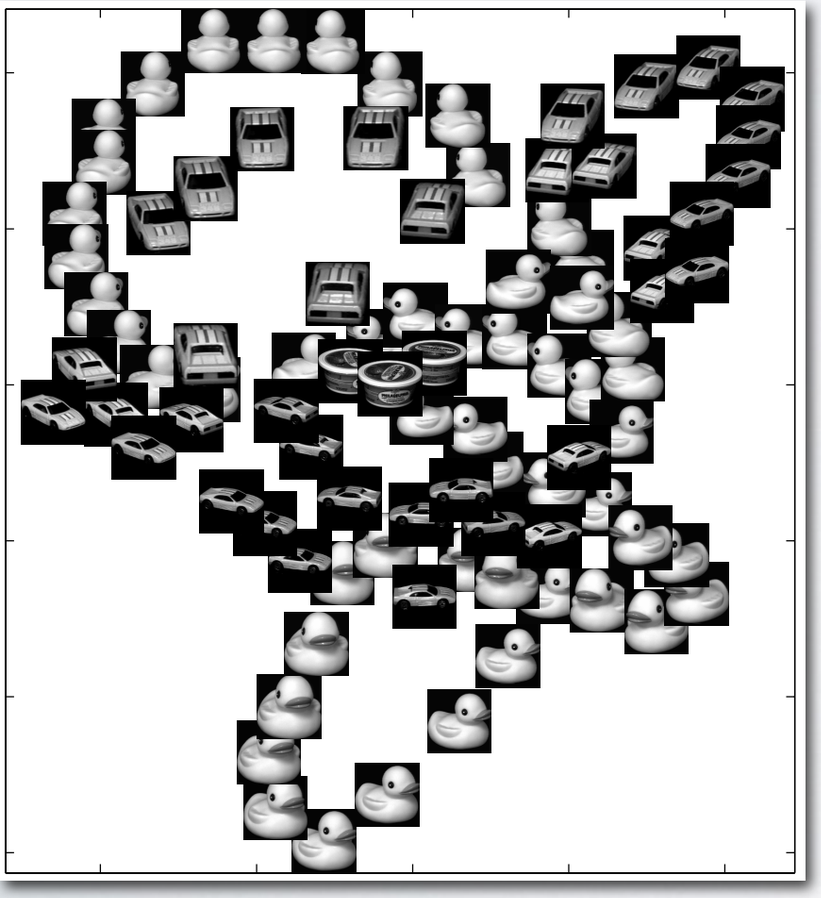
- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.

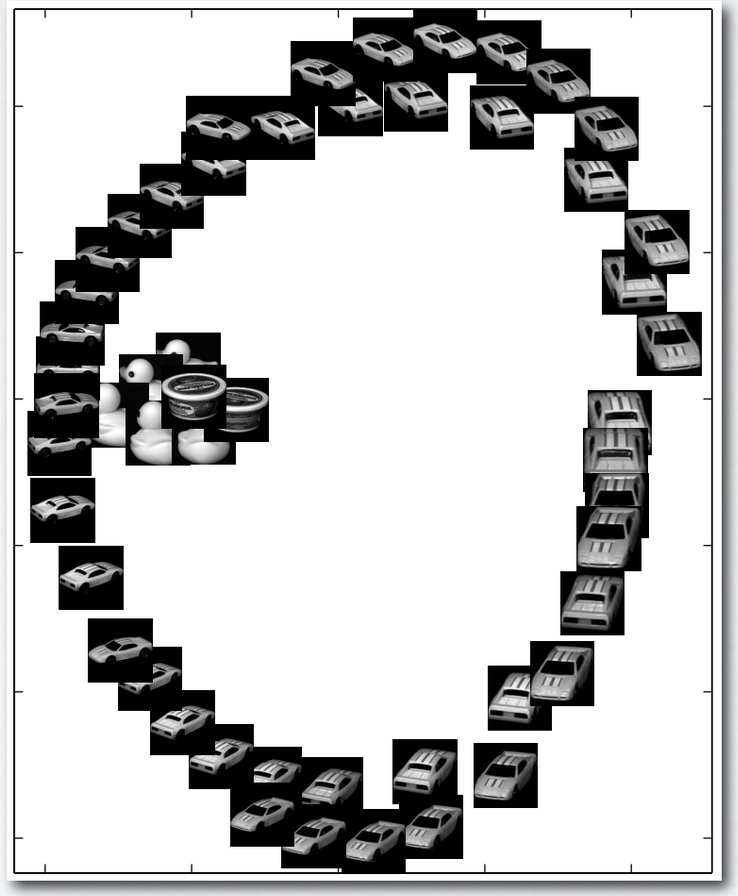




Classification of dimensionality reduction

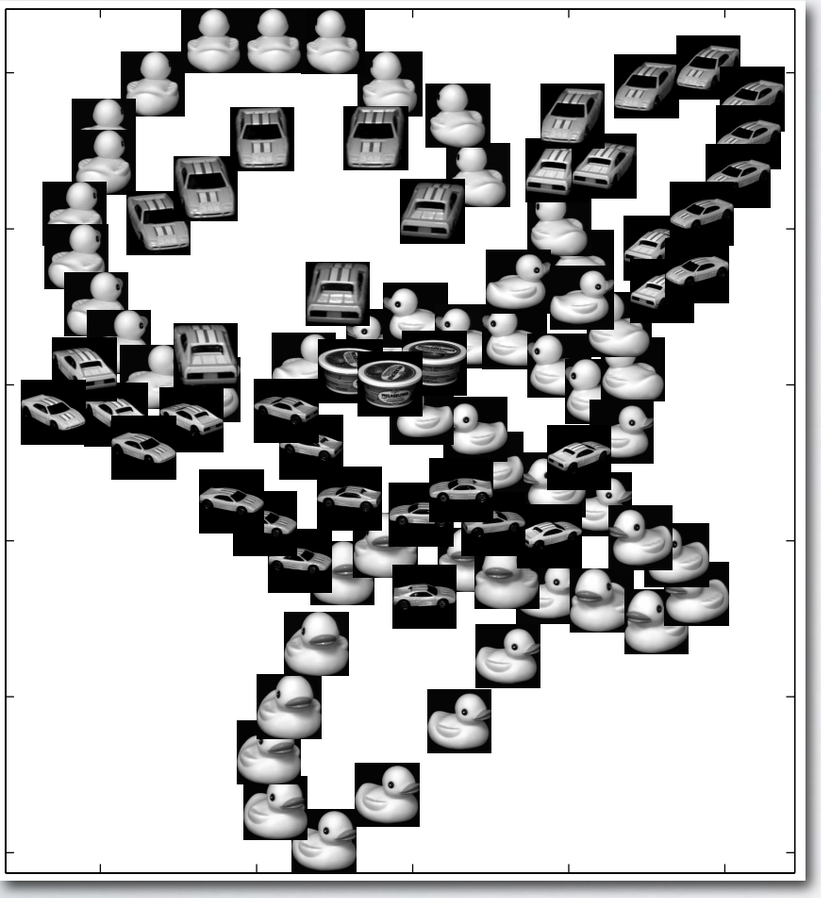
- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.
- Nonlinear embedding methods
  - Stochastic Neighbor Embedding,
  - *t*-SNE,
  - The Elastic Embedding (EE),
  - etc.

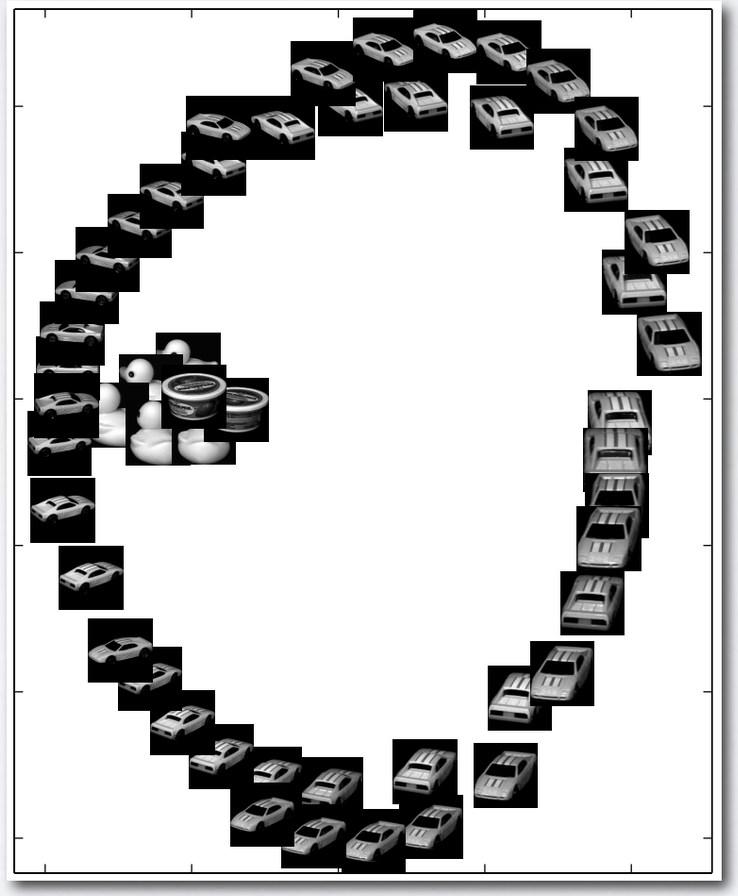


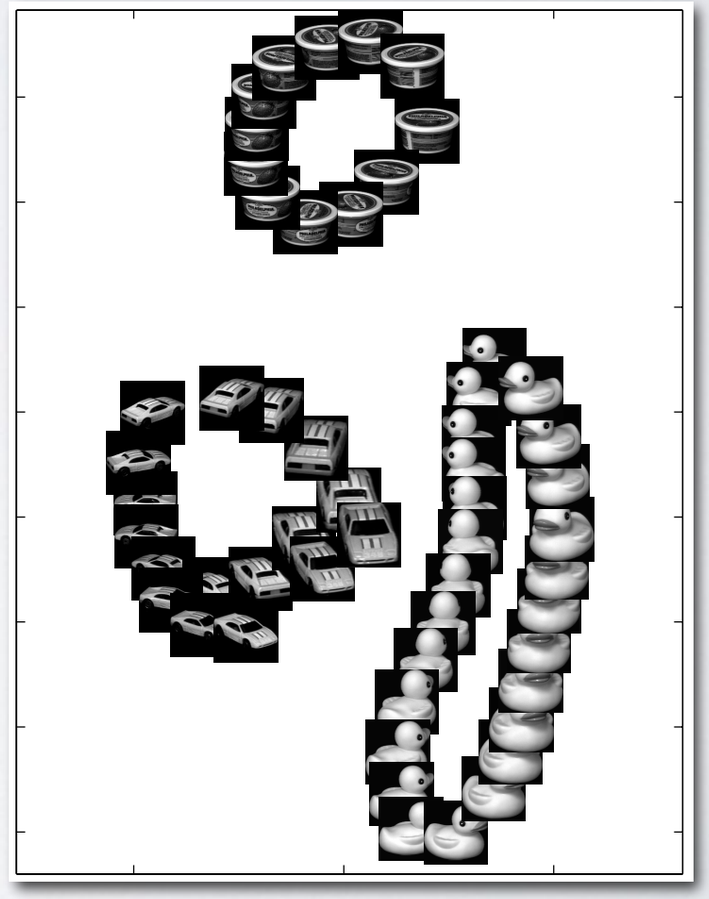


Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.
- Nonlinear embedding methods
  - Stochastic Neighbor Embedding,
  - *t*-SNE,
  - The Elastic Embedding (EE),
  - etc.

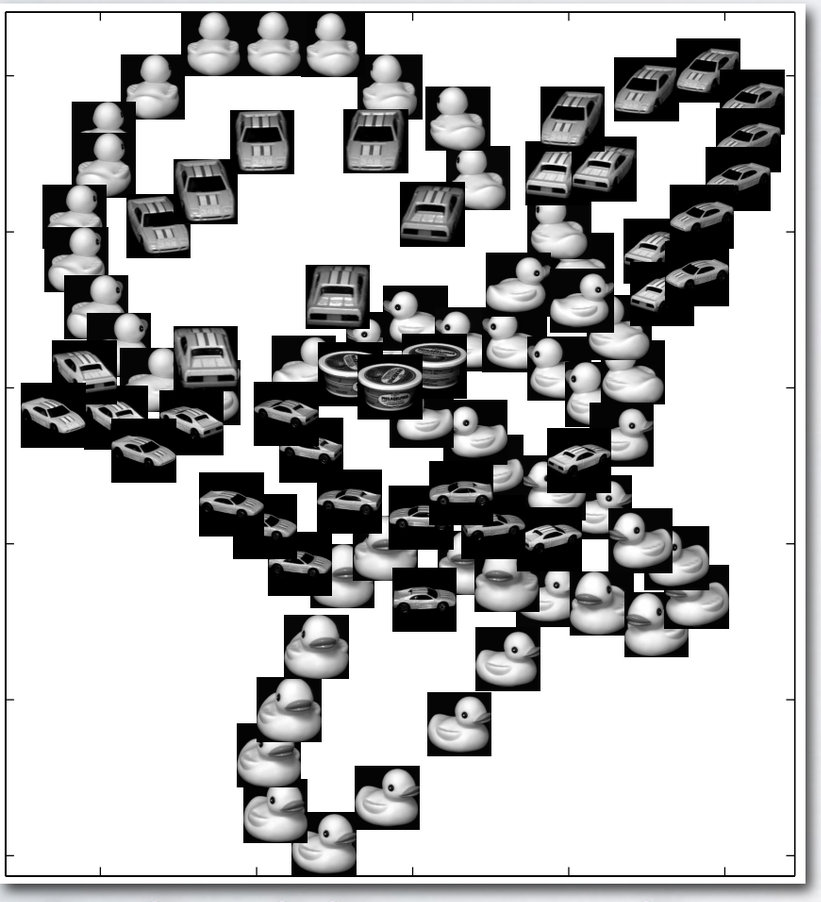


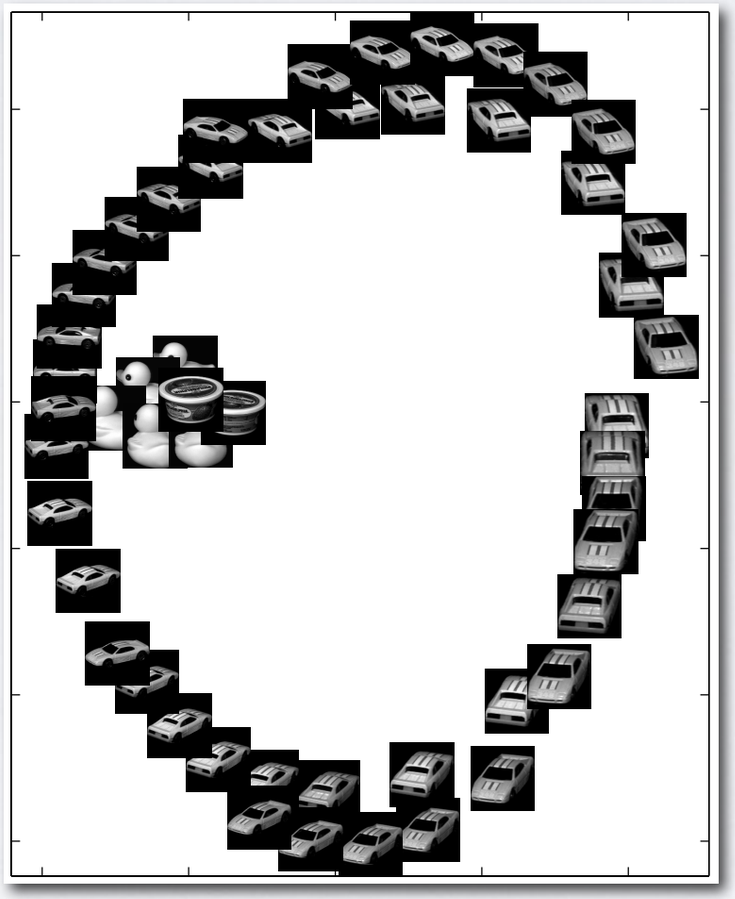


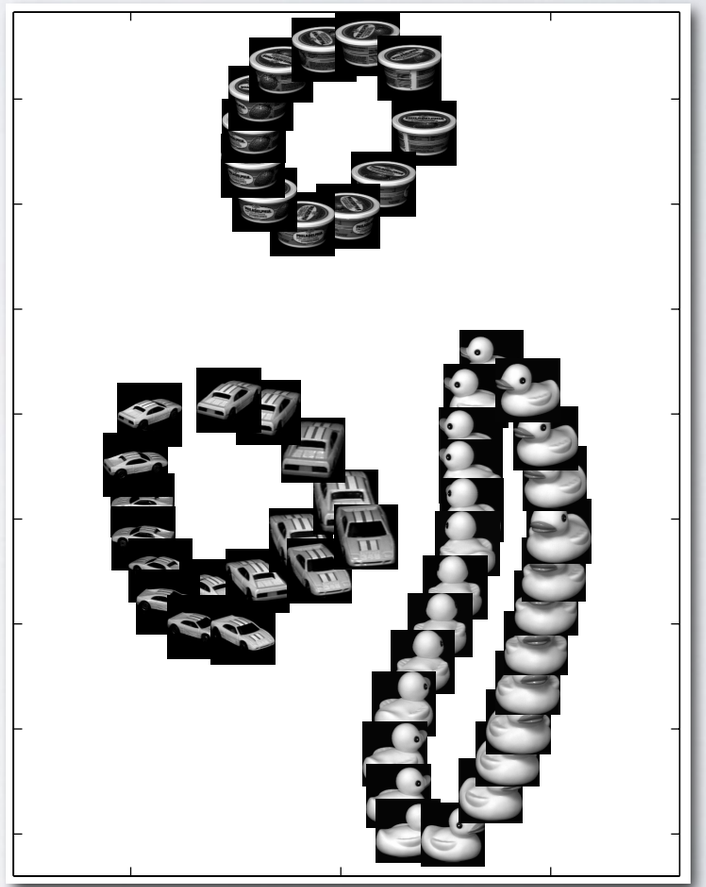


Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.
- Nonlinear embedding methods
  - Stochastic Neighbor Embedding,
  - *t*-SNE,
  - The Elastic Embedding (EE),
  - etc.



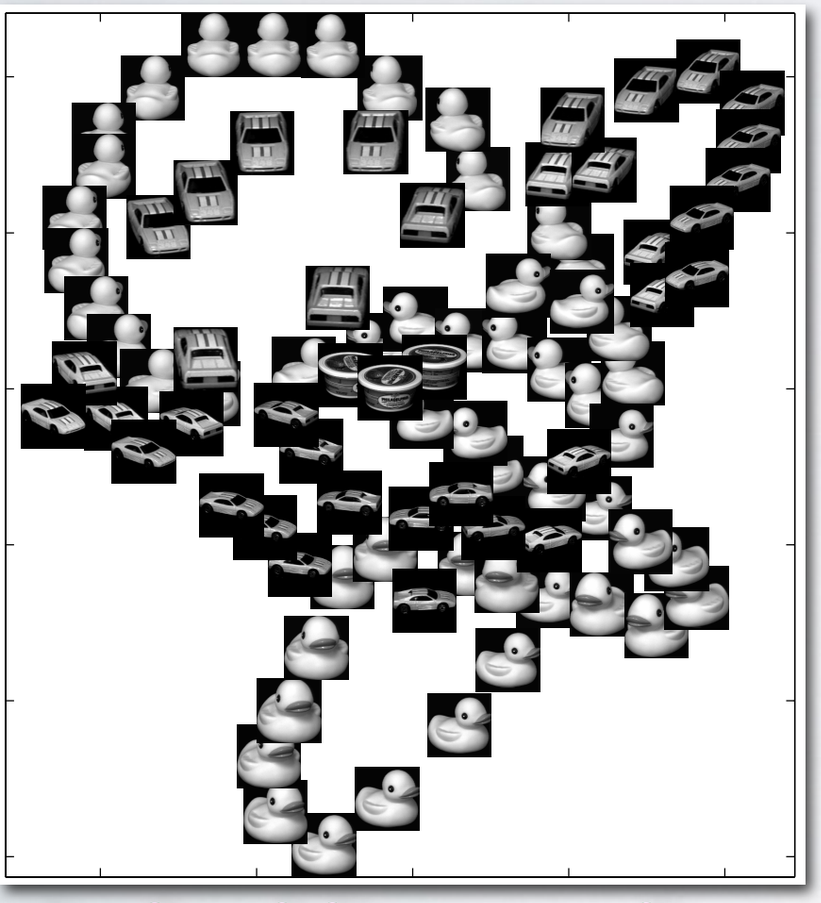


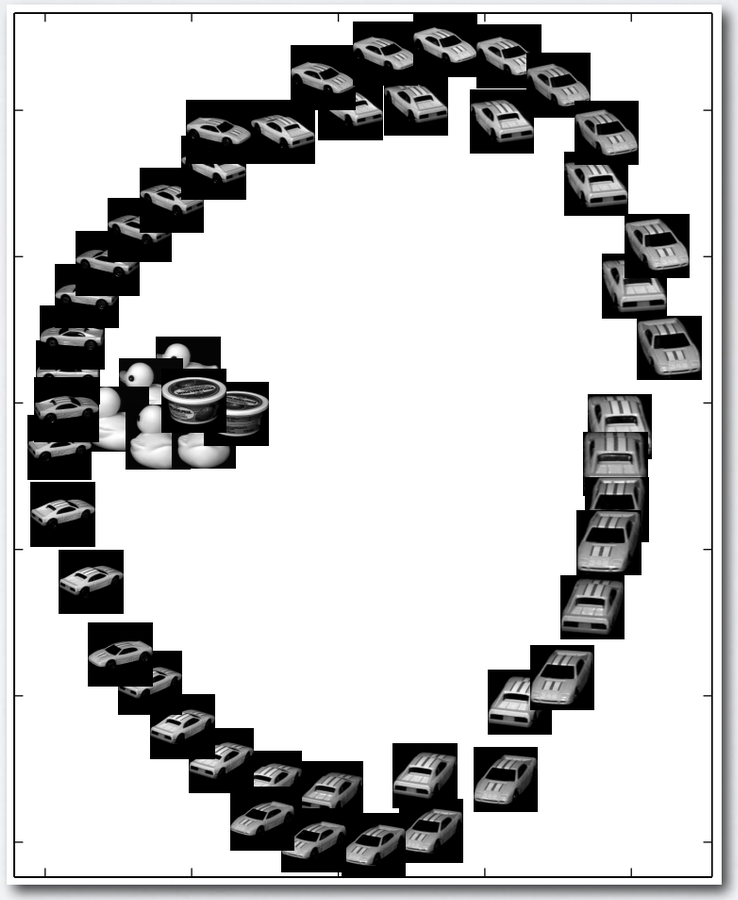


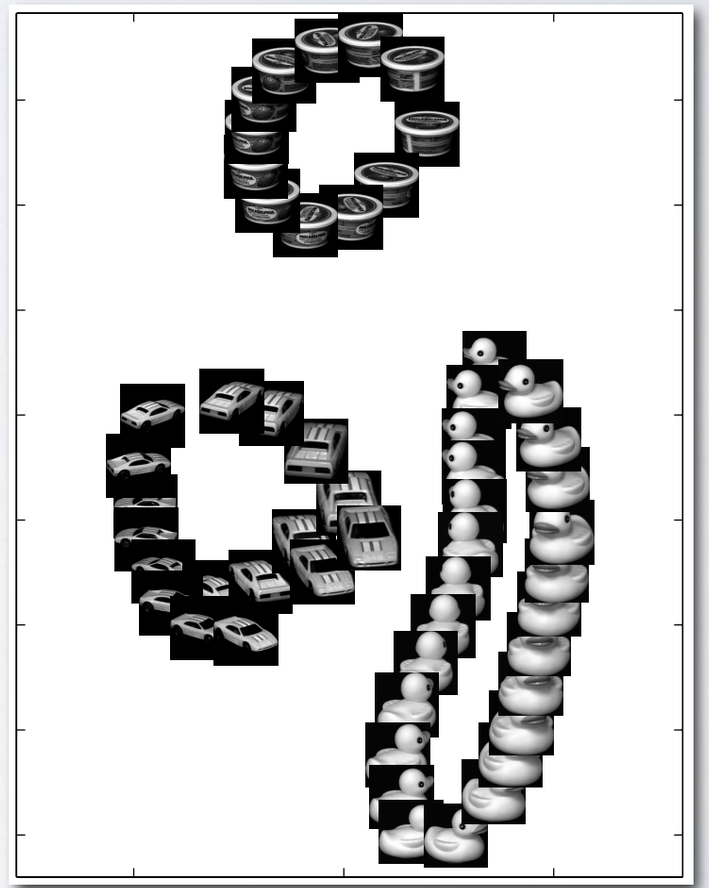
Embedding quality

Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.
- Nonlinear embedding methods
  - Stochastic Neighbor Embedding,
  - *t*-SNE,
  - The Elastic Embedding (EE),
  - etc.







Embedding quality

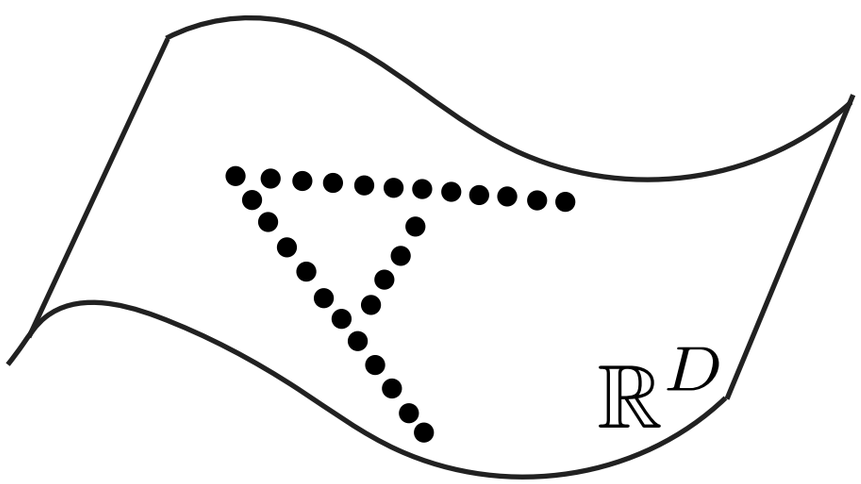
Runtime

Graph-based dimensionality reduction

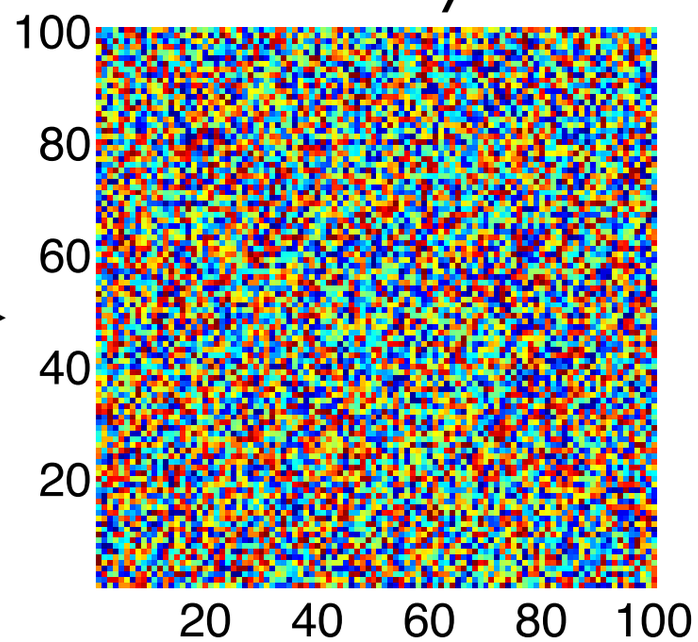
Given high-dimensional data points $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$.

1. Convert data points to a $N \times N$ *affinity* matrix $\mathbf{A}$.
2. Find low-dimensional coordinates $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, so that their similarity is as close as possible to $\mathbf{A}$.

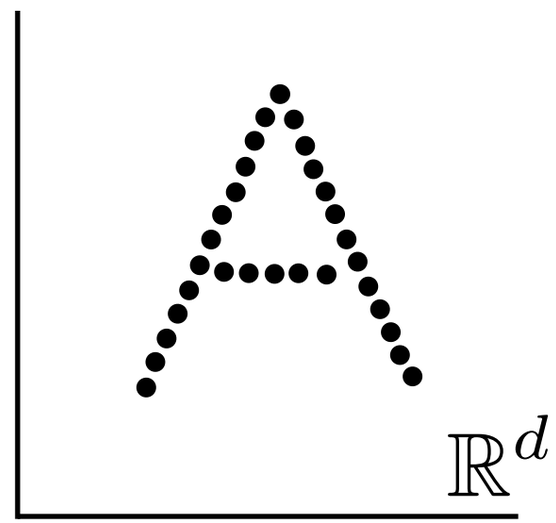
High-dimensional input $\mathbf{Y}$



Affinity $\mathbf{A}$



Low-dimensional output $\mathbf{X}$

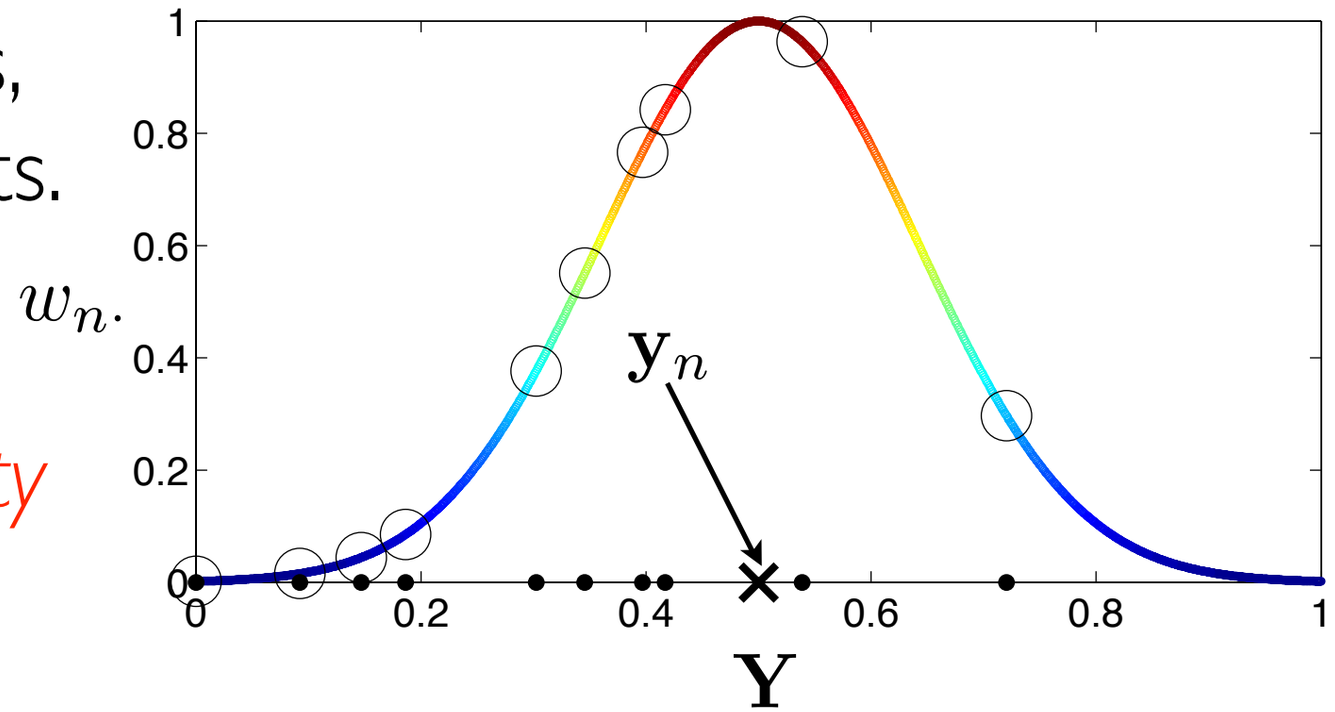


Affinity matrix

- Affinity matrix $W \in \mathbb{R}^{N \times N}$ represents the similarities between points in the dataset. The higher the affinity value, the more similar are the points to each other.

- Intuition:
  - high weight to nearby points,
  - low weight to far away points.

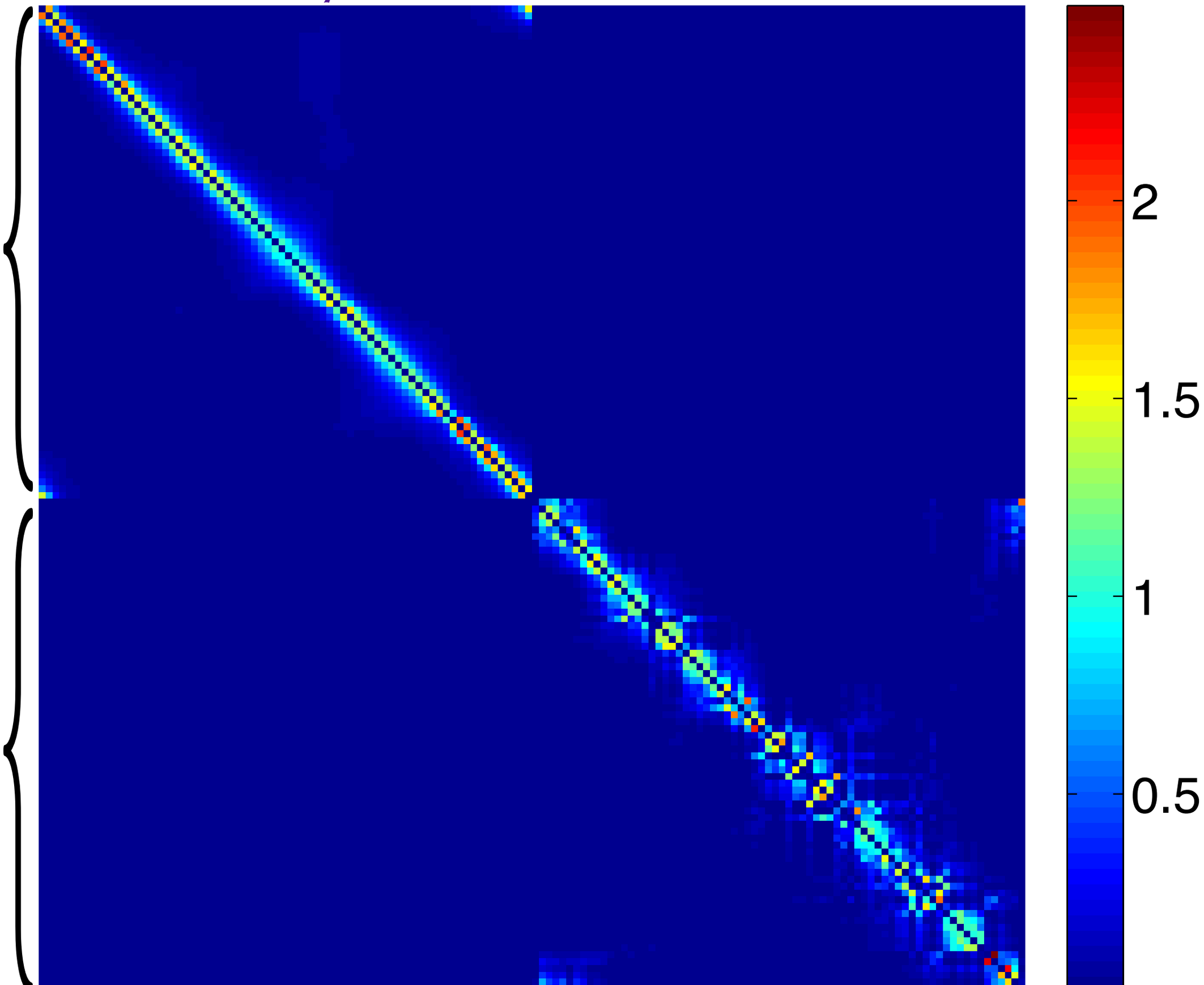
- Property:
  - affinity matrix enforces *locality* of the data.



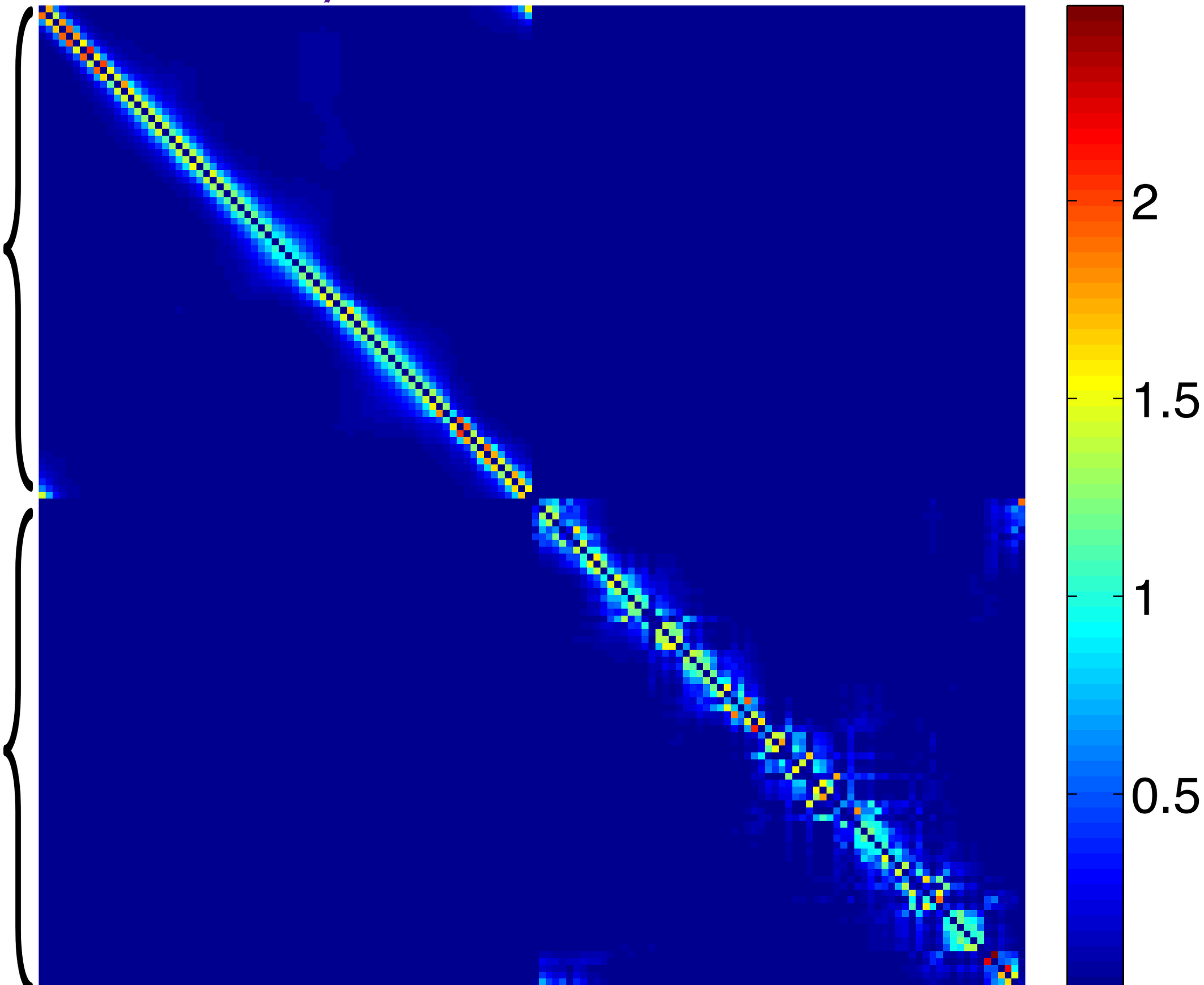
- For example, Gaussian affinities are given by:

$$w_{nm} = \exp(-\frac{1}{2} \left\| (\mathbf{y}_n - \mathbf{y}_m)/\sigma \right\|^2)$$

Gaussian affinity matrix

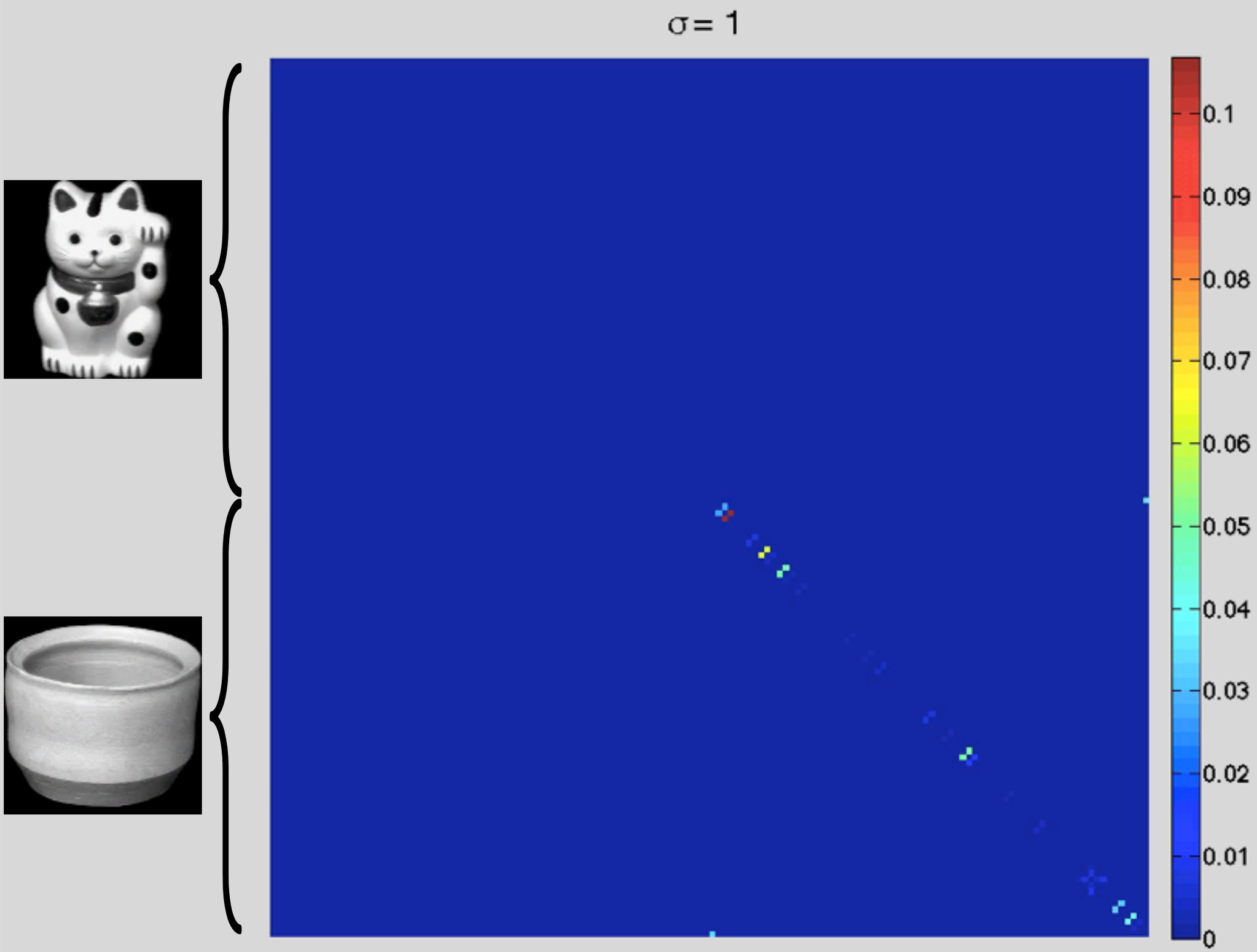


Gaussian affinity matrix



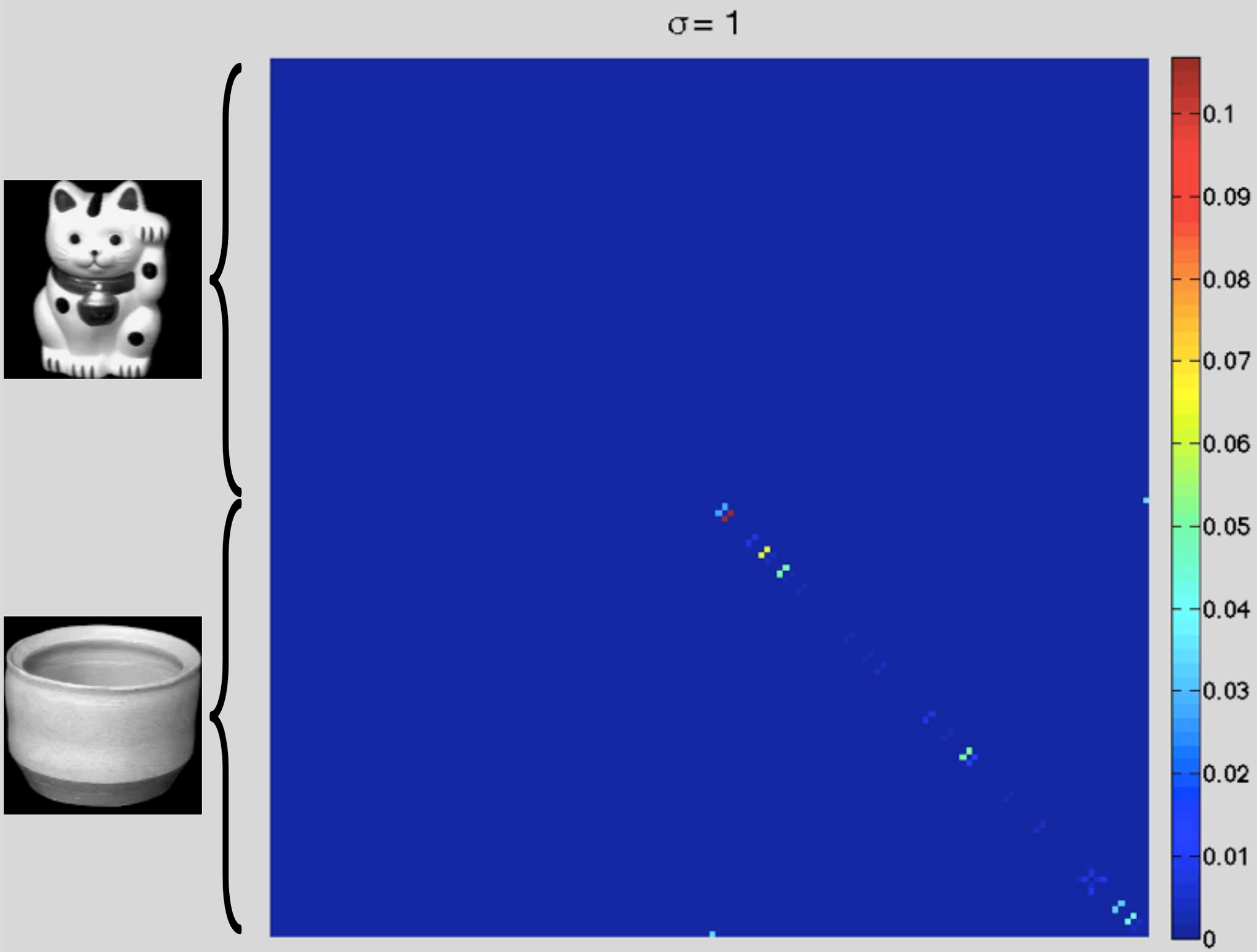
Gaussian affinity matrix: problem with σ

$$w_{nm} = \exp(-\frac{1}{2} \left\| (\mathbf{y}_n - \mathbf{y}_m)/\sigma \right\|^2)$$



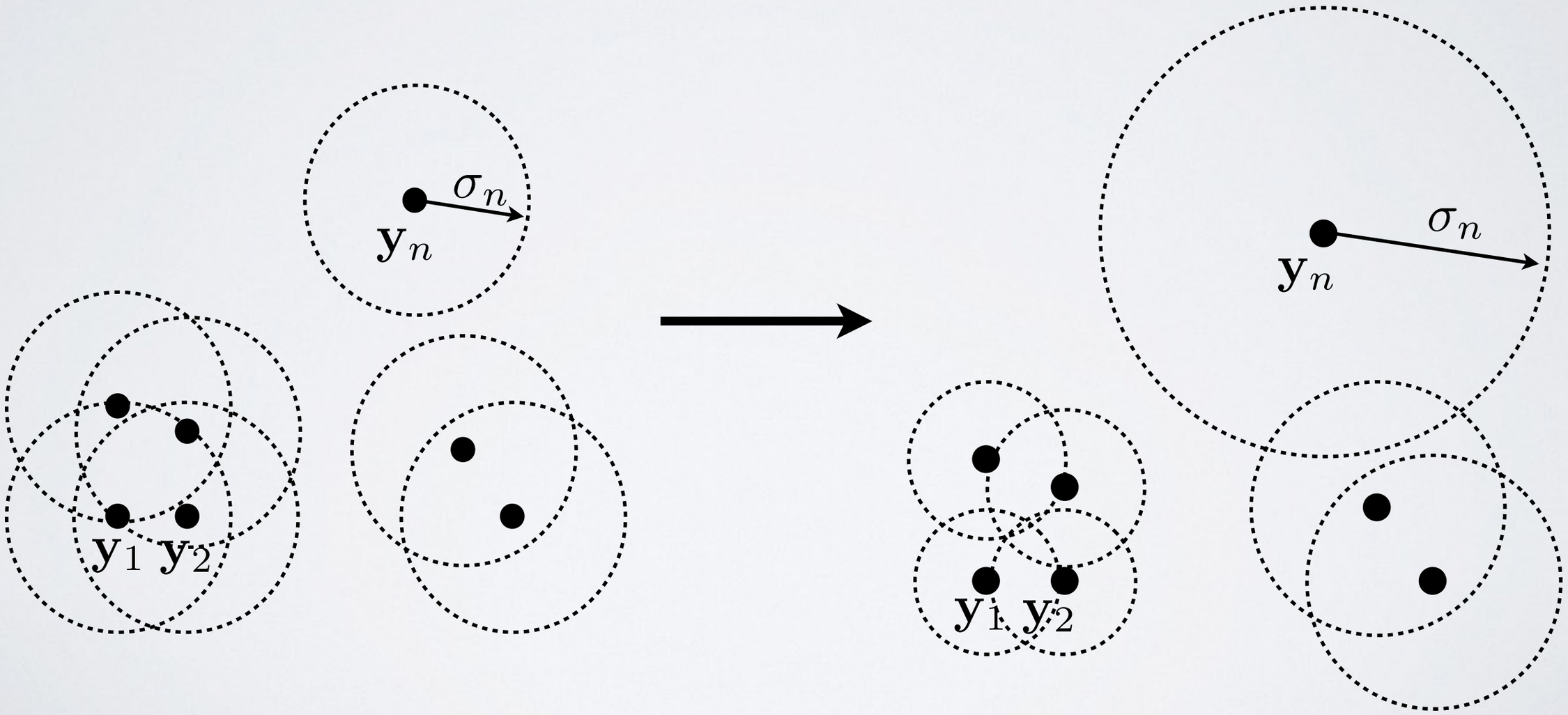
Gaussian affinity matrix: problem with σ

$$w_{nm} = \exp(-\frac{1}{2} \left\| (\mathbf{y}_n - \mathbf{y}_m)/\sigma \right\|^2)$$



Gaussian affinity matrix

- Good σ_n should be:
  - set separately for every data point,
  - take into account whole distribution of distances.
- σ_n represents *spatial* characteristic of the data, which is not intuitive and is hard to set (especially for every point).



Entropic affinities (Vladymyrov and Carreira-Perpiñán, '13)

For entropic affinities, σ is set individually for each point such that it has a distribution over neighbors with fixed perplexity K.

(Hinton & Rowies, 2003)

- Consider a distribution of the neighbors $\mathbf{y}_1, \dots, \mathbf{y}_n \in \mathbb{R}^D$ for $\mathbf{y} \in \mathbb{R}^D$

$$p_n(\mathbf{y}, \sigma) = \frac{K(\|(\mathbf{y} - \mathbf{y}_n)/\sigma\|^2)}{\sum_{k=1}^{N} K(\|(\mathbf{y} - \mathbf{y}_k)/\sigma\|^2)}$$

posterior distribution of Kernel Density Estimate.

- The entropy of the distribution is defined as:

$$H(\mathbf{y}, \sigma) = -\sum_{n=1}^{N} p_n(\mathbf{y}, \sigma) \log(p_n(\mathbf{y}, \sigma))$$

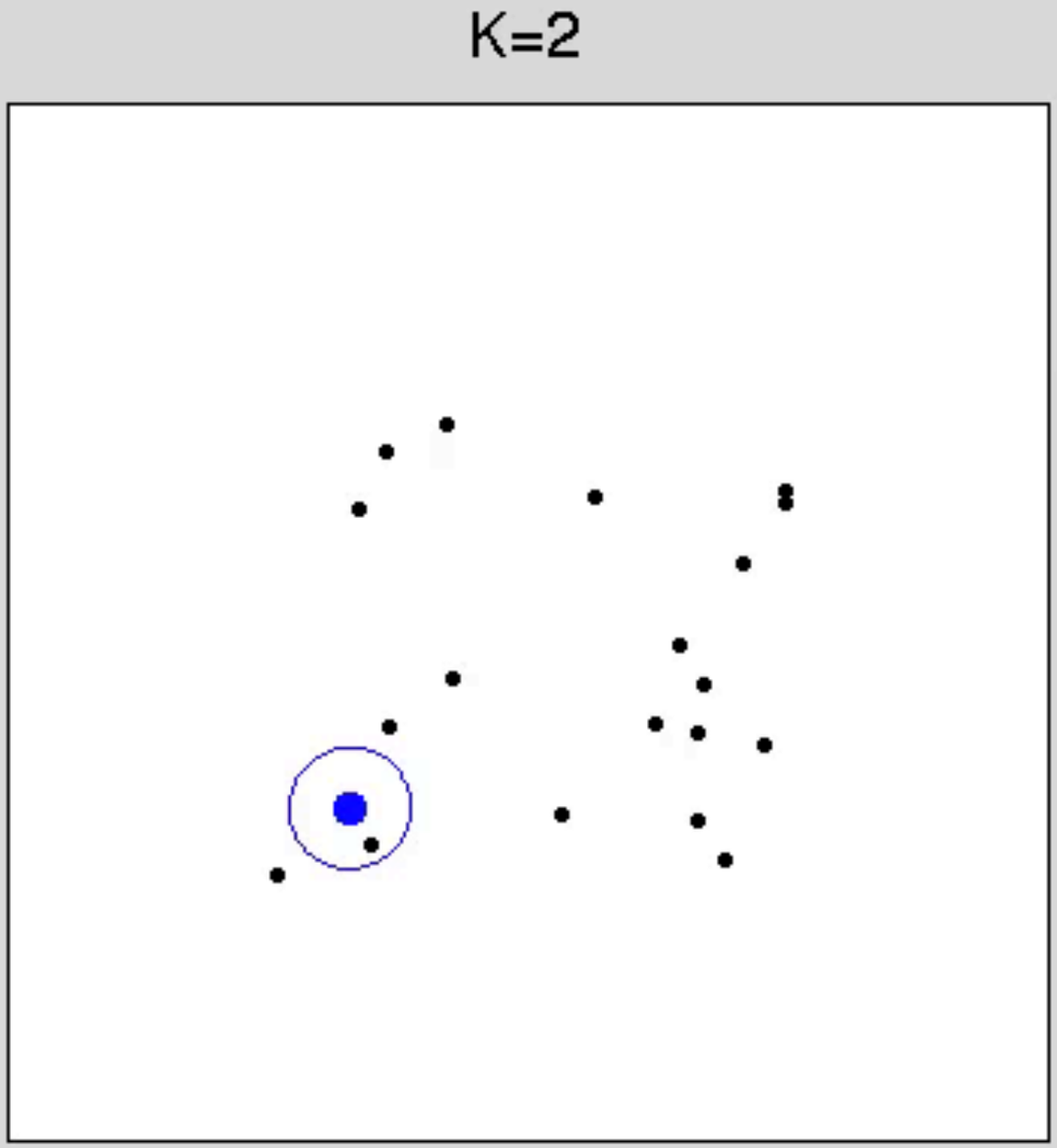
- Consider the bandwidth σ given the perplexity K :

$$H(\mathbf{y}, \sigma) = \log K$$

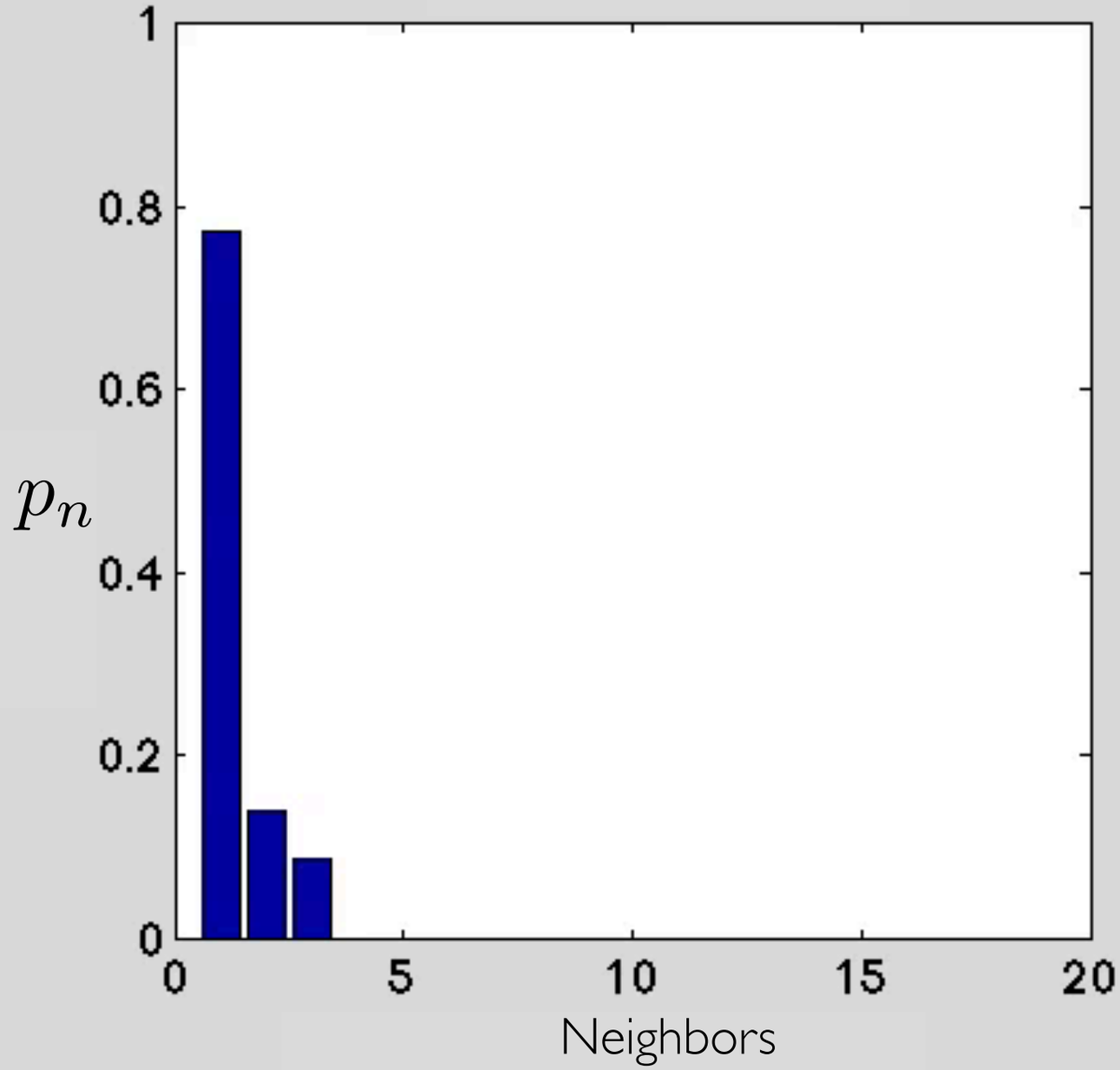
- We define entropic affinities as probabilities $p = (p_1, \dots, p_N)$ for $\mathbf{y}$ with respect to σ. These affinities define a random walk matrix.

Entropic affinities

Perplexity of K in a distribution p over N neighbors provides the same surprise as if we were to choose among K equiprobable neighbors.

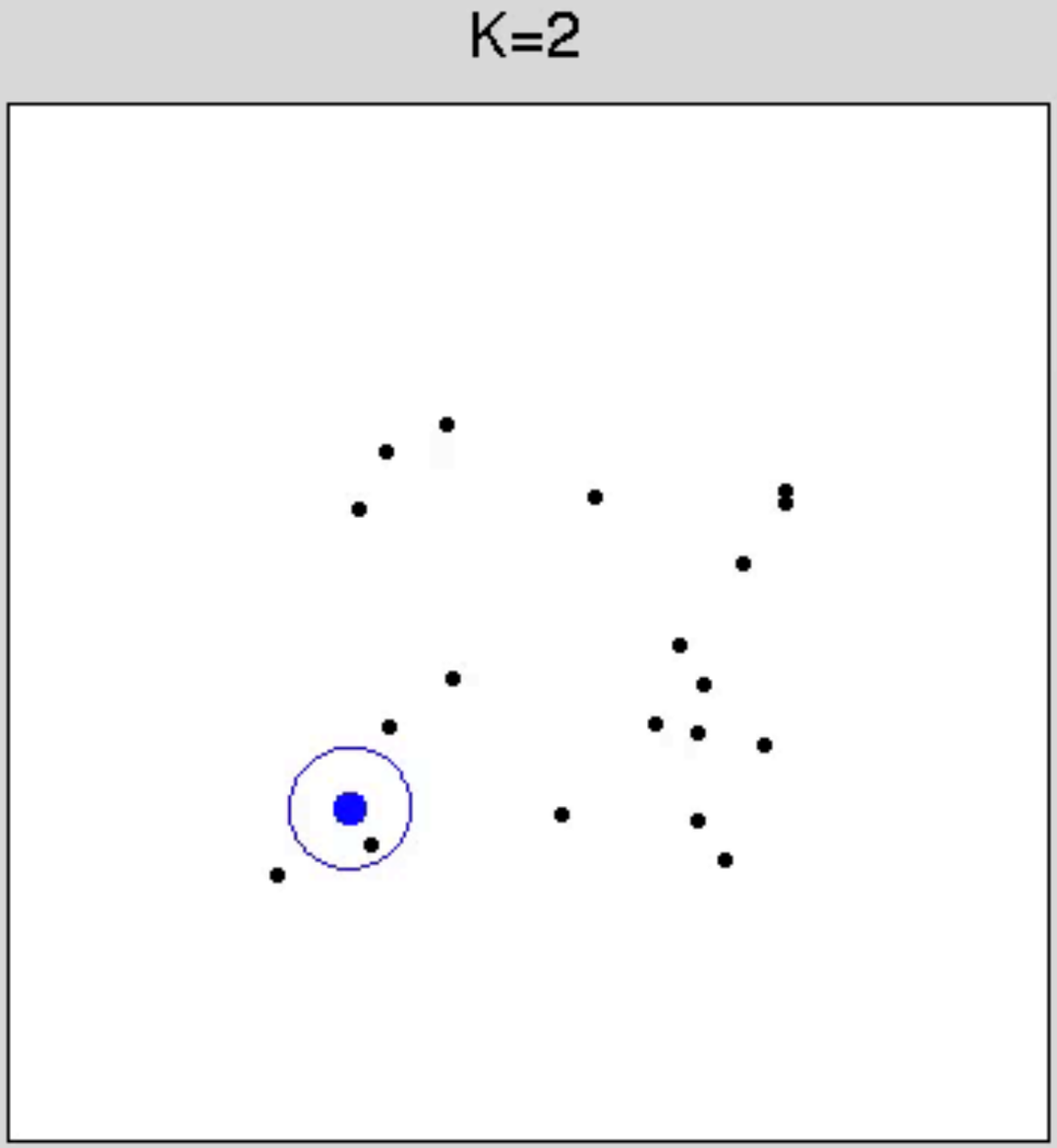


radius of the circle corresponds to σ

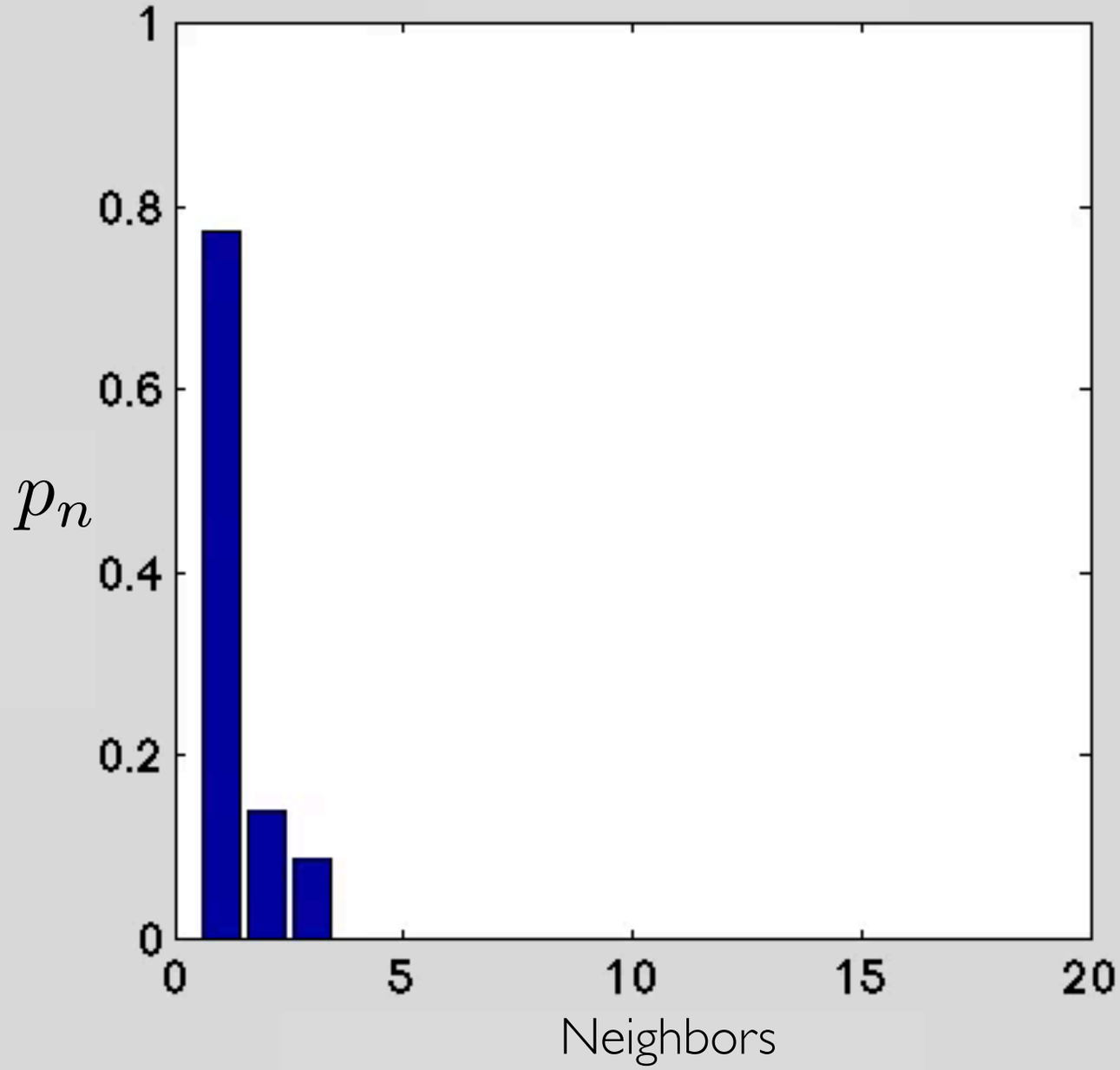


Entropic affinities

Perplexity of K in a distribution p over N neighbors provides the same surprise as if we were to choose among K equiprobable neighbors.



radius of the circle corresponds to σ



Entropic affinities (computation)

$H(\mathbf{y}_n, \sigma_n) = \log K$ defines a root-finding problem for σ_n.

- The problem is well defined for a Gaussian kernel for any σ_n and has a $K \in (0, N)$
- There exists σ_n in constant time.
- We can use with the bounds
- We can use points.

$\Downarrow$

- We can solve for σ_n in just almost machine precision ($tol = 10^{-15}$)

Entropic affinities (computation)

$H(\mathbf{y}_n, \sigma_n) = \log K$ defines a root-finding problem for σ_n.

- The problem is well defined for a Gaussian kernel for any σ_n, and has a unique root for any $K \in (0, N)$.
- There exists σ_n in constant time.
- We can use with the bounds
- We can use points.

$\Downarrow$

- We can solve for σ_n in just almost machine precision ($tol = 10^{-15}$)

Entropic affinities (computation)

$H(\mathbf{y}_n, \sigma_n) = \log K$ defines a root-finding problem for σ_n.

- The problem is well defined for a Gaussian kernel for any σ_n, and has a unique root for any $K \in (0, N)$.
- There exists tight bounds for the root σ_n that can be computed in constant time.
- We can use with the bounds
- We can use points.

$\Downarrow$

- We can solve for σ_n in just almost machine precision ($tol = 10^{-15}$)

Entropic affinities (computation)

$H(\mathbf{y}_n, \sigma_n) = \log K$ defines a root-finding problem for σ_n.

- The problem is well defined for a Gaussian kernel for any σ_n, and has a unique root for any $K \in (0, N)$.
- There exists tight bounds for the root σ_n that can be computed in constant time.
- We can use high-order convergence methods that together with the bounds guarantee the convergence of the algorithm.
- We can use
  points.

$\Downarrow$

- We can solve for σ_n in just
  almost machine precision ($tol = 10^{-15}$)

Entropic affinities (computation)

$H(\mathbf{y}_n, \sigma_n) = \log K$ defines a root-finding problem for σ_n.

- The problem is well defined for a Gaussian kernel for any σ_n, and has a unique root for any $K \in (0, N)$.
- There exists tight bounds for the root σ_n that can be computed in constant time.
- We can use high-order convergence methods that together with the bounds guarantee the convergence of the algorithm.
- We can use warm-start initialization based on the order of the points.

$\Downarrow$

- We can solve for σ_n in just almost machine precision ($tol = 10^{-15}$)

Entropic affinities (computation)

$H(\mathbf{y}_n, \sigma_n) = \log K$ defines a root-finding problem for σ_n.

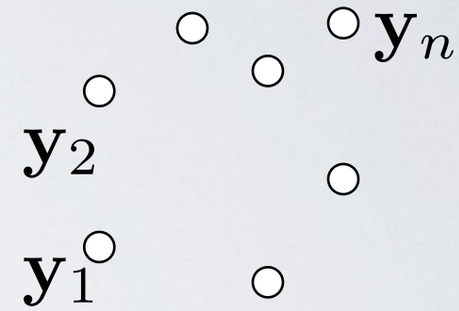
- The problem is well defined for a Gaussian kernel for any σ_n, and has a unique root for any $K \in (0, N)$.
- There exists tight bounds for the root σ_n that can be computed in constant time.
- We can use high-order convergence methods that together with the bounds guarantee the convergence of the algorithm.
- We can use warm-start initialization based on the order of the points.

$$\Downarrow$$

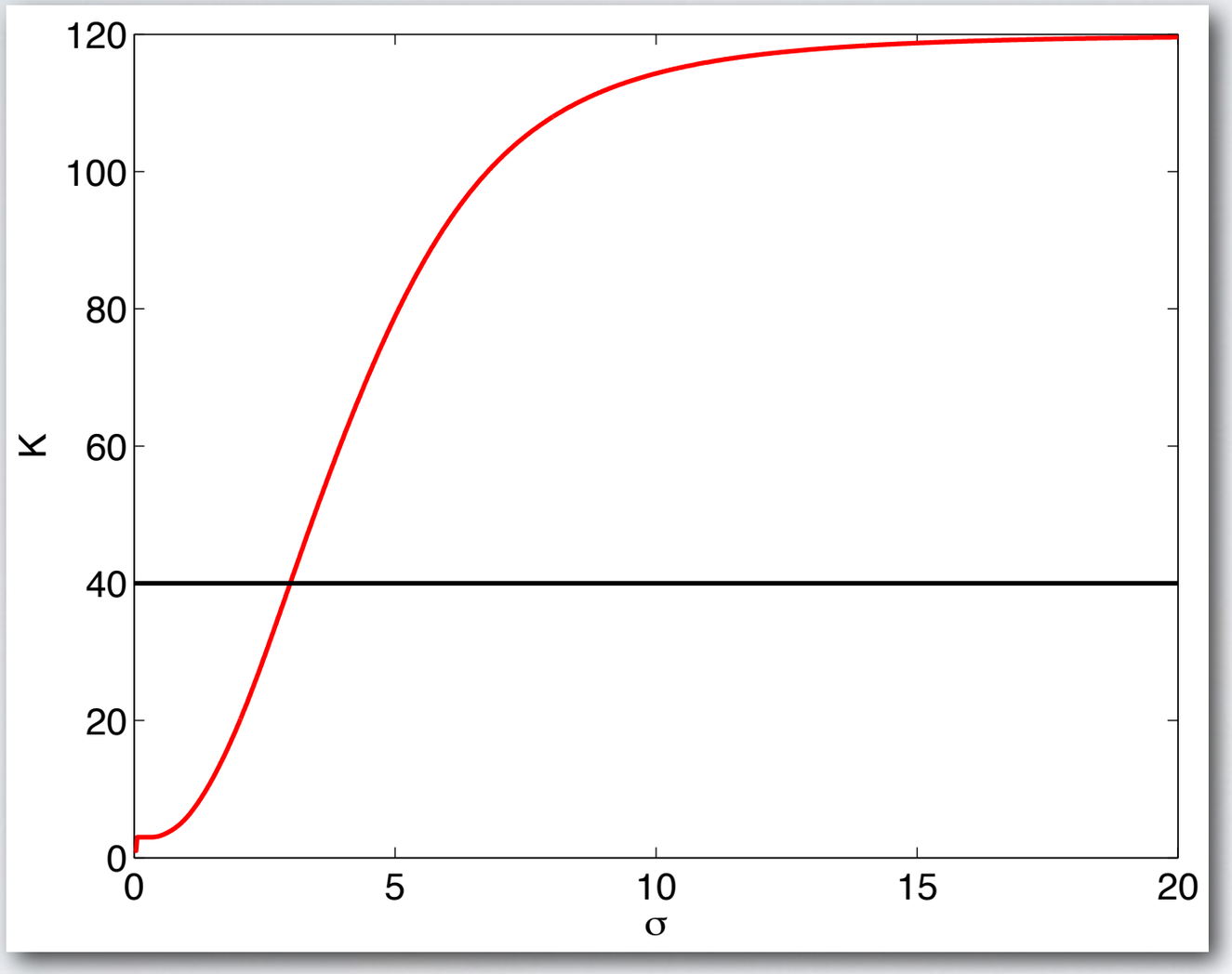
- We can solve for σ_n in just over one iteration per point to almost machine precision ($tol = 10^{-15}$).

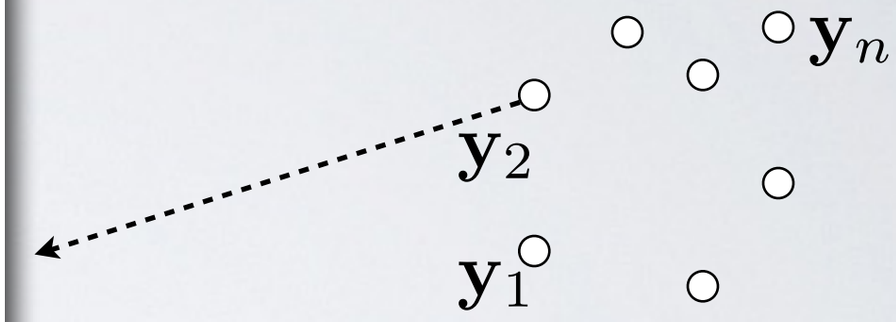
Entropic affinities (computation)

Entropic affinities (computation)

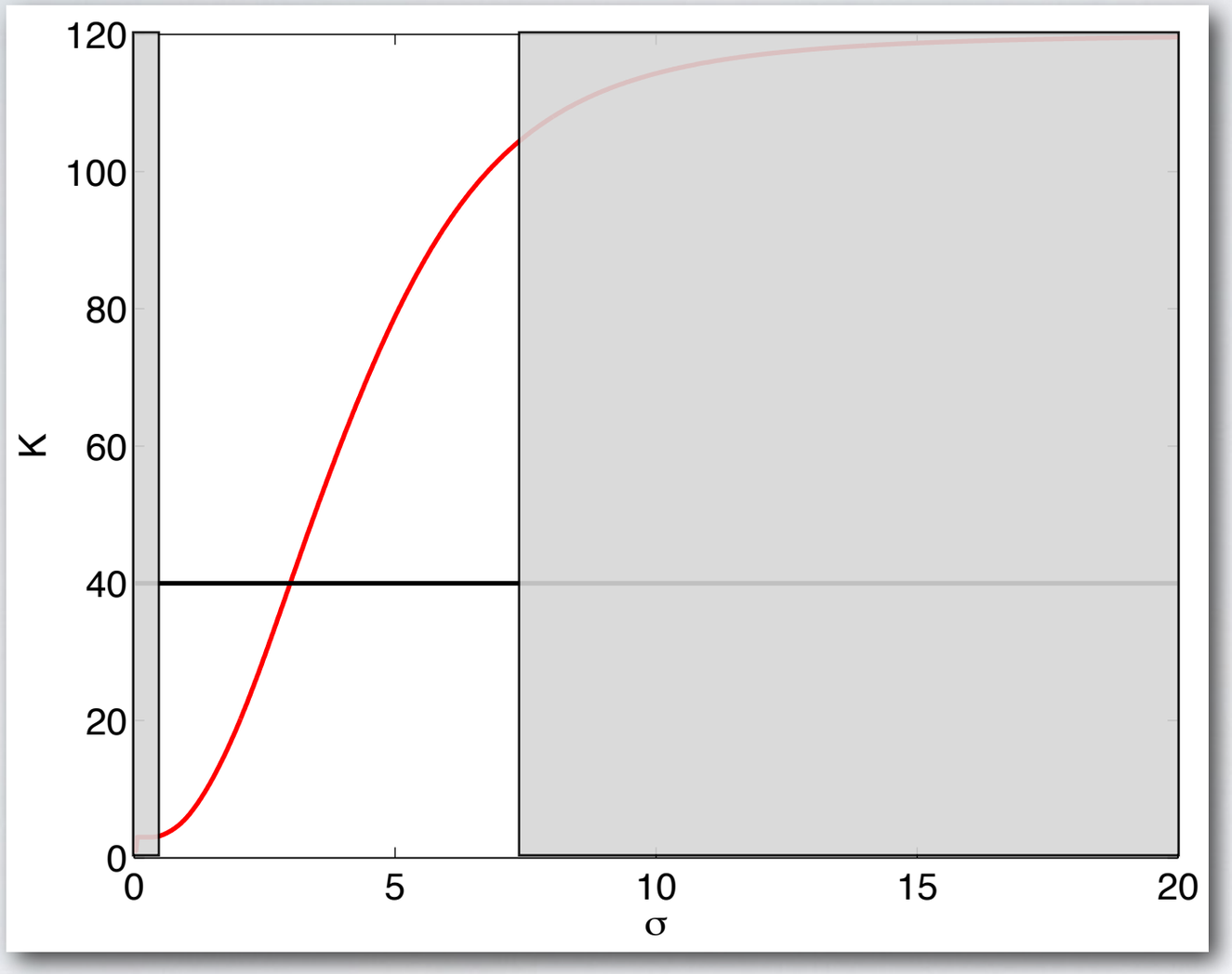


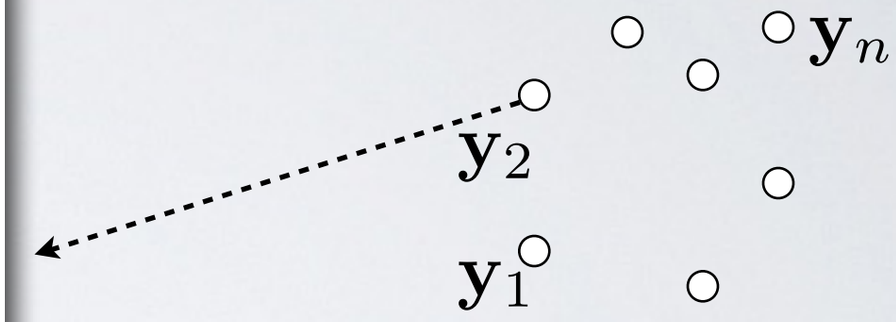
Entropic affinities (computation)



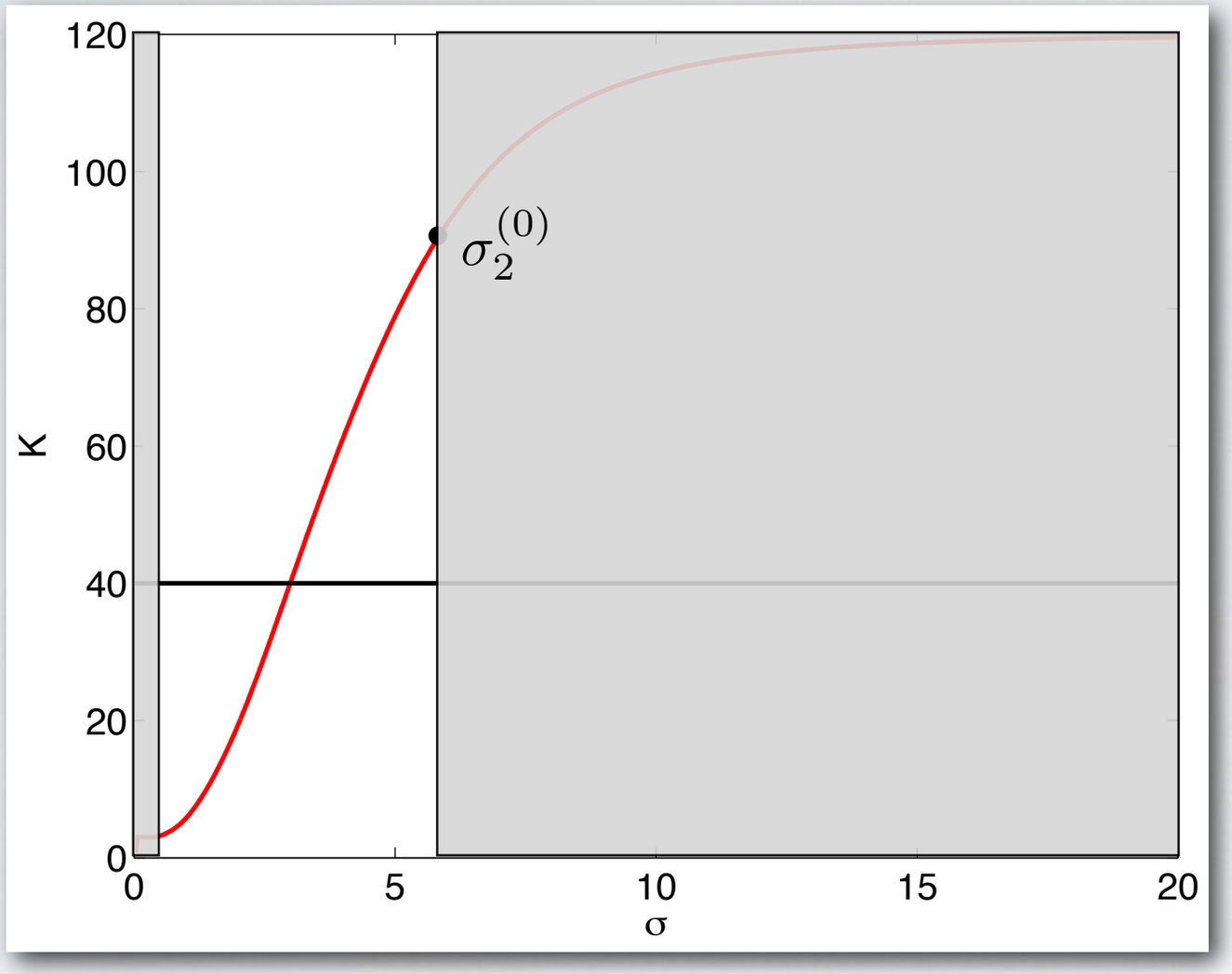


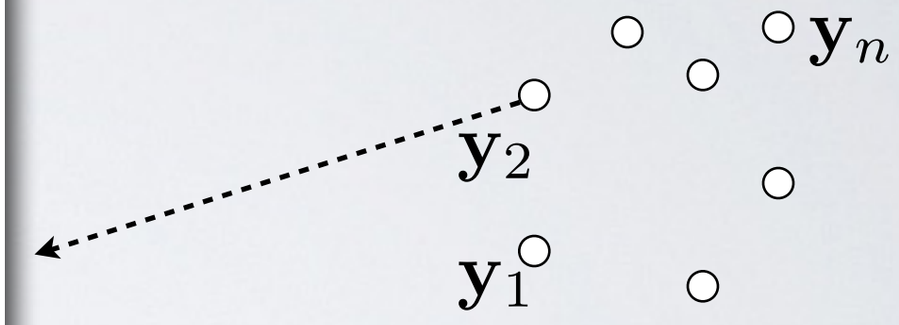
Entropic affinities (computation)



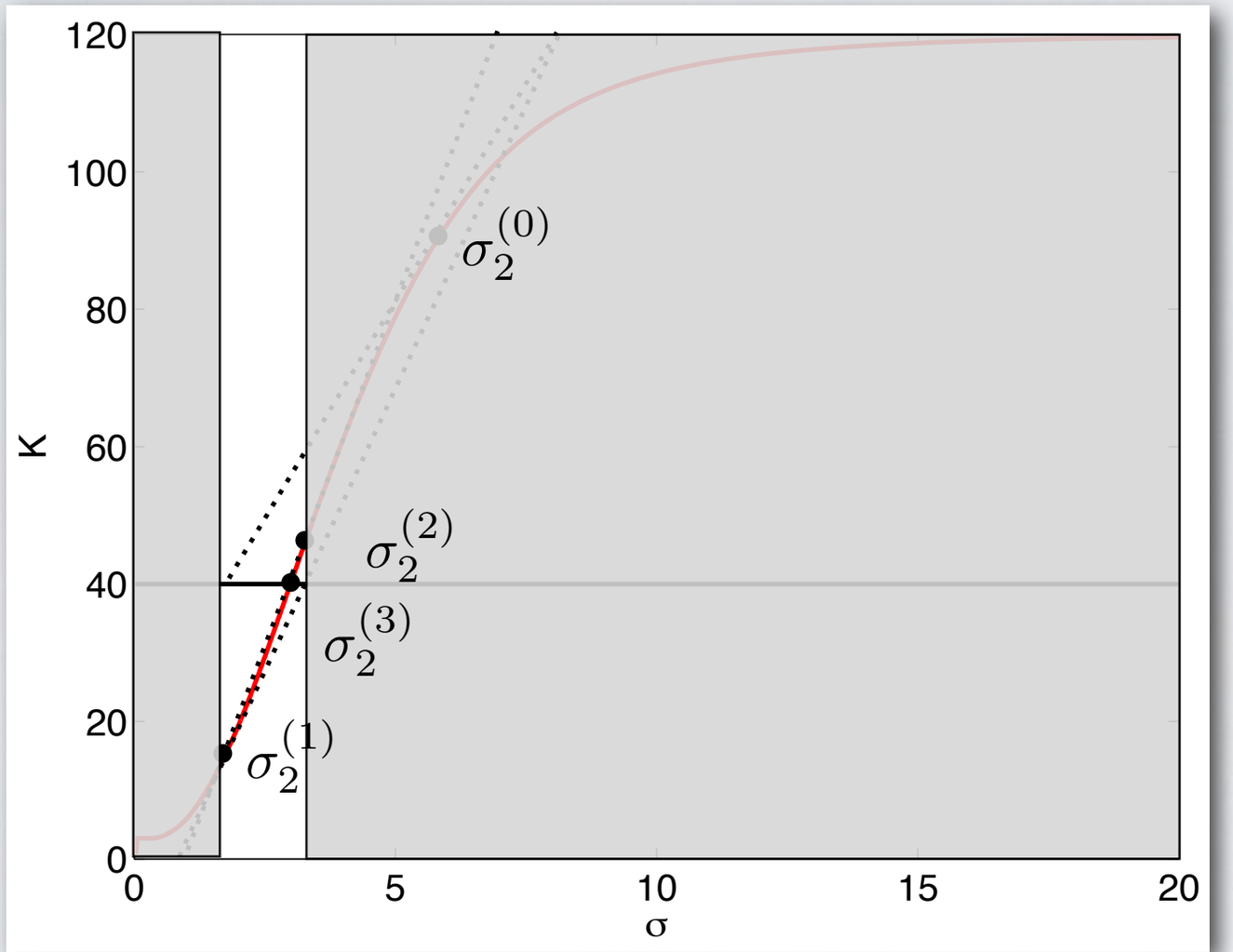


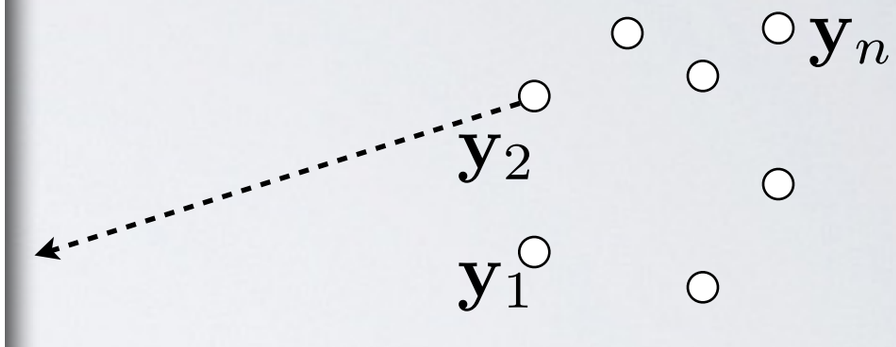
Entropic affinities (computation)



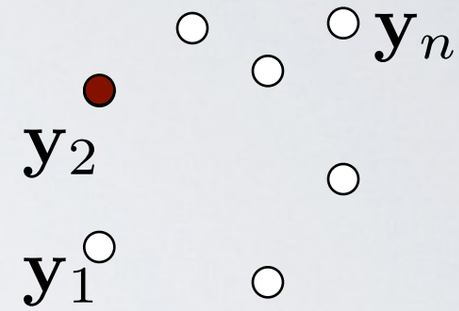


Entropic affinities (computation)

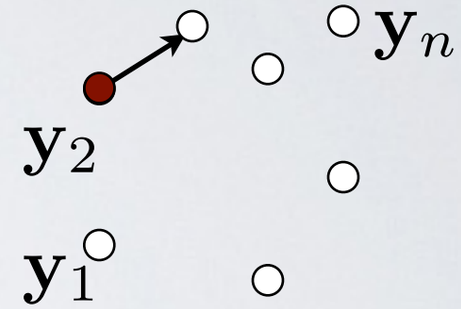




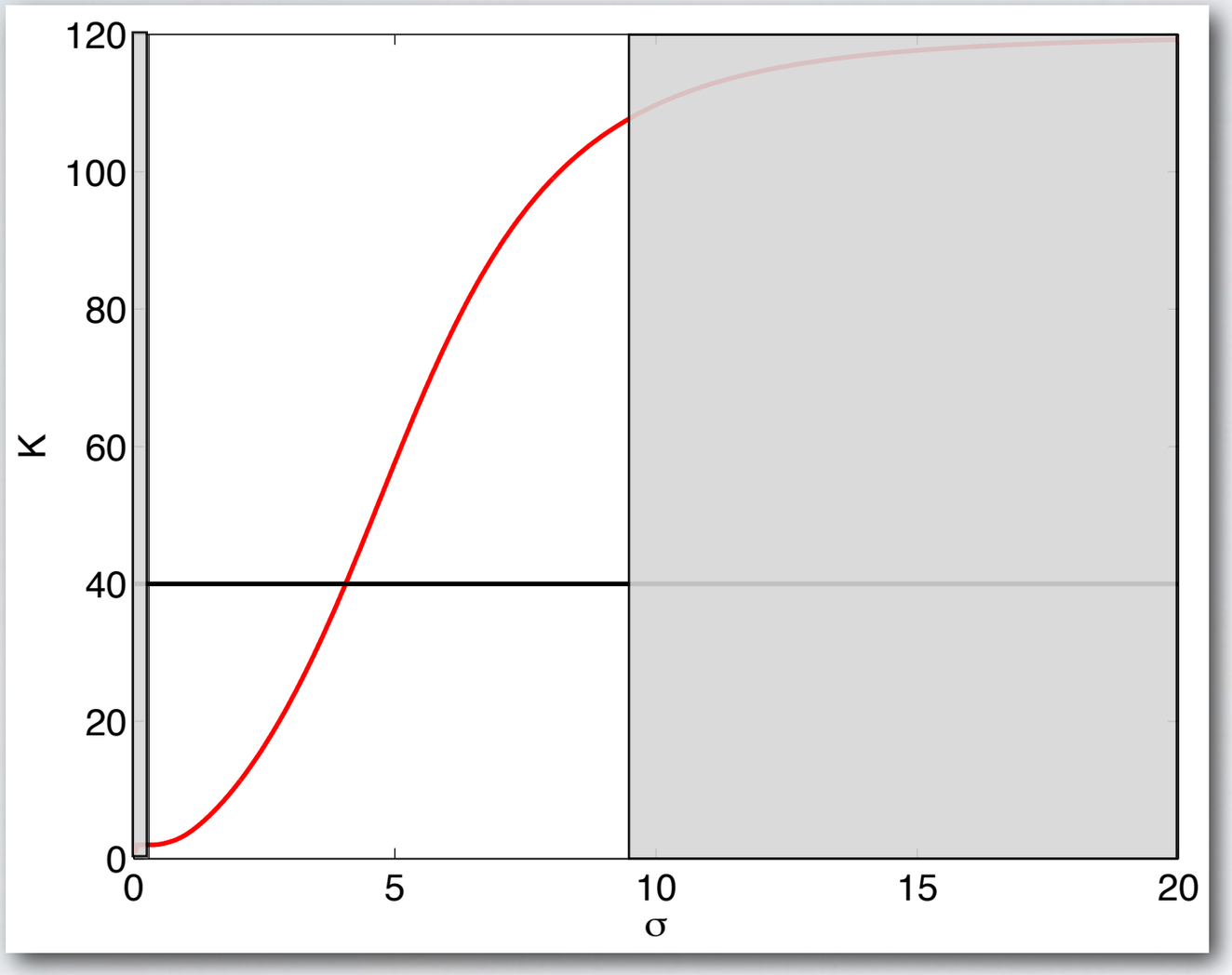
Entropic affinities (computation)

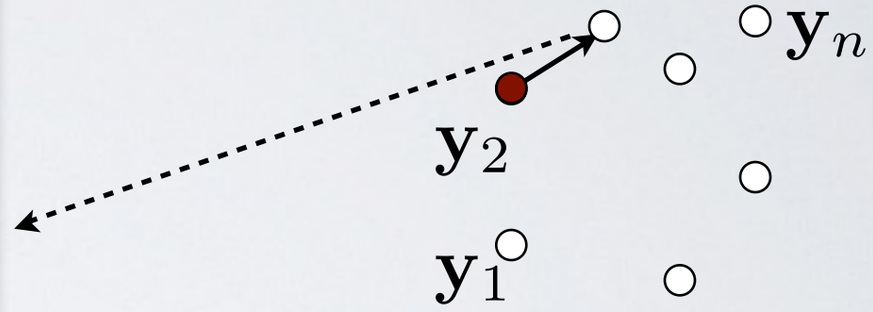


Entropic affinities (computation)

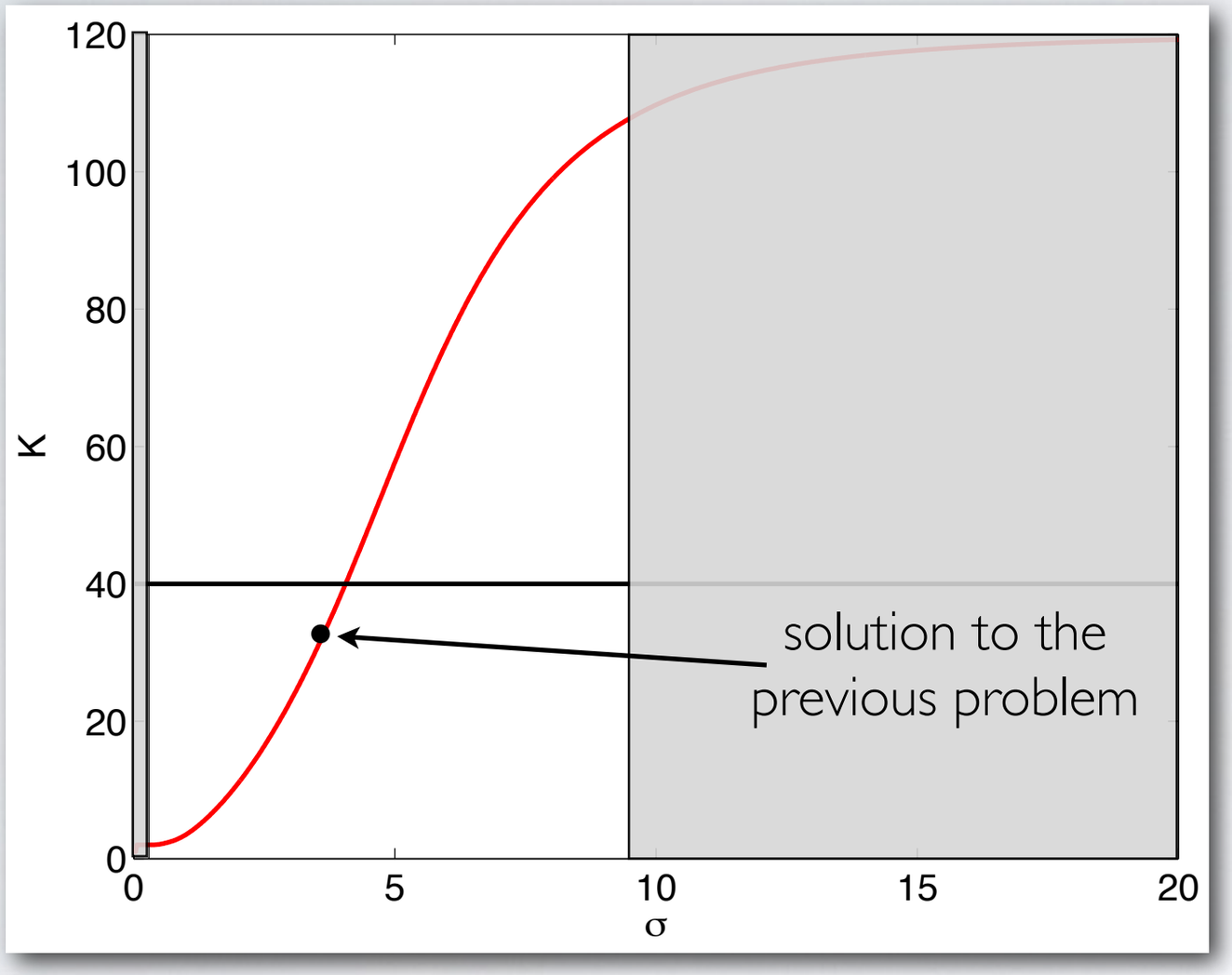


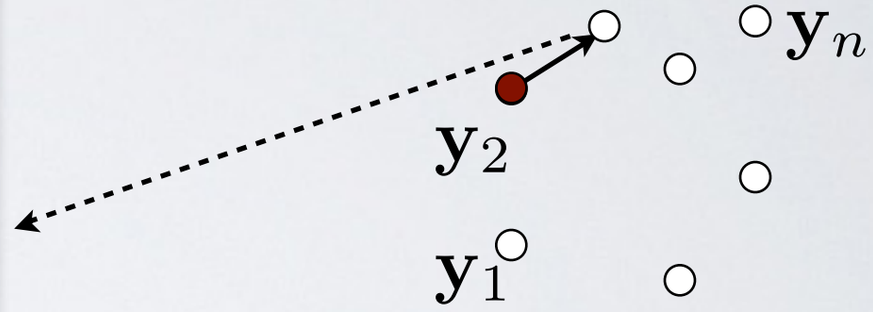
Entropic affinities (computation)



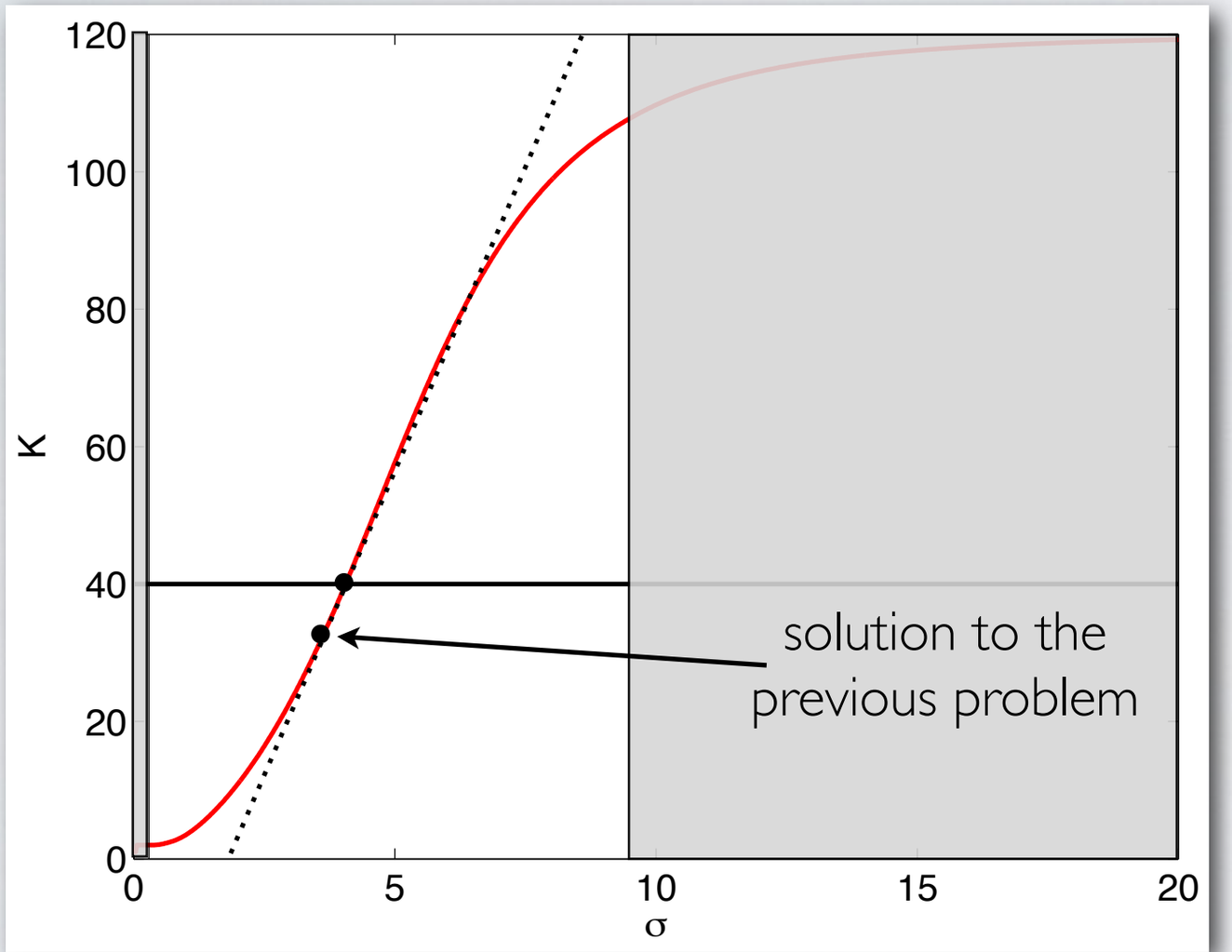


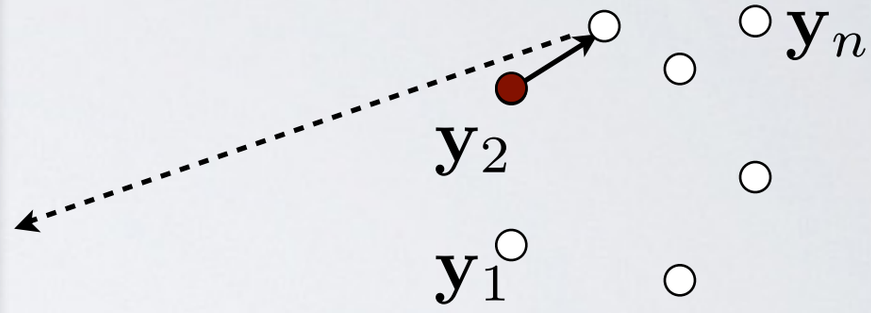
Entropic affinities (computation)



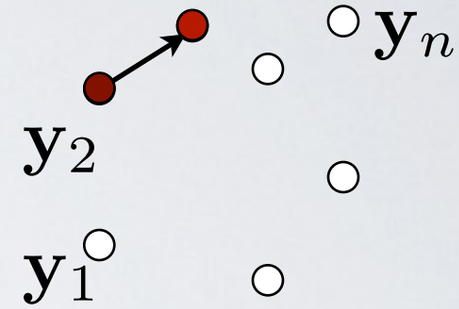


Entropic affinities (computation)

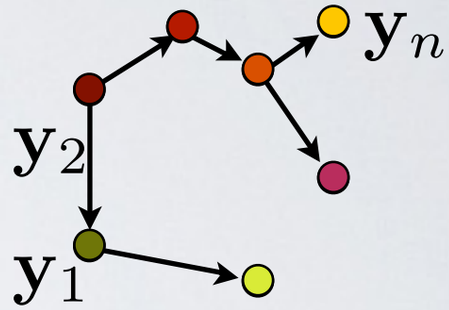




Entropic affinities (computation)



Entropic affinities (computation)



Spectral methods

- Minimize

$$\min_{\mathbf{X}} \operatorname{tr}\left(\mathbf{X}\mathbf{A}\mathbf{X}^T\right) \text{ s.t. } \mathbf{X}\mathbf{B}\mathbf{X}^T = \mathbf{I}$$

  - $\mathbf{A}_{N\times N}$: symmetric psd, contains information about the similarity between pairs of data points.
  - $\mathbf{B}_{N\times N}$: symmetric pd (usually diagonal), set the scale of $\mathbf{X}$.

- Examples:
  - Laplacian eigenmaps, $\mathbf{A}$ graph Laplacian,
  - ISOMAP, $\mathbf{A}$ is given by a matrix of shortest distances,
  - Kernel PCA, MDS, Locally Linear Embedding (LLE), etc.

- Solution is unique and can be found in a closed form from the eigenvectors of $N \times N$ matrix:

  $\mathbf{X} = \mathbf{U}^T\mathbf{B}^{-\frac{1}{2}}$, where $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_d)$ are the d trailing eigenvectors of the $N \times N$ matrix $\mathbf{C} = \mathbf{B}^{-\frac{1}{2}}\mathbf{A}\mathbf{B}^{-\frac{1}{2}}$.

Laplacian Eigenmaps (LE) (Belkin and Niyogi, '03)

- Minimize with respect to $\mathbf{X}$

$$E_{LE}(\mathbf{X}) = \frac{1}{2}\operatorname{tr}\left(\mathbf{X}\mathbf{L}\mathbf{X}^T\right) = \sum_{n,m=1}^{N} w_{nm} \overbrace{\|\mathbf{x}_n - \mathbf{x}_m\|^2}^{\text{distance between } \mathbf{x}_n \text{ and } \mathbf{x}_m}$$

s.t. translation and scale constraints

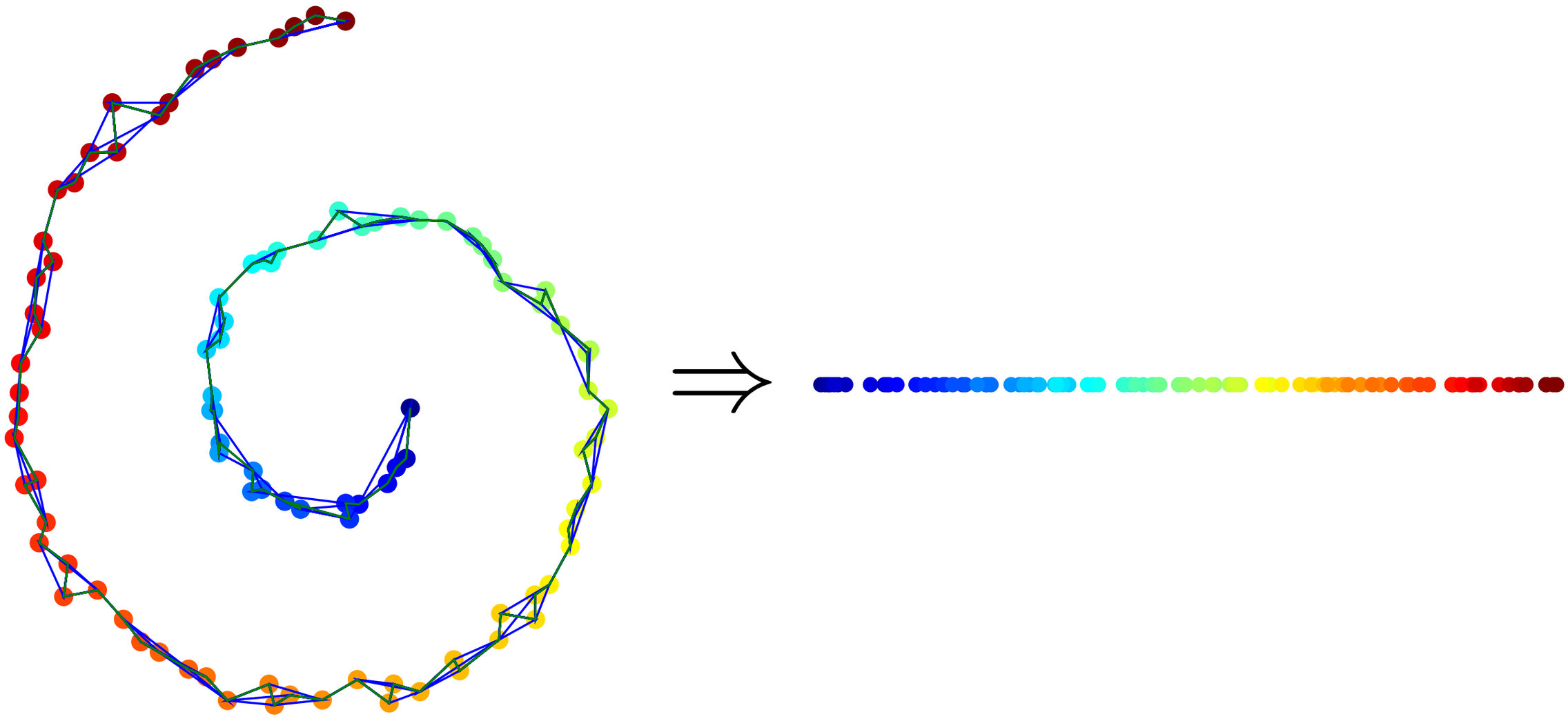
$$= w_{12}\|\mathbf{x}_1 - \mathbf{x}_2\|^2 + w_{13}\|\mathbf{x}_1 - \mathbf{x}_3\|^2 + \cdots + w_{nm}\|\mathbf{x}_n - \mathbf{x}_m\|^2 + \ldots$$

- Intuition:
  - if w_{nm} is large (original points are located nearby to each other) $\Rightarrow$ place $\mathbf{x}_n$ and $\mathbf{x}_m$ nearby.
  - if w_{nm} is small (original points are far away) $\Rightarrow$ there is no direct constraint.

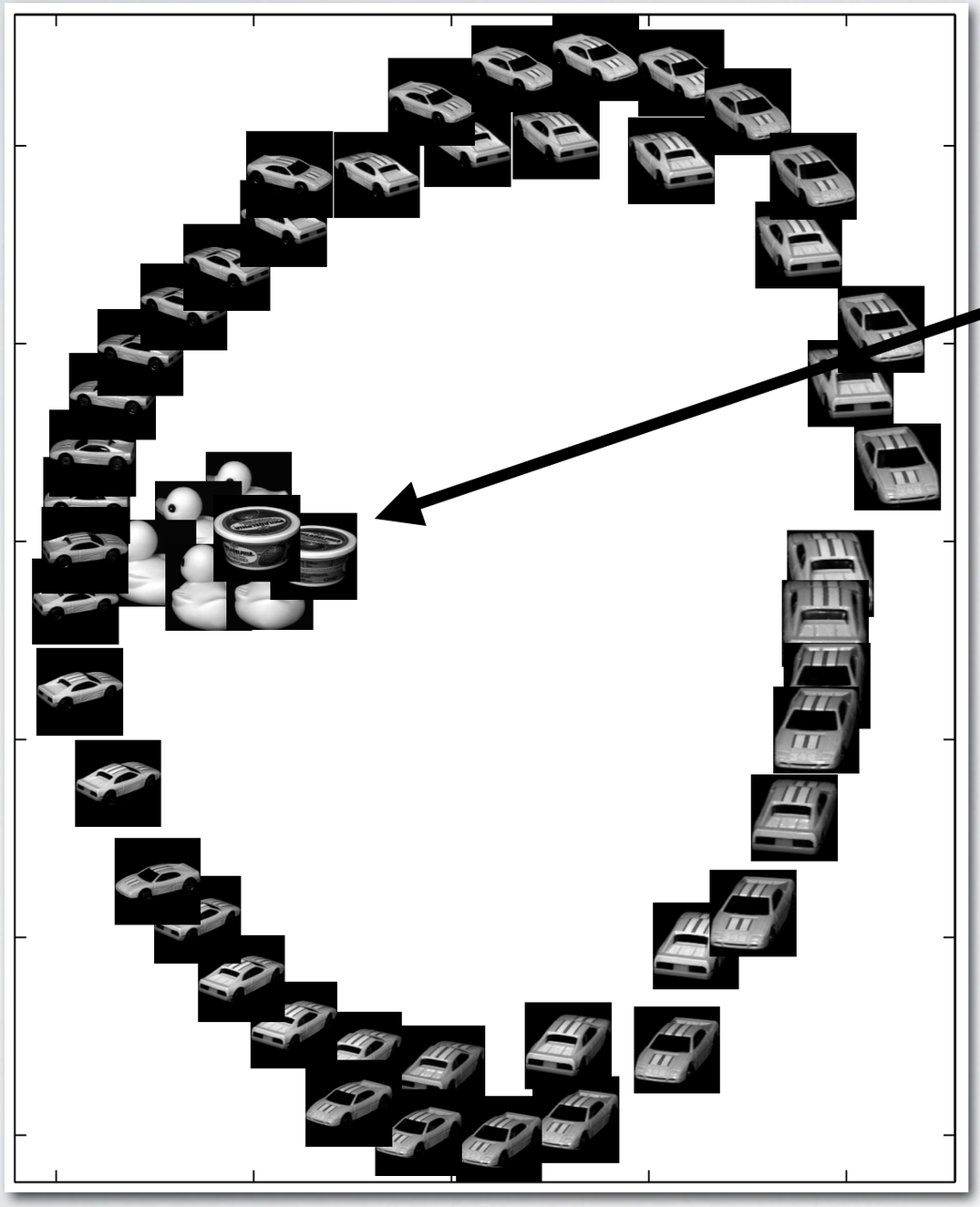
- Spectral method $\Rightarrow$ global minimum is given by trailing eigenvectors of graph Laplacian $\mathbf{L} = \operatorname{diag}\left(\sum_{n=1}^{N} w_{nm}\right) - W$.

Laplacian Eigenmaps (LE) (Belkin and Niyogi, '03)

Laplacian Eigenmaps tries to preserve *local* structure of the data with the *scale* of the embedding being *fixed*.



Laplacian Eigenmaps



Locality is preserved, scale is preserved!

There is nothing that pushes points apart from each other, except for the scale constraint!

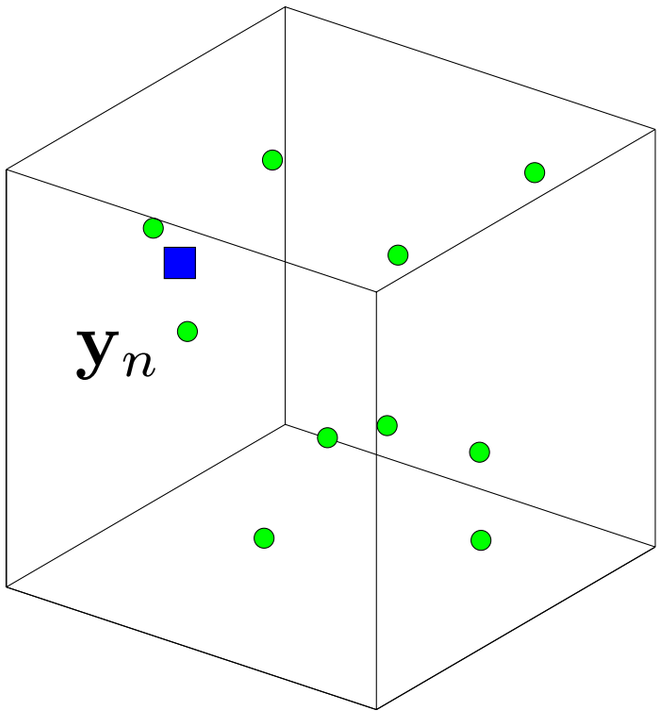
Stochastic neighbor embedding (SNE)

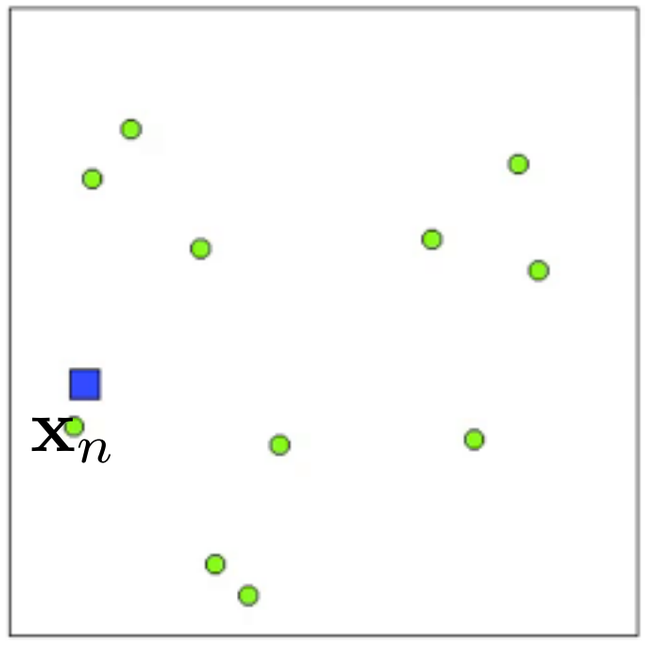
(Hinton and Roweis, '03)

- Define a conditional probability in both spaces that point selects any other point as its neighbor:

$$p_{m|n} = \frac{\exp(-\left\|(\mathbf{y}_n - \mathbf{y}_m)/\sigma^2\right\|)}{\sum_{k \neq n} \exp(-\left\|(\mathbf{y}_n - \mathbf{y}_k)/\sigma^2\right\|)}$$

$$q_{m|n} = \frac{\exp(-\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2)}{\sum_{k \neq n} \exp(-\left\|\mathbf{x}_n - \mathbf{x}_k\right\|^2)}$$





- Minimize the KL-divergence between those probability distributions:

$$E_{SNE}(\mathbf{X}) = \sum_{n=1}^{N} \mathrm{D}\left(P_n \| Q_n\right) = \sum_{n,m=1}^{N} p_{n|m} \log \frac{p_{n|m}}{q_{n|m}}$$

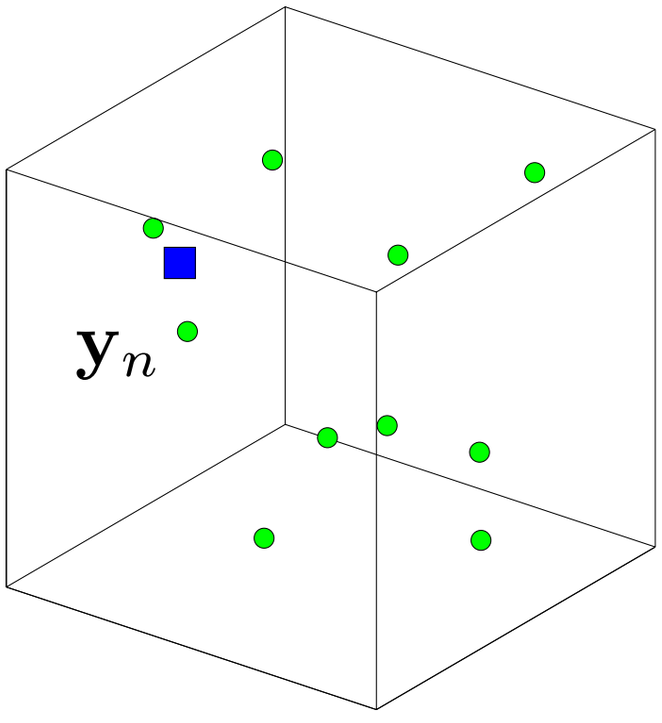
Stochastic neighbor embedding (SNE)

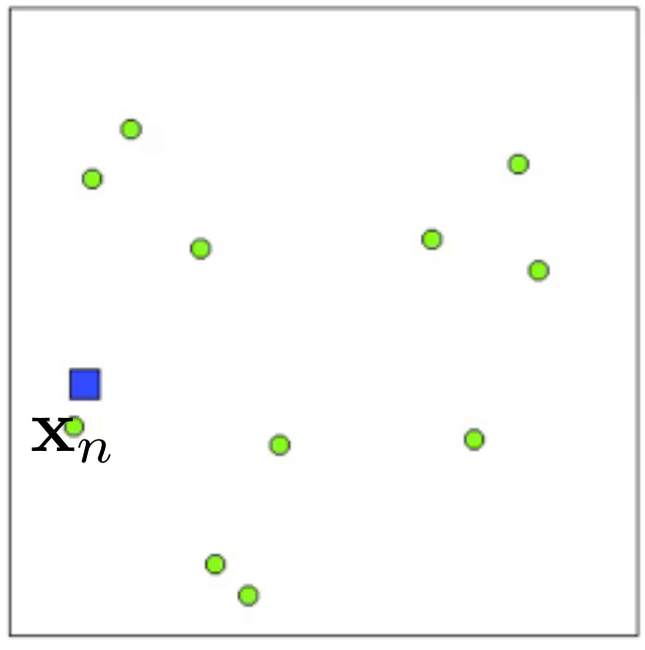
(Hinton and Roweis, '03)

- Define a conditional probability in both spaces that point selects any other point as its neighbor:

$$p_{m|n} = \frac{\exp(-\left\|(\mathbf{y}_n - \mathbf{y}_m)/\sigma^2\right\|)}{\sum_{k \neq n} \exp(-\left\|(\mathbf{y}_n - \mathbf{y}_k)/\sigma^2\right\|)}$$

$$q_{m|n} = \frac{\exp(-\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2)}{\sum_{k \neq n} \exp(-\left\|\mathbf{x}_n - \mathbf{x}_k\right\|^2)}$$





- Minimize the KL-divergence between those probability distributions:

$$E_{SNE}(\mathbf{X}) = \sum_{n=1}^{N} \mathrm{D}\left(P_n \| Q_n\right) = \sum_{n,m=1}^{N} p_{n|m} \log \frac{p_{n|m}}{q_{n|m}}$$

Variations of SNE

- **s-SNE (Cook et al, '07):** Normalizes both pdf over all interactions, not just over distances to a query point

$$p_{nm} = \frac{\exp(-\|(\mathbf{y}_n - \mathbf{y}_m)/\sigma\|^2)}{\sum_{k,l=1}^{N} \exp(-\|(\mathbf{y}_k - \mathbf{y}_l)/\sigma\|^2)} \qquad q_{nm} = \frac{\exp(-\|\mathbf{x}_n - \mathbf{x}_m\|^2)}{\sum_{k,l=1}^{N} \exp(-\|\mathbf{x}_l - \mathbf{x}_k\|^2)}$$

  - symmetric interactions,
  - easier computation,
  - very similar results.

- ***t*-SNE (van der Maaten and Hinton '08):** Defines low-d pdf over Student's t kernel instead of Gaussian:

$$p_{nm} = \frac{\exp(-\|(\mathbf{y}_n - \mathbf{y}_m)/\sigma\|^2)}{\sum_{k,l=1}^{N} \exp(-\|(\mathbf{y}_k - \mathbf{y}_l)/\sigma\|^2)} \qquad q_{nm} = \frac{(1+\|\mathbf{x}_n - \mathbf{x}_m\|^2)^{-1}}{\sum_{k,l=1}^{N} (1+\|\mathbf{x}_l - \mathbf{x}_k\|^2)^{-1}}$$

  - new kernel has heavier tails $\Rightarrow$ better far-field interaction,
  - better for visualization, but worse for exact structure preservation (because we match different kernels).

Relation between LE and SNE

The objective function equals (up to constants):

$$E_{SNE}(\mathbf{X}) = \underbrace{\sum_{n,m=1}^{N} p_{nm} \left\| \mathbf{x}_n - \mathbf{x}_m \right\|^2}_{\text{data-dependent term ①}} + \underbrace{\sum_{n=1}^{N} \log \sum_{m \neq n}^{N} \exp(- \left\| \mathbf{x}_n - \mathbf{x}_m \right\|^2)}_{\text{data-independent term ②}}$$

- Term ① is like Laplacian Eigenmaps.
- Term ② is a "prior" that pushes apart all latent point pairs equally, irrespectively of whether their high-dimensional counterparts are close or far in data space.

Intuition:

- SNE enforces keeping the images of nearby objects nearby (like LE) while pushing all images apart from each other.
- The prior ② is what makes SNE improve significantly over LE.

The elastic embedding (EE) (Carreira-Perpiñán, '10)

- Define two neighborhood graphs:

$$w^+_{nm} = \exp(-\frac{1}{2}\left\|(\mathbf{y}_n - \mathbf{y}_m)/\sigma\right\|^2) \qquad w^-_{nm} = \left\|\mathbf{y}_n - \mathbf{y}_m\right\|^2$$

- Minimize with respect to $\mathbf{X}$

$$E_{EE}(\mathbf{X}, \lambda) = \sum_{n,m=1}^{N} w^+_{nm}\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \lambda \sum_{n,m=1}^{N} w^-_{nm}\exp(\left\|-\mathbf{x}_n - \mathbf{x}_m\right\|^2)$$

Intuition:

- if $\mathbf{y}_n$ and $\mathbf{y}_m$ are similar, first term will pull $\mathbf{x}_n$ and $\mathbf{x}_m$ together,
- if $\mathbf{y}_n$ and $\mathbf{y}_m$ are different, second term will push $\mathbf{x}_n$ and $\mathbf{x}_m$ apart.

Properties:

- the first part is quadratic, the second is more nonlinear and non-convex,
- positive affinity matrix can be sparse (because of a kernel decay),
- negative affinity matrix should be full.

Connections between methods

$$E_{LE}(\mathbf{X}) = \sum_{n,m=1}^{N} w_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2$$ s.t. translation and scale constraints

$$E_{SNE}(\mathbf{X}) = \sum_{n,m=1}^{N} p_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \sum_{n=1}^{N} \log \sum_{m \neq n}^{N} \exp(-\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2)$$

$$E_{s\text{-}SNE}(\mathbf{X}) = \sum_{n,m=1} p_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \log \sum_{n,m=1} \exp(-\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2)$$

$$E_{t\text{-}SNE}(\mathbf{X}) = \sum_{n,m=1}^{N} p_{nm} \log(1 + \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2) + \sum_{n,m=1}^{N} (1 + \left\|-\mathbf{x}_n - \mathbf{x}_m\right\|^2)^{-1}$$

$$E_{EE}(\mathbf{X}) = \sum_{n,m=1}^{N} w^{+}_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \lambda \sum_{n,m=1}^{N} w^{-}_{nm} \exp(\left\|-\mathbf{x}_n - \mathbf{x}_m\right\|^2)$$

Nonlinear Embedding (NLE) methods

General embedding formulation:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

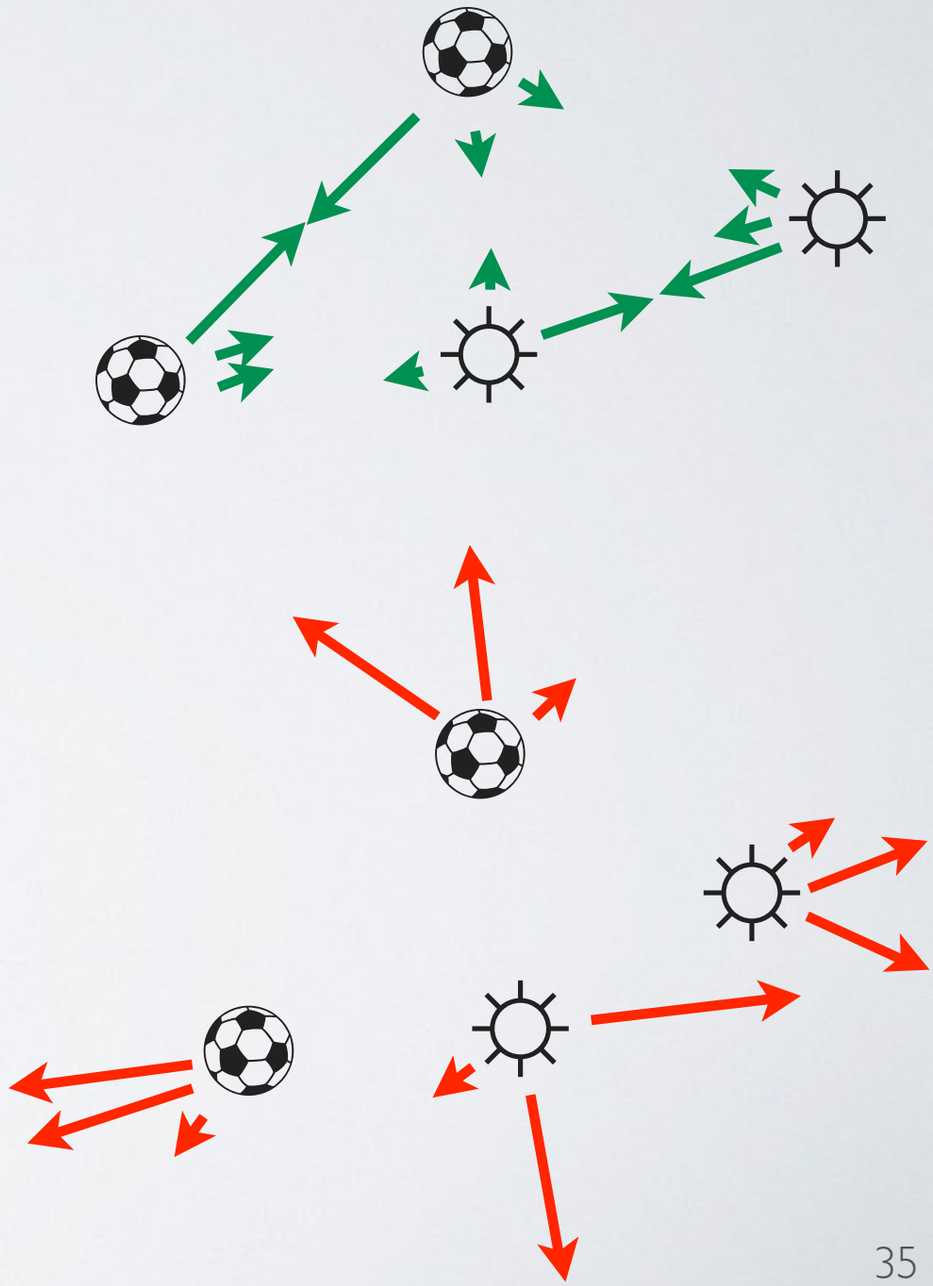
$E^+(\mathbf{X})$ is an *attractive* term:

- often quadratic,
- minimal with coincident points,
- defined usually on the sparse affinity (not all interactions are computed).

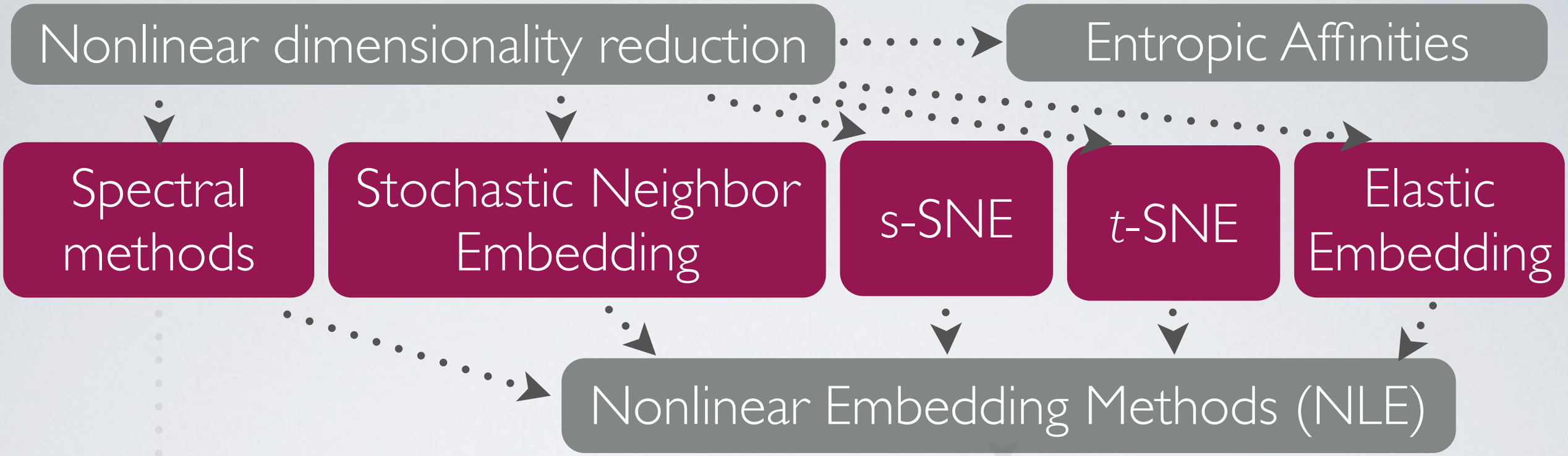
$E^-(\mathbf{X})$ is a *repulsive* term:

- often very nonlinear,
- minimal with points separated infinitely,
- all interactions should be computed.

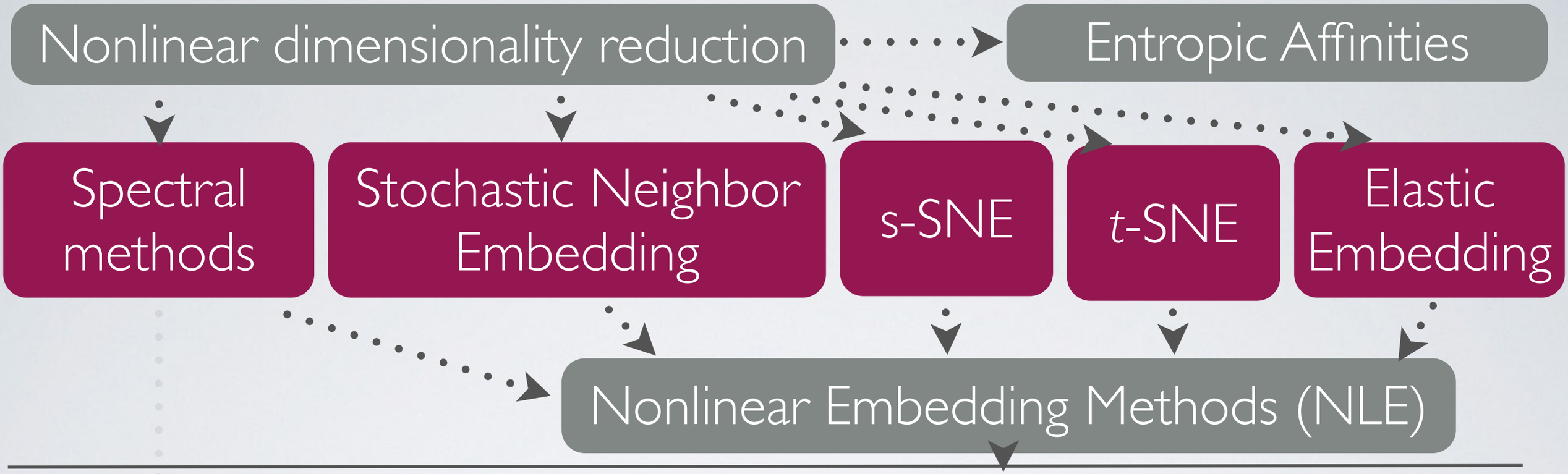
Optimal embedding balances both forces.



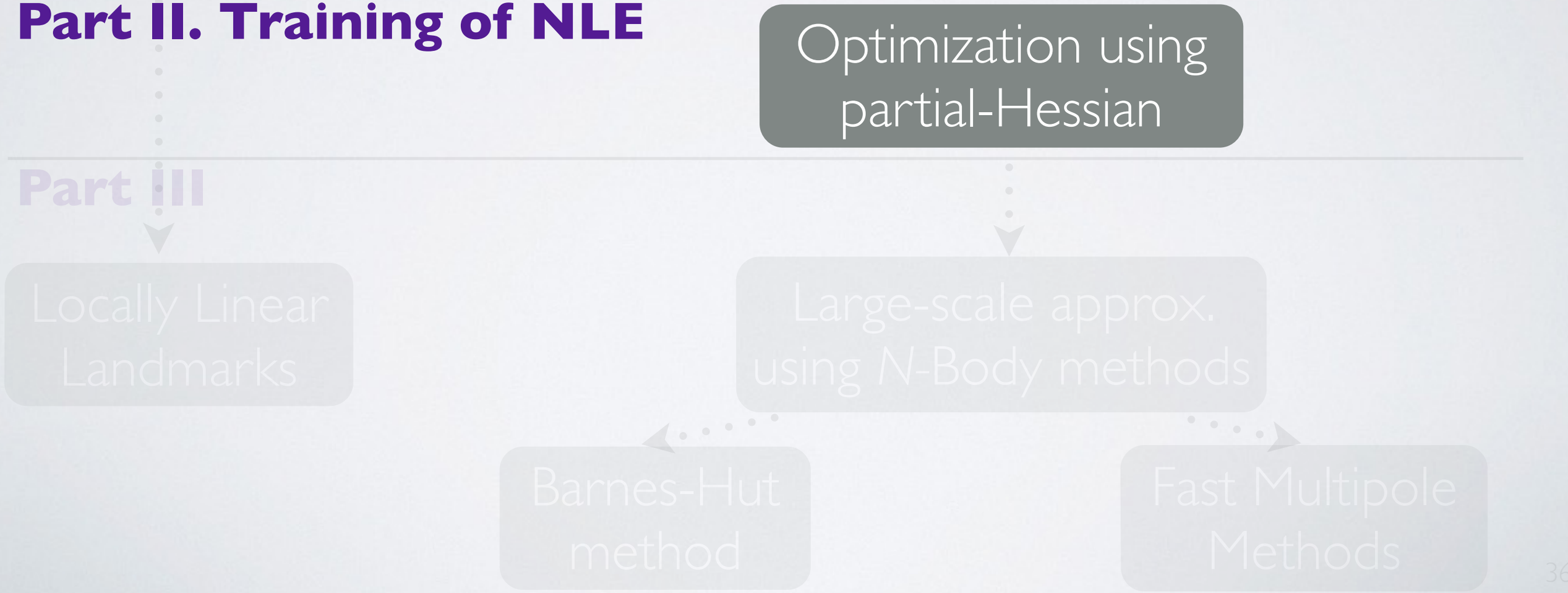
Part I. Nonlinear dimensionality reduction



Part I. Nonlinear dimensionality reduction



Part II. Training of NLE



NLE: Simple optimization algorithm

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Gradient Descent:
  - compute the gradient

$$\mathbf{g} \equiv \nabla \mathbf{E} = 4\mathbf{X}(\mathbf{L}^+ - \lambda \mathbf{L}^-)$$

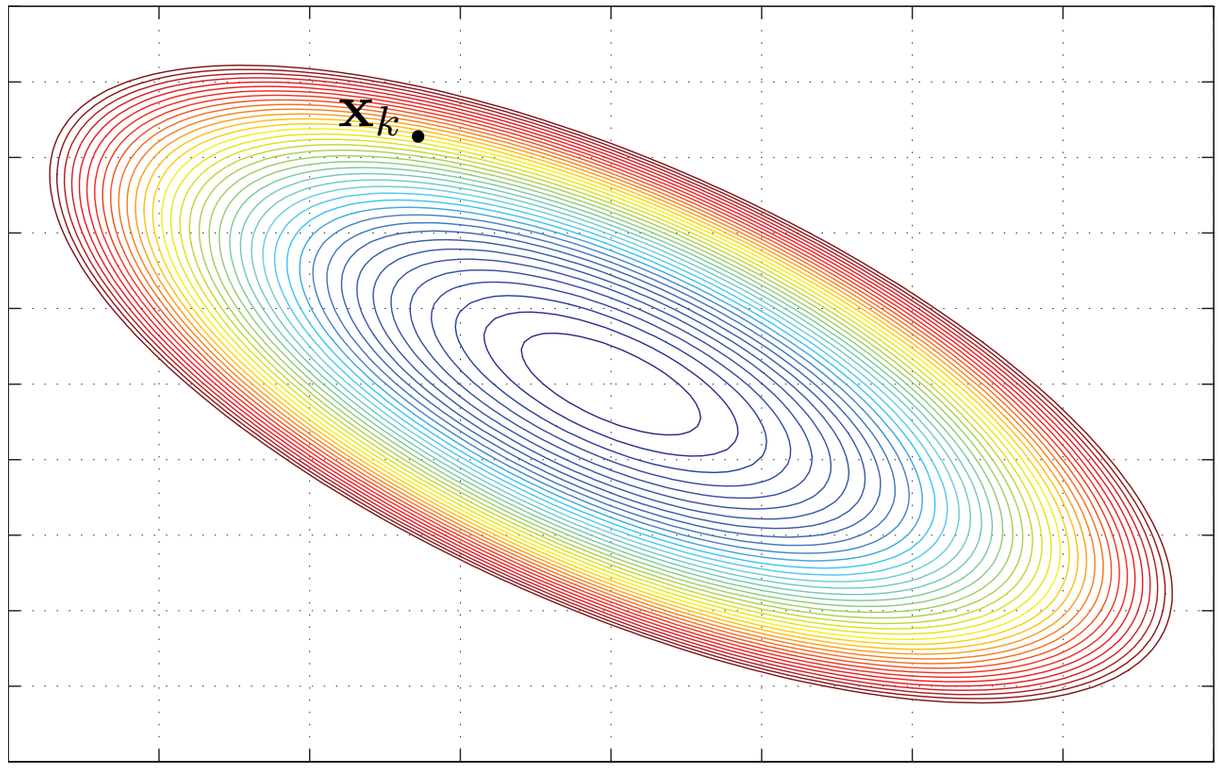
  - compute the direction

$$\mathbf{p}_k = -\mathbf{g}_k$$

  - compute new iteration $\mathbf{x}_{k+1}$ with a line search:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta \mathbf{p}_k$$

  - repeat till convergence.



- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc.

Can we do better?

NLE: Simple optimization algorithm

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$

- Gradient Descent:
  - compute the gradient

$$\mathbf{g} \equiv \nabla \mathbf{E} = 4\mathbf{X}(\mathbf{L}^{+} - \lambda \mathbf{L}^{-})$$

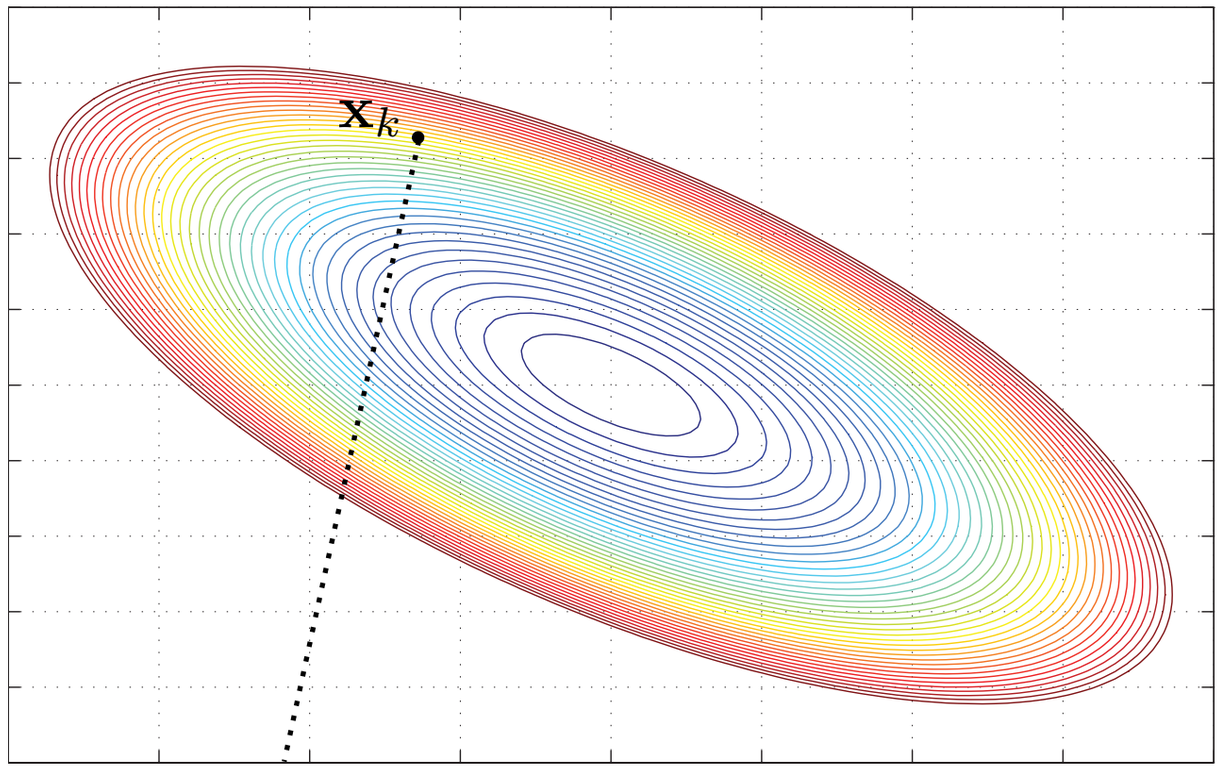
  - compute the direction

$$\mathbf{p}_k = -\mathbf{g}_k$$

  - compute new iteration $\mathbf{x}_{k+1}$ with a line search:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta \mathbf{p}_k$$

  - repeat till convergence.



- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc.

Can we do better?

NLE: Simple optimization algorithm

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$

- Gradient Descent:
  - compute the gradient

$$\mathbf{g} \equiv \nabla \mathbf{E} = 4\mathbf{X}(\mathbf{L}^{+} - \lambda \mathbf{L}^{-})$$

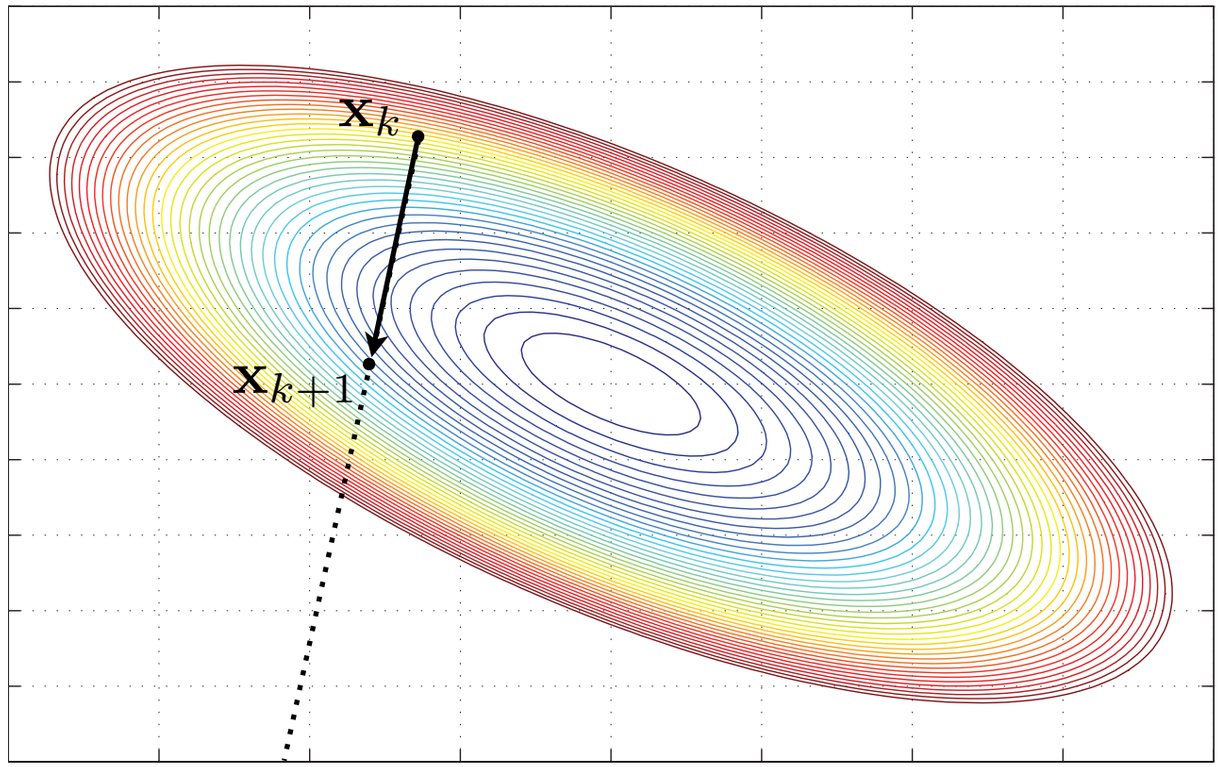
  - compute the direction

$$\mathbf{p}_k = -\mathbf{g}_k$$

  - compute new iteration $\mathbf{x}_{k+1}$ with a line search:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta \mathbf{p}_k$$

  - repeat till convergence.



- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc.

Can we do better?

NLE: Simple optimization algorithm

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Gradient Descent:
  - compute the gradient

$$\mathbf{g} \equiv \nabla \mathbf{E} = 4\mathbf{X}(\mathbf{L}^+ - \lambda \mathbf{L}^-)$$

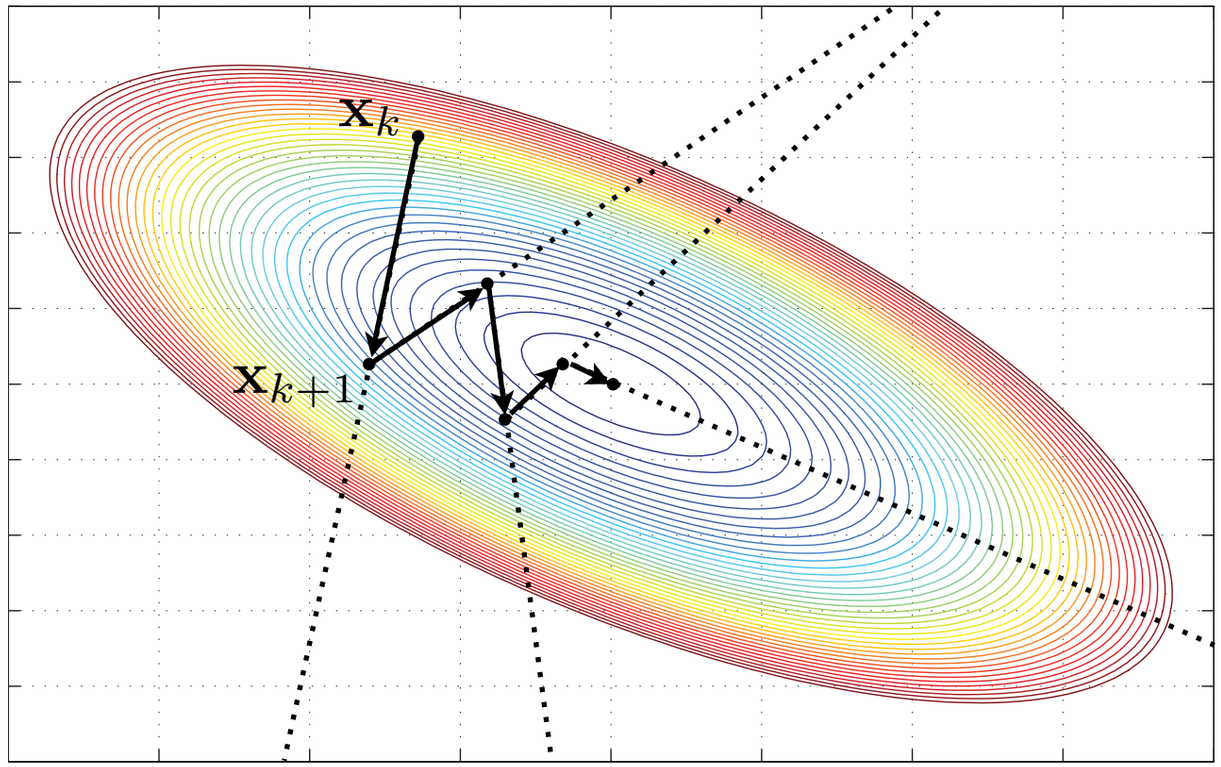
  - compute the direction

$$\mathbf{p}_k = -\mathbf{g}_k$$

  - compute new iteration $\mathbf{x}_{k+1}$ with a line search:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta \mathbf{p}_k$$

  - repeat till convergence.



- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc.

Can we do better?

NLE: Simple optimization algorithm

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Gradient Descent:
  - compute the gradient

$$\mathbf{g} \equiv \nabla \mathbf{E} = 4\mathbf{X}(\mathbf{L}^+ - \lambda \mathbf{L}^-)$$

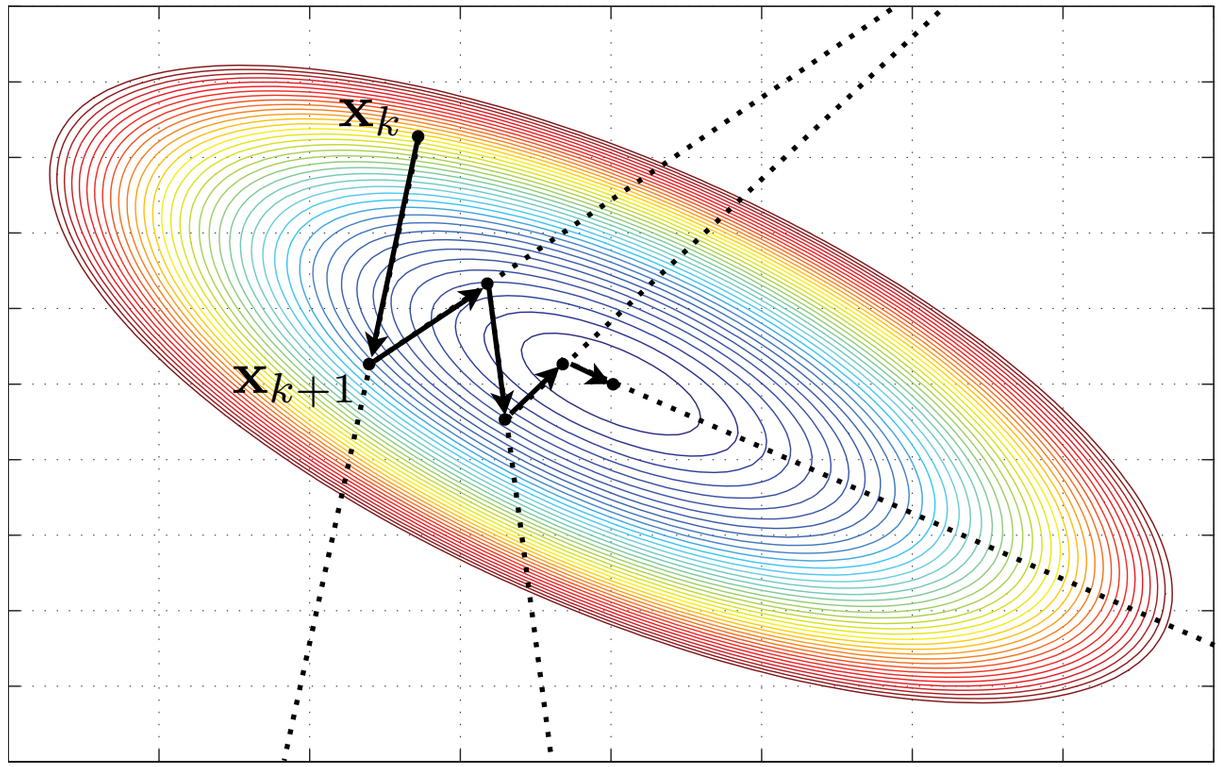
  - compute the direction

$$\mathbf{p}_k = -\mathbf{g}_k$$

  - compute new iteration $\mathbf{x}_{k+1}$ with a line search:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta \mathbf{p}_k$$

  - repeat till convergence.



- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc.

Can we do better?

NLE: Simple optimization algorithm

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Gradient Descent:
  - compute the gradient

$$\mathbf{g} \equiv \nabla \mathbf{E} = 4\mathbf{X}(\mathbf{L}^+ - \lambda \mathbf{L}^-)$$

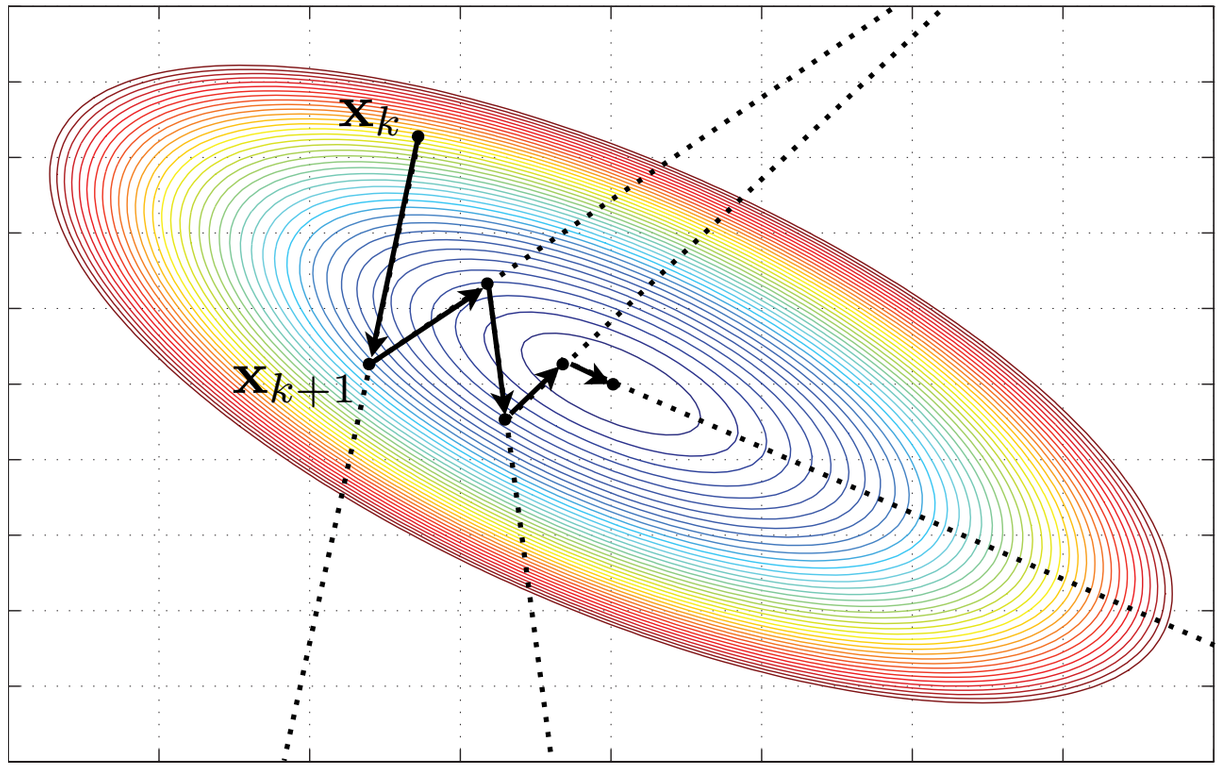
  - compute the direction

$$\mathbf{p}_k = -\mathbf{g}_k$$

  - compute new iteration $\mathbf{x}_{k+1}$ with a line search:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta \mathbf{p}_k$$

  - repeat till convergence.



- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc.

Can we do better?

Including second-order information

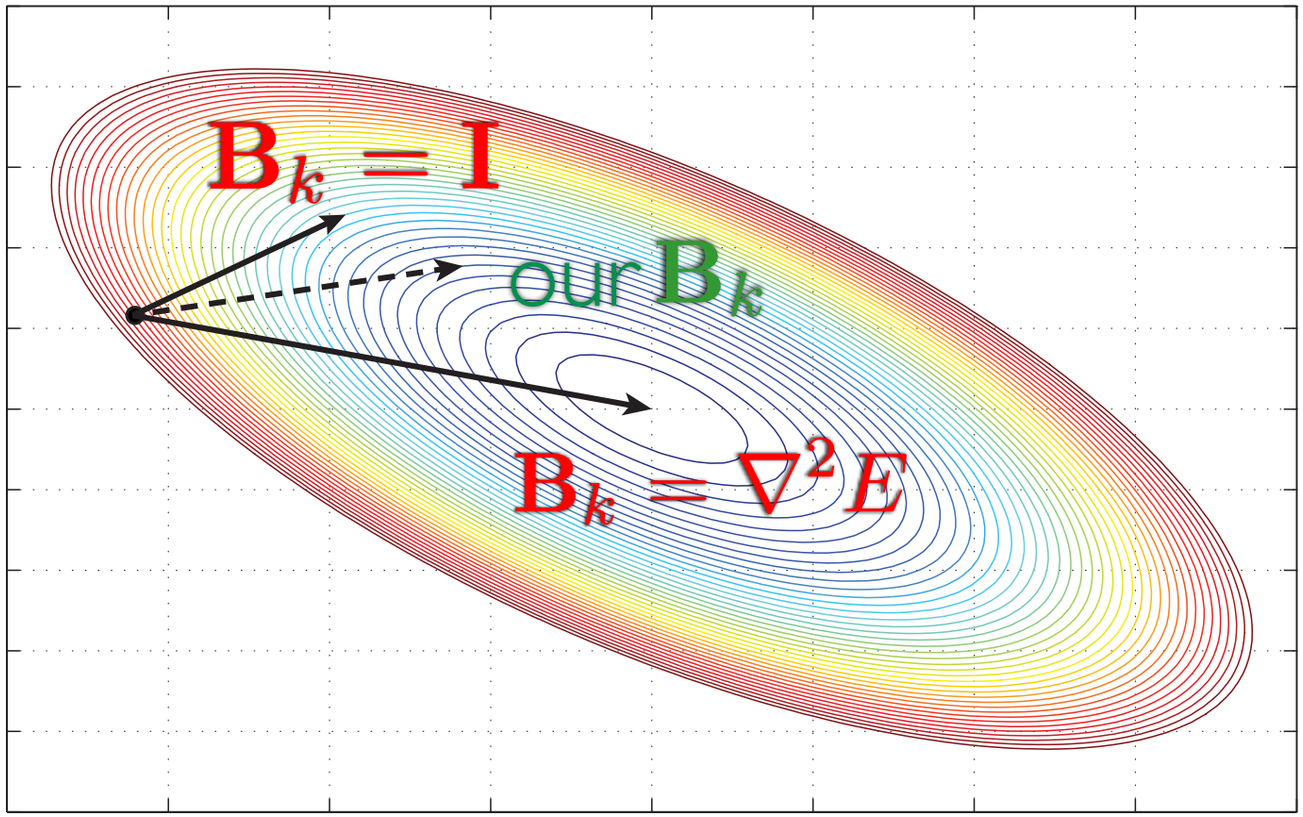
Consider the following method. For every iteration k:

- choose any positive definite $\mathbf{B}_k$,
- solve a linear system $\mathbf{B}_k \mathbf{p}_k = -\mathbf{g}_k$ for a search direction $\mathbf{p}_k$,
- use line search to find a step size η for the next iteration $\mathbf{x}_{k+1} = \mathbf{x}_k + \eta \mathbf{p}_k$ (e.g. with backtracking line search).

Convergence is guaranteed!

How to choose good $\mathbf{B}_k$?

$\mathbf{B}_k = \mathbf{I}$ (gradient descent) —more Hessian information / faster convergence rate→ $\mathbf{B}_k = \nabla^2 E$ (Newton's method)



We want $\mathbf{B}_k$:

- positive definite (pd),
- fast to compute and solve the linear system $\mathbf{B}_k \mathbf{p}_k = -\mathbf{g}_k$.
- contain as much Hessian information as possible,

The Hessian of Nonlinear Embedding

The Hessian is $Nd \times Nd$ matrix and given by:

$$\begin{aligned}\nabla^2 E = & \ 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d} \\ & - \ 4\lambda\mathbf{L}^- \otimes \mathbf{I}_{d\times d} \\ & + \ 8\mathbf{L}^{xx} \\ & - \ 16\lambda \mathrm{vec}(\mathbf{X}\mathbf{L}^q)\mathrm{vec}(\mathbf{X}\mathbf{L}^q)^T\end{aligned}$$

where $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^{xx}, \mathbf{L}^q$ are graph Laplacians:

- $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^q$: constant for Gaussian kernel $\Rightarrow$ psd.
- $\mathbf{L}^{xx}$: depends on the embedding $\mathbf{X}$. Some parts are psd.

The Hessian of Nonlinear Embedding

The Hessian is $Nd \times Nd$ matrix and given by:

$$\begin{aligned}\nabla^2 E = & \ 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d} \\ & - \ 4\lambda\mathbf{L}^- \otimes \mathbf{I}_{d\times d} \\ & + \ 8\mathbf{L}^{xx} \\ & - \ 16\lambda \mathrm{vec}(\mathbf{X}\mathbf{L}^q)\mathrm{vec}(\mathbf{X}\mathbf{L}^q)^T\end{aligned}$$

where $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^{xx}, \mathbf{L}^q$ are graph Laplacians:

- $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^q$: constant for Gaussian kernel $\Rightarrow$ psd.
- $\mathbf{L}^{xx}$: depends on the embedding $\mathbf{X}$. Some parts are psd.

The Hessian of Nonlinear Embedding

The Hessian is $Nd \times Nd$ matrix and given by:

$$\begin{aligned}\nabla^2 E = & \ 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d} \\ & - \ 4\lambda \mathbf{L}^- \otimes \mathbf{I}_{d\times d} \\ & + \ 8\mathbf{L}^{xx} \\ & - \ 16\lambda \mathrm{vec}(\mathbf{X}\mathbf{L}^q)\mathrm{vec}(\mathbf{X}\mathbf{L}^q)^T\end{aligned}$$

where $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^{xx}, \mathbf{L}^q$ are graph Laplacians:

- $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^q$: constant for Gaussian kernel $\Rightarrow$ psd.
- $\mathbf{L}^{xx}$: depends on the embedding $\mathbf{X}$. Some parts are psd.

The Hessian of Nonlinear Embedding

The Hessian is $Nd \times Nd$ matrix and given by:

$$\begin{aligned}\nabla^2 E = & \ 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d} \\ & - \ 4\lambda \mathbf{L}^- \otimes \mathbf{I}_{d\times d} \\ & + \ 8\mathbf{L}^{xx} \\ & - \ 16\lambda \mathrm{vec}(\mathbf{X}\mathbf{L}^q)\mathrm{vec}(\mathbf{X}\mathbf{L}^q)^T\end{aligned}$$

where $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^{xx}, \mathbf{L}^q$ are graph Laplacians:

- $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^q$: constant for Gaussian kernel $\Rightarrow$ psd.
- $\mathbf{L}^{xx}$: depends on the embedding $\mathbf{X}$. Some parts are psd.

The Hessian of Nonlinear Embedding

The Hessian is $Nd \times Nd$ matrix and given by:

$$\begin{aligned}\nabla^2 E = & \ 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d} \\ & - \ 4\lambda \mathbf{L}^- \otimes \mathbf{I}_{d\times d} \\ & + \ 8\mathbf{L}^{xx} \\ & - \ 16\lambda \mathrm{vec}(\mathbf{X}\mathbf{L}^q)\mathrm{vec}(\mathbf{X}\mathbf{L}^q)^T\end{aligned}$$

where $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^{xx}, \mathbf{L}^q$ are graph Laplacians:

- $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^q$: constant for Gaussian kernel $\Rightarrow$ psd.
- $\mathbf{L}^{xx}$: depends on the embedding $\mathbf{X}$. Some parts are psd.

The Hessian of Nonlinear Embedding

The Hessian is $Nd \times Nd$ matrix and given by:

$$\begin{aligned}\nabla^2 E = & \ 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d} \\ & - \ 4\lambda \mathbf{L}^- \otimes \mathbf{I}_{d\times d} \\ & + \ 8\mathbf{L}^{xx} \\ & - \ 16\lambda \mathrm{vec}(\mathbf{X}\mathbf{L}^q)\mathrm{vec}(\mathbf{X}\mathbf{L}^q)^T\end{aligned}$$

where $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^{xx}, \mathbf{L}^q$ are graph Laplacians:

- $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^q$: constant for Gaussian kernel $\Rightarrow$ psd.
- $\mathbf{L}^{xx}$: depends on the embedding $\mathbf{X}$. Some parts are psd.

The Hessian of Nonlinear Embedding

The Hessian is $Nd \times Nd$ matrix and given by:

$$\begin{aligned}\nabla^2 E = & \ 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d} \\ & - \ 4\lambda \mathbf{L}^- \otimes \mathbf{I}_{d\times d} \\ & + \ 8\mathbf{L}^{xx} \\ & - \ 16\lambda \mathrm{vec}(\mathbf{X}\mathbf{L}^q)\mathrm{vec}(\mathbf{X}\mathbf{L}^q)^T\end{aligned}$$

where $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^{xx}, \mathbf{L}^q$ are graph Laplacians:

- $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^q$: constant for Gaussian kernel $\Rightarrow$ psd.
- $\mathbf{L}^{xx}$: depends on the embedding $\mathbf{X}$. Some parts are psd.

The Hessian of Nonlinear Embedding

The Hessian is $Nd \times Nd$ matrix and given by:

$$\begin{aligned}\nabla^2 E = & \ 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d} \\ & - \ 4\lambda \mathbf{L}^- \otimes \mathbf{I}_{d\times d} \\ & + \ 8\mathbf{L}^{xx} \\ & - \ 16\lambda \mathrm{vec}(\mathbf{X}\mathbf{L}^q)\mathrm{vec}(\mathbf{X}\mathbf{L}^q)^T\end{aligned}$$

where $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^{xx}, \mathbf{L}^q$ are graph Laplacians:

- $\mathbf{L}^+, \mathbf{L}^-, \mathbf{L}^q$: constant for Gaussian kernel $\Rightarrow$ psd.
- $\mathbf{L}^{xx}$: depends on the embedding $\mathbf{X}$. Some parts are psd.

The spectral direction (Vladymyrov and Carreira-Perpiñán, '12)

$\mathbf{B}_k = 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d}$ is a convenient Hessian approximation.

We wanted $\mathbf{B}_k$:

- positive definite (pd),
- fast to compute and solve the linear system $\mathbf{B}_k\mathbf{p}_k = -\mathbf{g}_k$
- contain as much Hessian information as possible,

The spectral direction (Vladymyrov and Carreira-Perpiñán, '12)

$\mathbf{B}_k = 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d}$ is a convenient Hessian approximation.

We wanted $\mathbf{B}_k$:

✓positive definite (pd),

- fast to compute and solve the linear system $\mathbf{B}_k\mathbf{p}_k = -\mathbf{g}_k$
- contain as much Hessian information as possible,

The spectral direction (Vladymyrov and Carreira-Perpiñán, '12)

$\mathbf{B}_k = 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d}$ is a convenient Hessian approximation.

We wanted $\mathbf{B}_k$:

✓positive definite (pd),

- fast to compute and solve the linear system $\mathbf{B}_k\mathbf{p}_k = -\mathbf{g}_k$
  - block-diagonal and has d blocks of $N \times N$ graph Laplacian $4\mathbf{L}^+$
  - constant for Gaussian kernel $\mathbf{B} = \mathbf{B}_k$. For other kernels we can fix it as some intermediate $\mathbf{X}$,
  - linear system can be solved efficiently using Cholesky factorization of $\mathbf{B}$,
  - (further) sparsify the weights $\mathbf{W}^+$with a κ-NN graph.
- contain as much Hessian information as possible,

The spectral direction (Vladymyrov and Carreira-Perpiñán, '12)

$\mathbf{B}_k = 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d}$ is a convenient Hessian approximation.

We wanted $\mathbf{B}_k$:

✓positive definite (pd),

✓fast to compute and solve the linear system $\mathbf{B}_k\mathbf{p}_k = -\mathbf{g}_k$

- block-diagonal and has d blocks of $N \times N$ graph Laplacian $4\mathbf{L}^+$
- constant for Gaussian kernel $\mathbf{B} = \mathbf{B}_k$. For other kernels we can fix it as some intermediate $\mathbf{X}$,
- linear system can be solved efficiently using Cholesky factorization of $\mathbf{B}$,
- (further) sparsify the weights $\mathbf{W}^+$with a κ-NN graph.

• contain as much Hessian information as possible,

The spectral direction (Vladymyrov and Carreira-Perpiñán, '12)

$\mathbf{B}_k = 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d}$ is a convenient Hessian approximation.

We wanted $\mathbf{B}_k$:

✓positive definite (pd),

✓fast to compute and solve the linear system $\mathbf{B}_k\mathbf{p}_k = -\mathbf{g}_k$

- block-diagonal and has d blocks of $N \times N$ graph Laplacian $4\mathbf{L}^+$
- constant for Gaussian kernel $\mathbf{B} = \mathbf{B}_k$. For other kernels we can fix it as some intermediate $\mathbf{X}$,
- linear system can be solved efficiently using Cholesky factorization of $\mathbf{B}$,
- (further) sparsify the weights $\mathbf{W}^+$with a κ-NN graph.

- contain as much Hessian information as possible,
  - equal to the Hessian of the spectral methods: $\mathbf{B}_k = \nabla^2 E^+(\mathbf{X})$,
  - "bends" the gradient of the nonlinear E using the curvature of the spectral E^+.

The spectral direction (Vladymyrov and Carreira-Perpiñán, '12)

$\mathbf{B}_k = 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d}$ is a convenient Hessian approximation.

We wanted $\mathbf{B}_k$:

✓positive definite (pd),

✓fast to compute and solve the linear system $\mathbf{B}_k\mathbf{p}_k = -\mathbf{g}_k$

- block-diagonal and has d blocks of $N \times N$ graph Laplacian $4\mathbf{L}^+$
- constant for Gaussian kernel $\mathbf{B} = \mathbf{B}_k$. For other kernels we can fix it as some intermediate $\mathbf{X}$,
- linear system can be solved efficiently using Cholesky factorization of $\mathbf{B}$,
- (further) sparsify the weights $\mathbf{W}^+$with a κ-NN graph.

✓contain as much Hessian information as possible,

- equal to the Hessian of the spectral methods: $\mathbf{B}_k = \nabla^2 E^+(\mathbf{X})$,
- "bends" the gradient of the nonlinear E using the curvature of the spectral E^+.

The spectral direction (computation)

	Cost per iteration
Objective function	$\mathcal{O}(N^2 d)$
Gradient	$\mathcal{O}(N^2 d)$
Spectral direction	$\mathcal{O}(N \kappa d)$

- Runtime is faster and convergence is still guaranteed.
- The strategy adds almost no overhead when compared to the objective function and the gradient computation.
- Applicable to any nonlinear embedding method (s-SNE, *t*-SNE, EE, …).

The spectral direction (experiments)

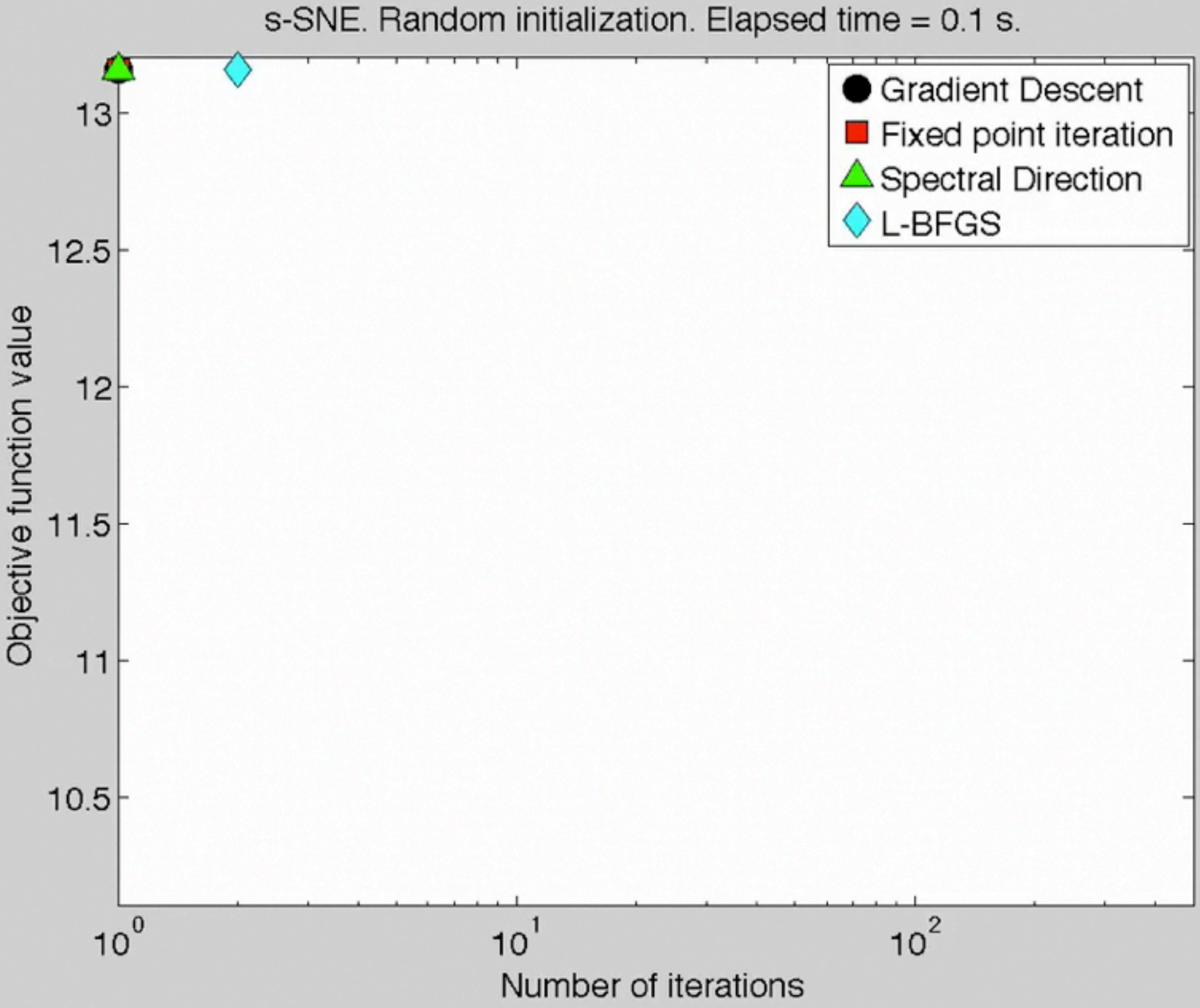
Optimization methods compared:

- Gradient descent
- Fixed-point iterations
- The spectral direction
- L-BFGS

$$\mathbf{B}_k = \mathbf{I}$$
$$\mathbf{B}_k = 4\mathbf{D}^+ \otimes \mathbf{I}_{d\times d}$$
$$\mathbf{B}_k = 4\mathbf{L}^+ \otimes \mathbf{I}_{d\times d}$$

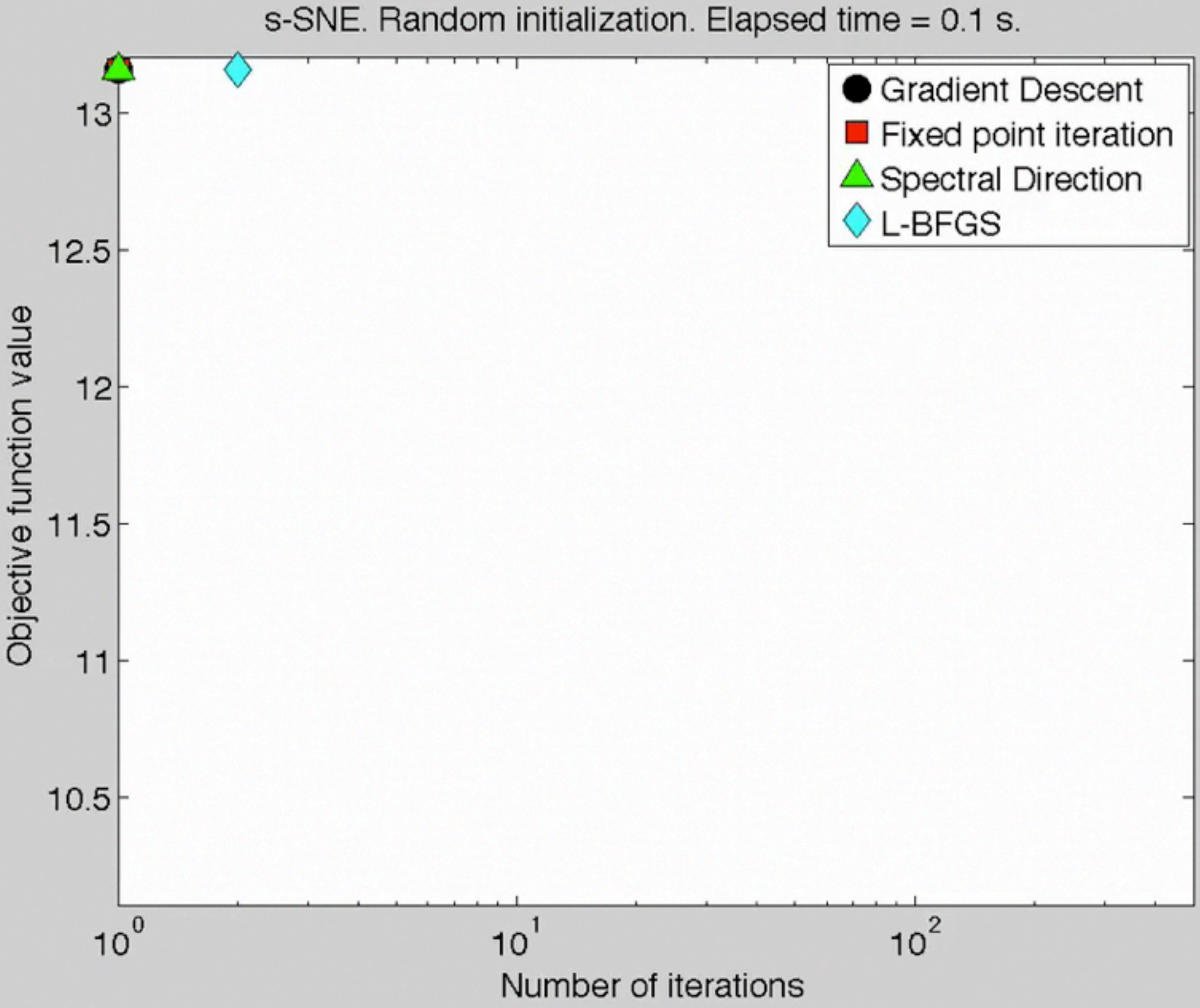
COIL-20. Convergence analysis, s-SNE

50 runs for each algorithm with random initial location.



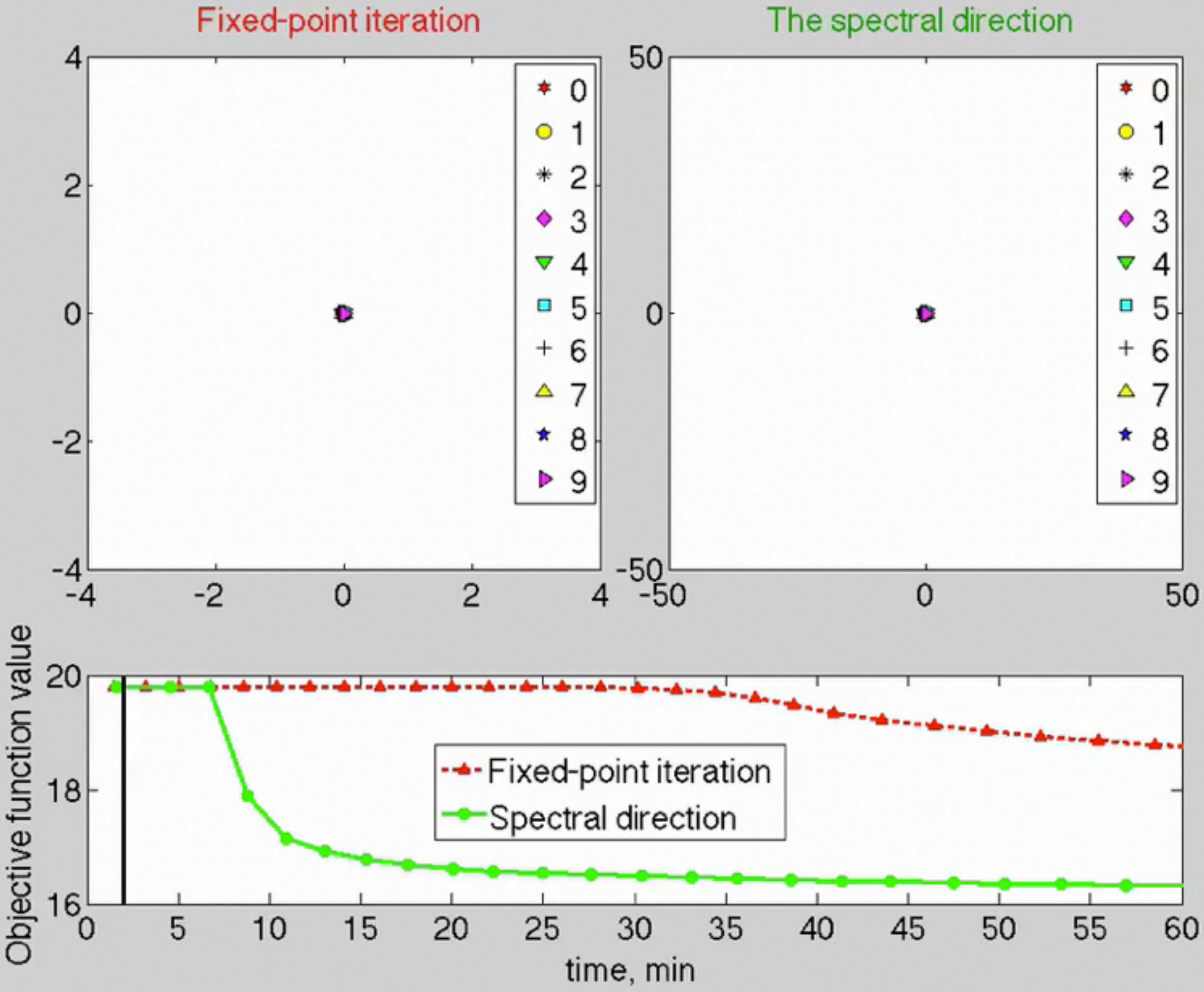
COIL-20. Convergence analysis, s-SNE

50 runs for each algorithm with random initial location.



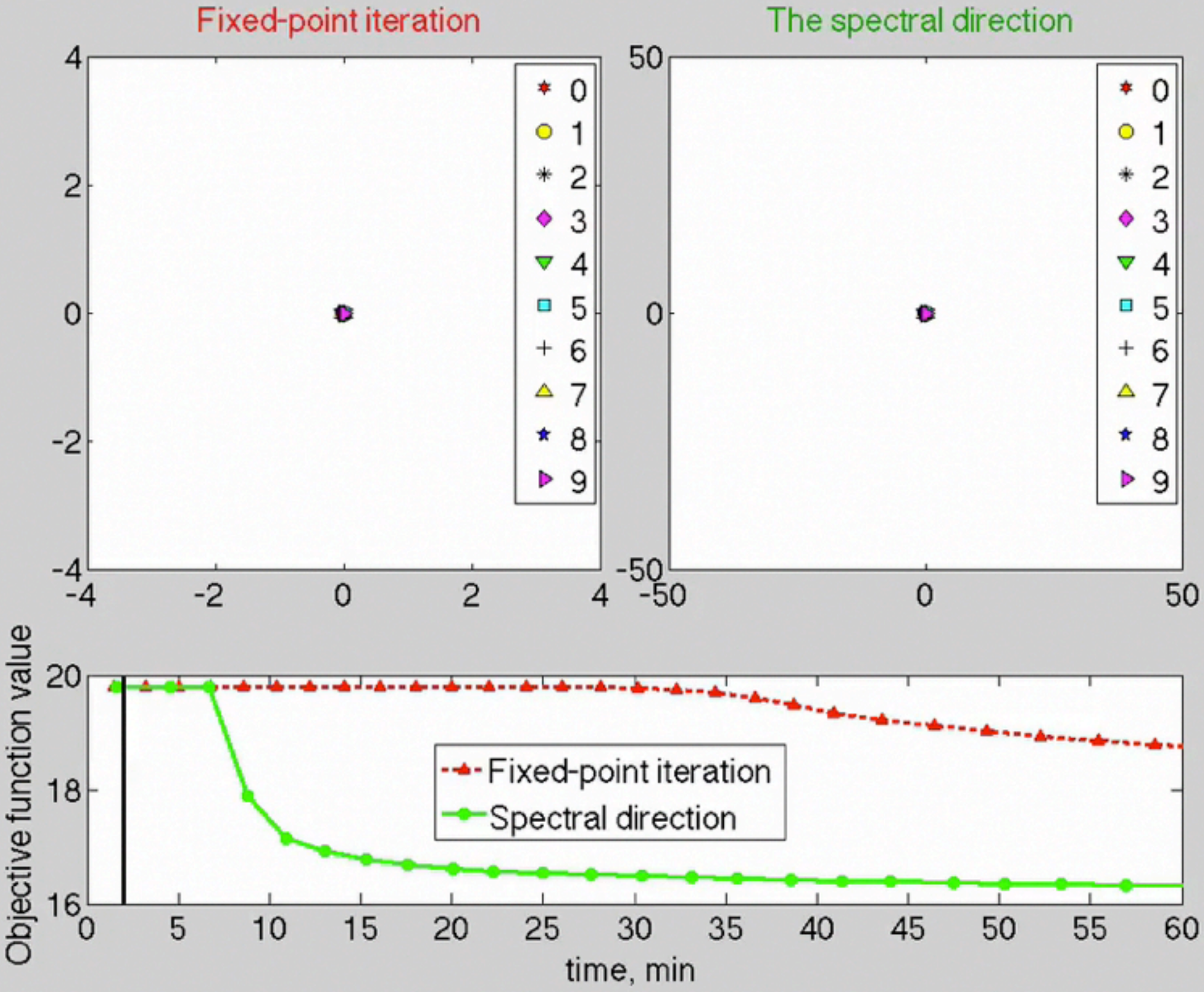
MNIST. Convergence analysis, *t*-SNE

Comparing fixed-point iteration to the spectral diction for $20\,000$ MNIST digits in one hour of optimization.

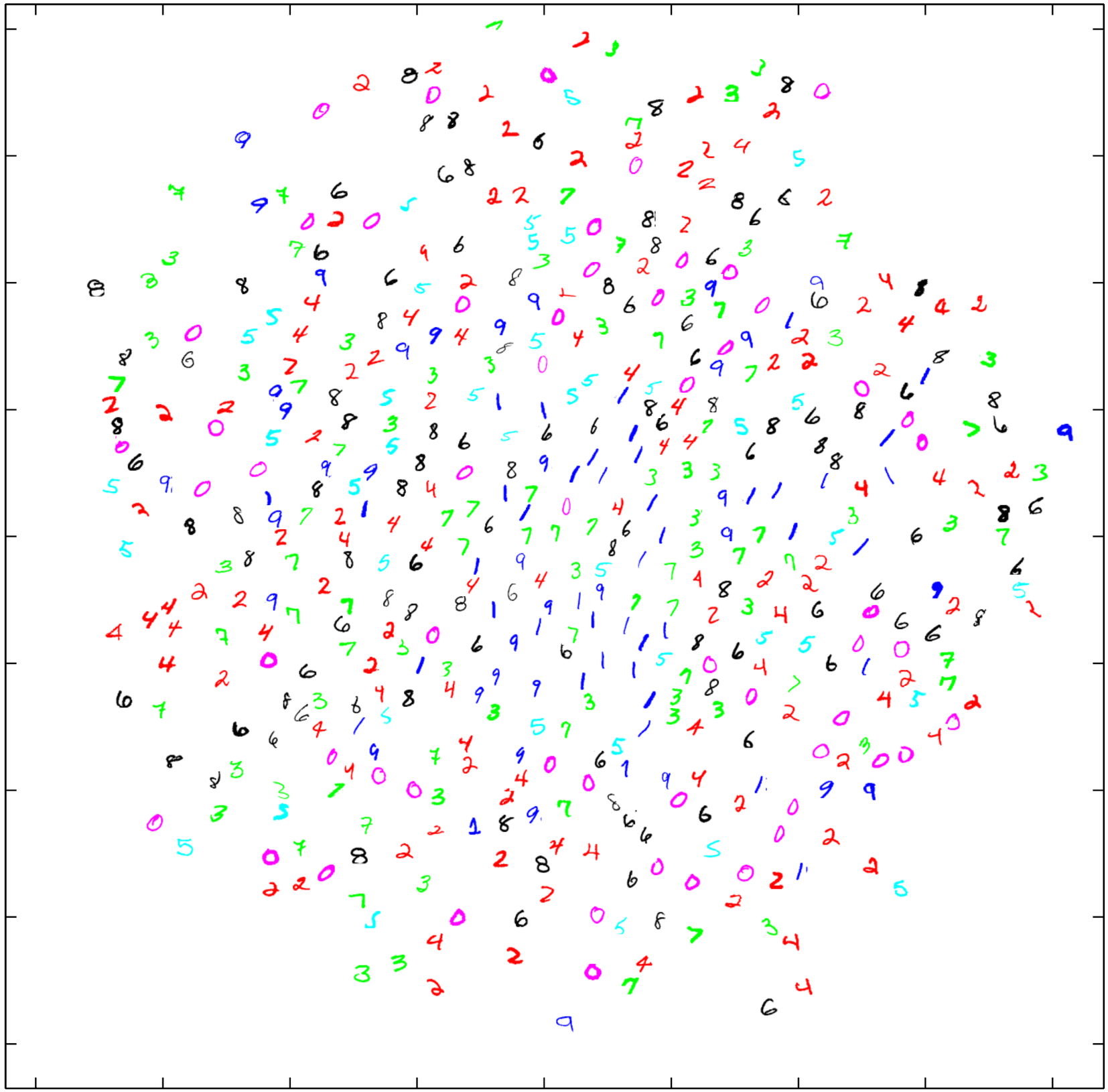


MNIST. Convergence analysis, *t*-SNE

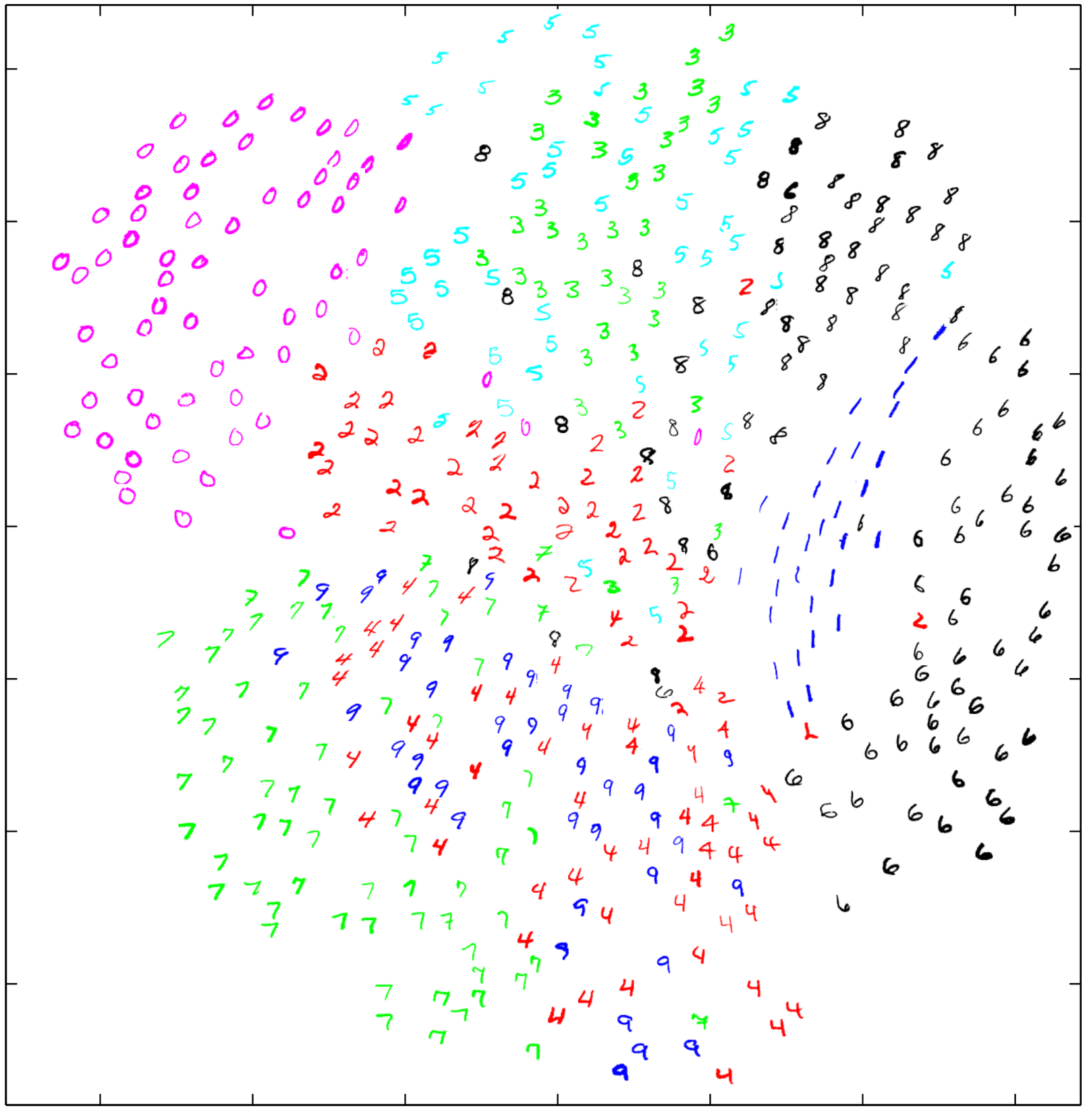
Comparing fixed-point iteration to the spectral diction for $20\,000$ MNIST digits in one hour of optimization.



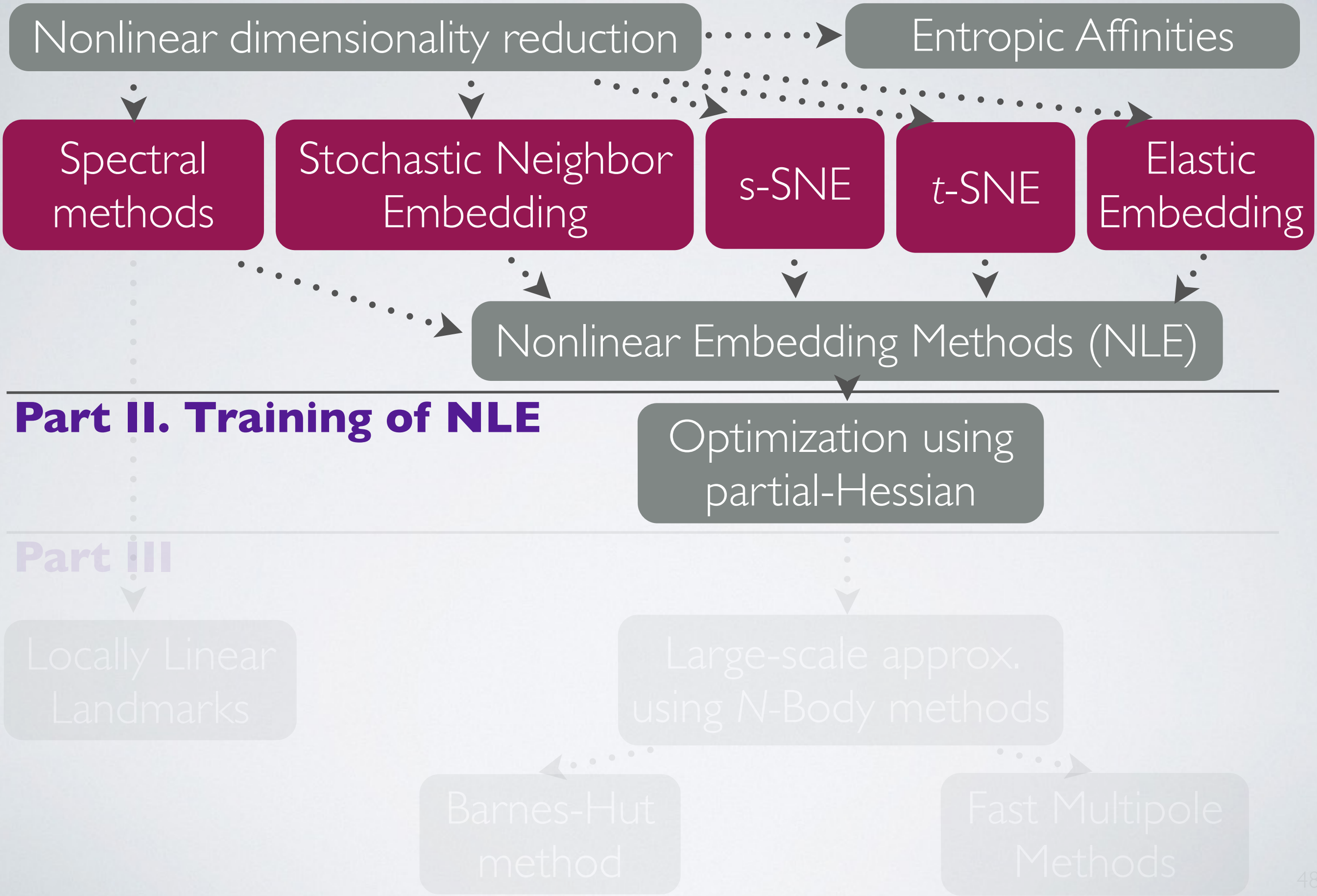
Fixed-point iteration, 20 min, EE



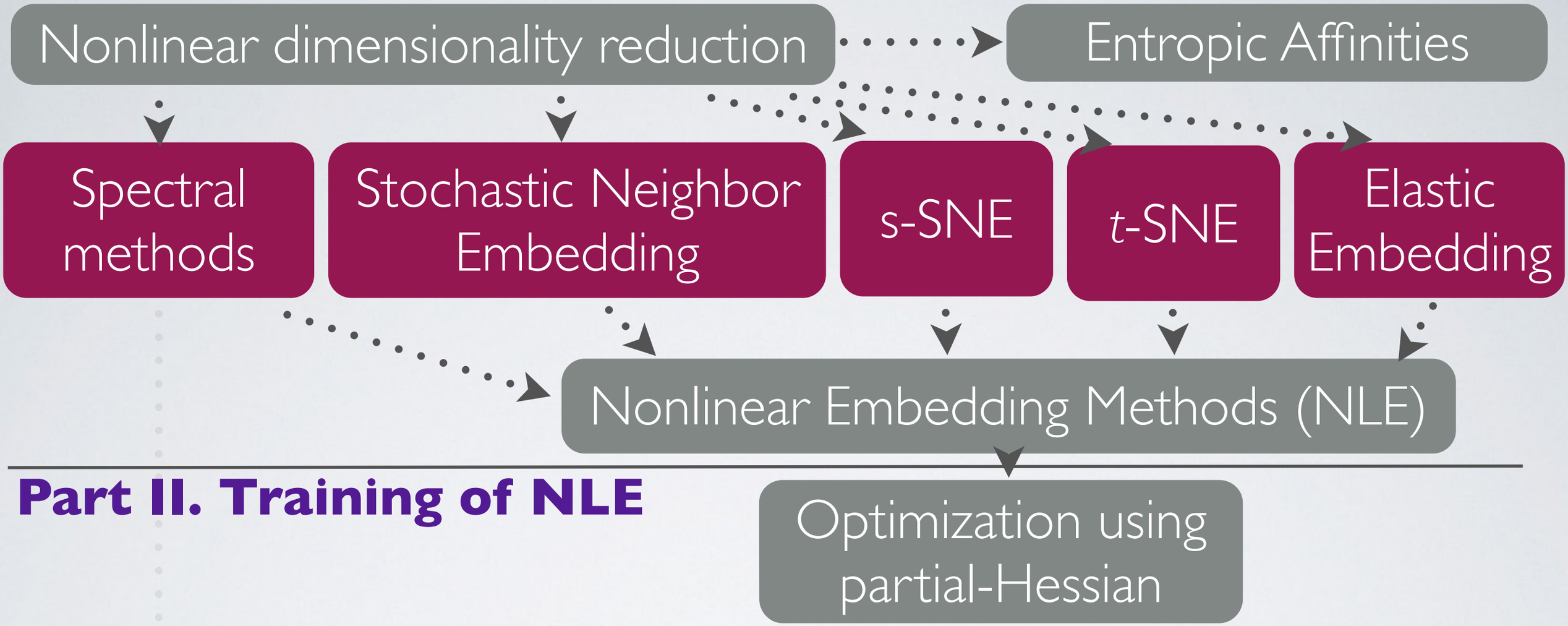
Spectral direction, 20 min, EE



Part I. Nonlinear dimensionality reduction



Part I. Nonlinear dimensionality reduction



Part II. Training of NLE

Part III. Scaling-up to large-scale datasets

Part I. Nonlinear dimensionality reduction

Nonlinear dimensionality reduction

Entropic Affinities

Spectral methods

Stochastic Neighbor Embedding

s-SNE

t-SNE

Elastic Embedding

Nonlinear Embedding Methods (NLE)

Optimization using partial-Hessian

Part II. Training of NLE

Part III. Scaling-up to large-scale datasets

Locally Linear Landmarks

Large-scale approx. using *N*-Body methods

Barnes-Hut method

Fast Multipole Methods

Problem of spectral methods

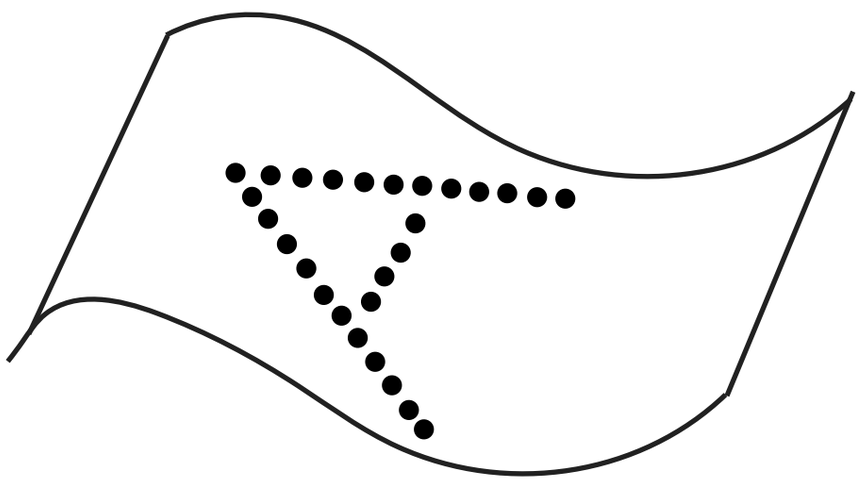
- Consider a spectral problem:

$$\min_X \operatorname{tr}\left(\mathbf{X}\mathbf{A}\mathbf{X}^T\right) \text{ s.t. } \mathbf{X}\mathbf{B}\mathbf{X}^T = \mathbf{I}$$

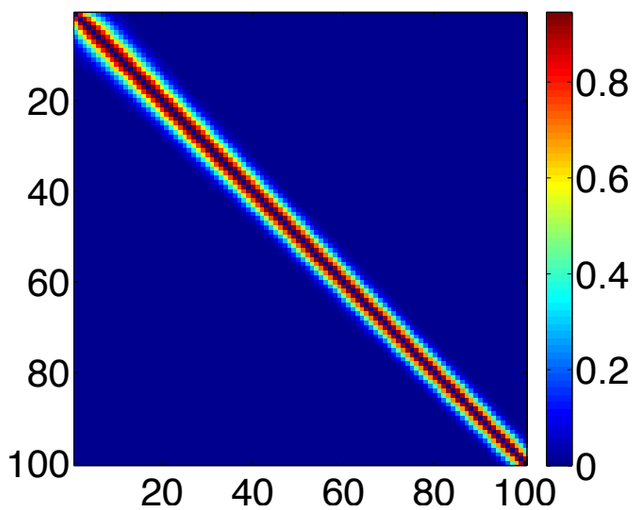
- Solution is unique and can be found in closed form for by the eigenvectors of $N \times N$ matrix constructed from $\mathbf{A}$ and $\mathbf{B}$.

With large N, solving this eigenproblem is *infeasible* even if $\mathbf{A}$ and $\mathbf{B}$ are sparse.

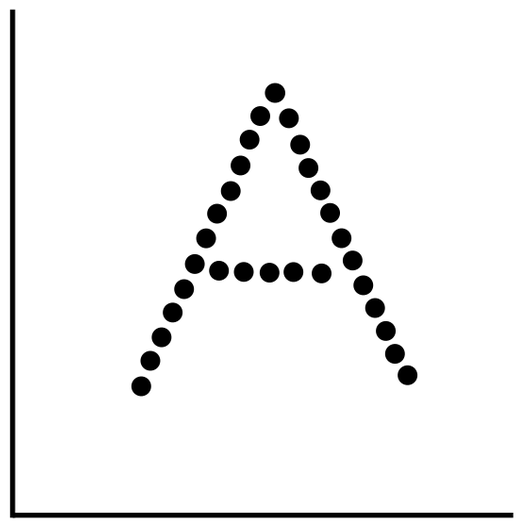
Original dataset $\mathbf{Y}$



Affinity matrix

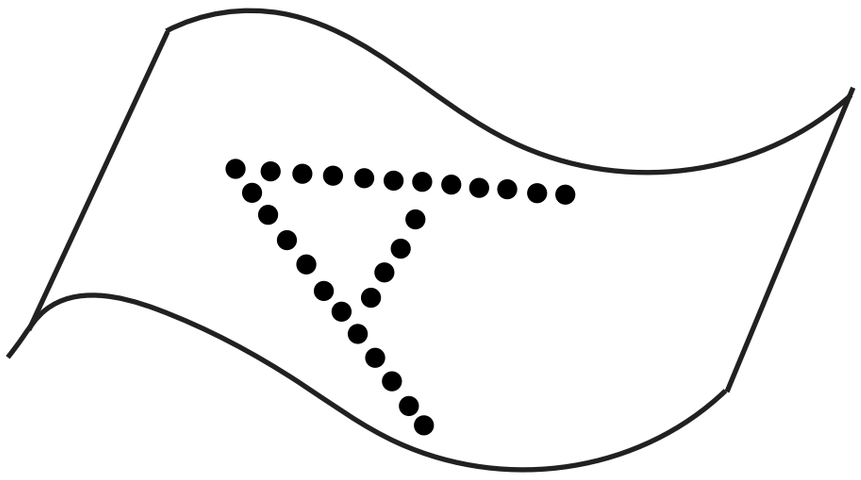


Learned $\mathbf{X}$

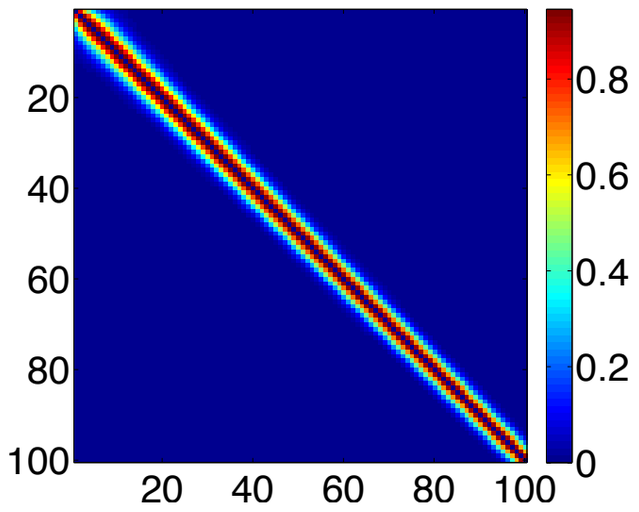


Learning with landmarks

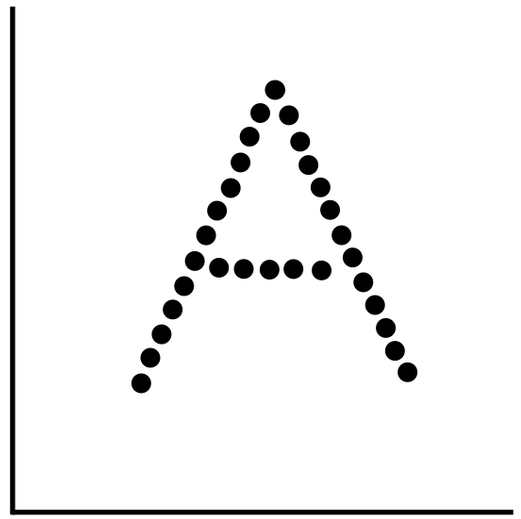
Original dataset $\mathbf{Y}$



Affinity matrix

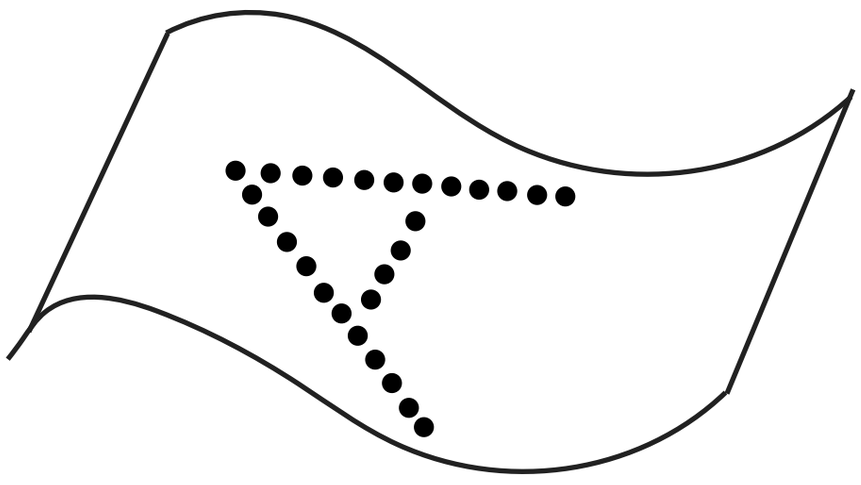


Learned $\mathbf{X}$

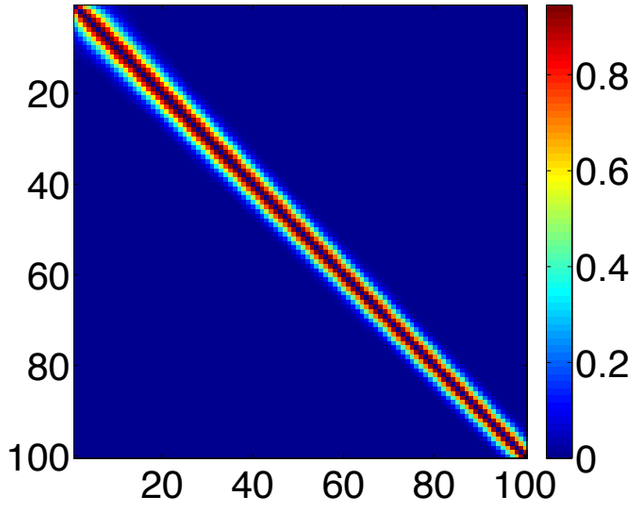


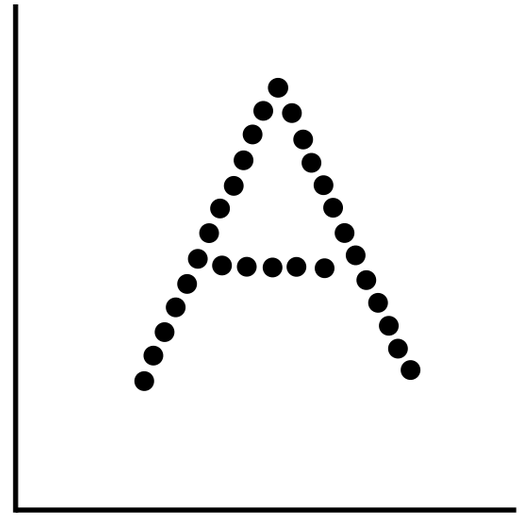
Learning with landmarks

Original dataset $\mathbf{Y}$



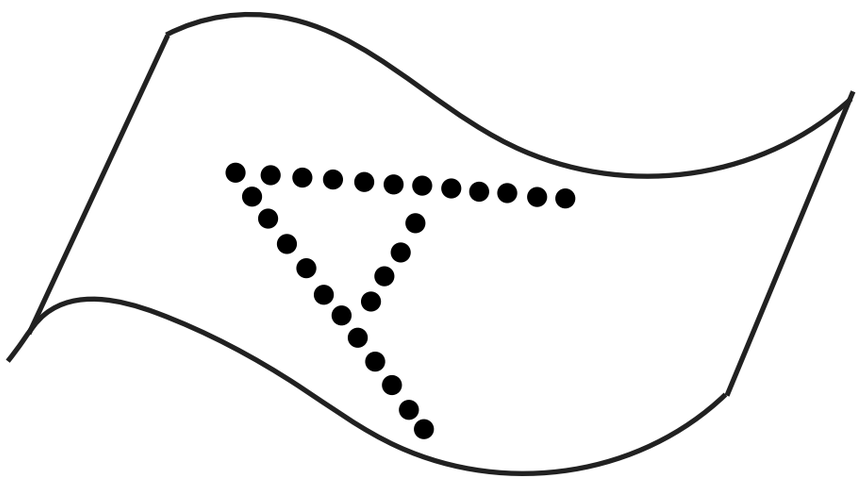
Affinity matrix



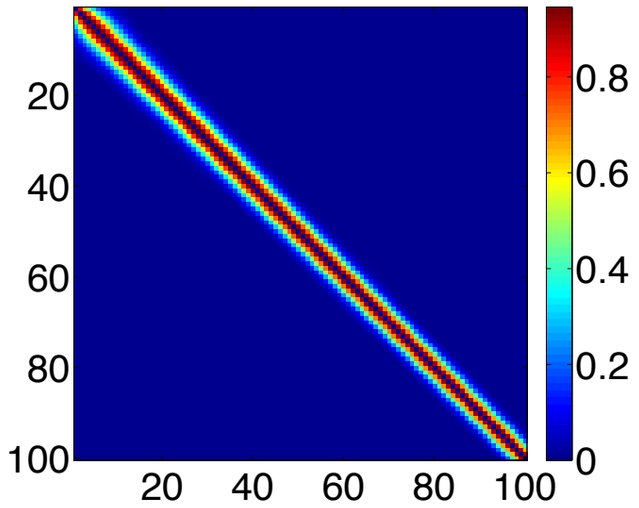


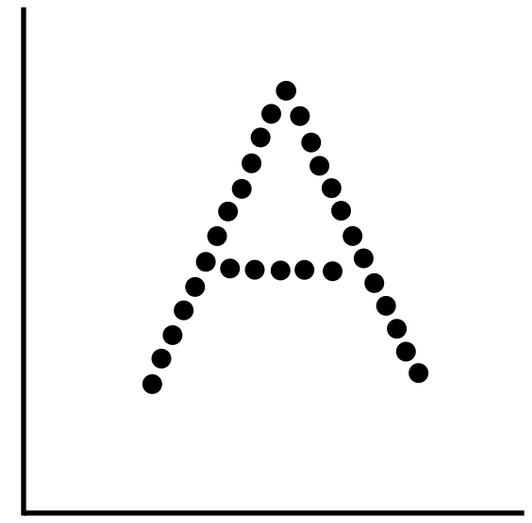
Learning with landmarks

Original dataset $\mathbf{Y}$

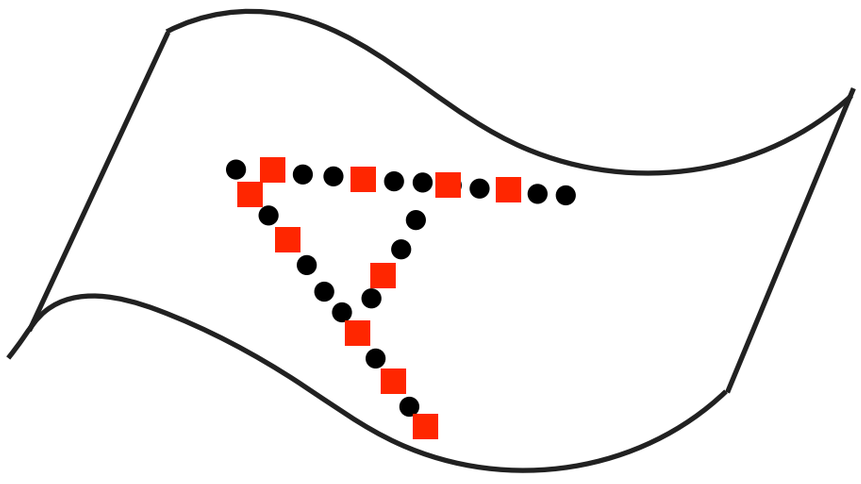


Affinity matrix

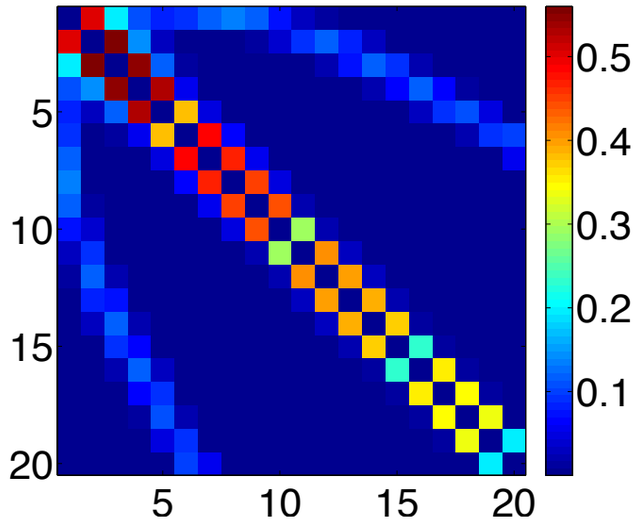




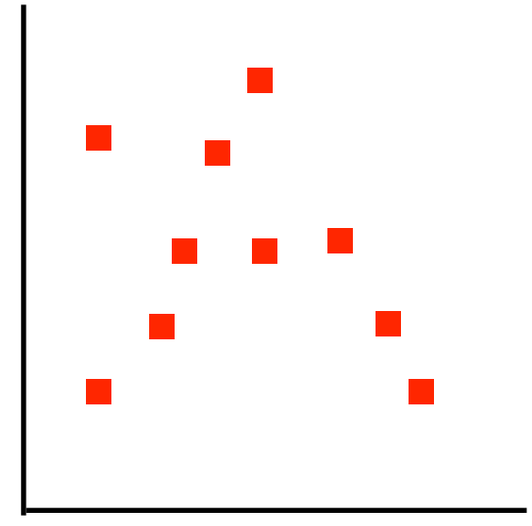
Landmarks $\tilde{\mathbf{Y}}$



Reduced affinity matrix



Learned $\tilde{\mathbf{X}}$

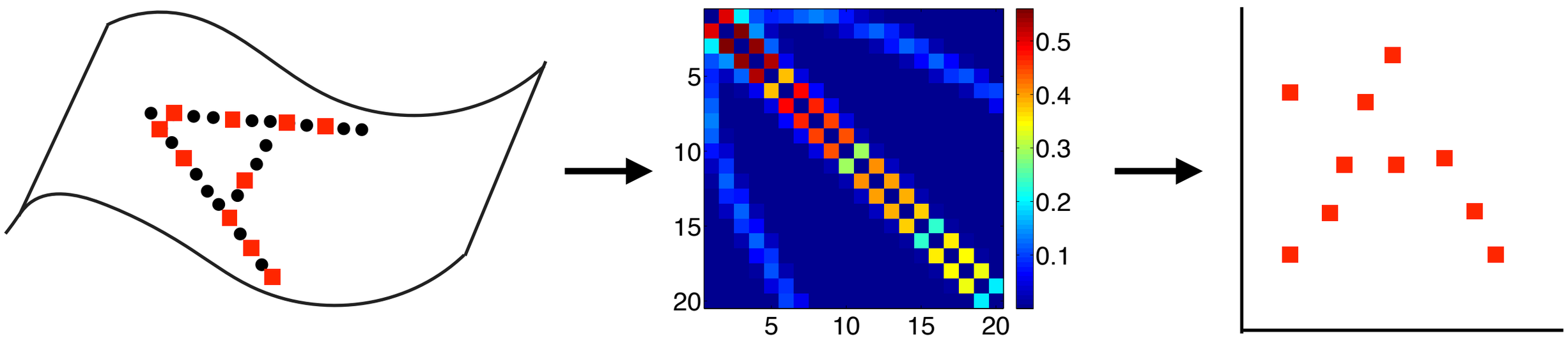


Learning with landmarks

Problems:

- We need a way to project the non-landmark points, e.g. with Nyström method (Talwalkar el at, 2008).
- It only uses the information in $\mathbf{A}$ about the landmarks, ignoring the non-landmarks. This requires using many landmarks to represent the data manifold well. If too few landmarks are used:
  - Bad solution for the landmarks $\widetilde{\mathbf{X}} = \widetilde{\mathbf{x}}_1 \ldots, \widetilde{\mathbf{x}}_L$.
  - …and bad prediction for the non-landmarks.

Landmarks $\widetilde{\mathbf{Y}}$ Reduced affinity matrix Learned $\widetilde{\mathbf{X}}$



Locally Linear Landmarks (LLL)

(Vladymyrov and Carreira-Perpiñán, '13)

- Assume each projection is a locally function of the landmarks:

$$\mathbf{x}_n = \sum_{l=1}^{L} z_{ln}\widetilde{\mathbf{x}}, n = 1, \dots, N \;\Rightarrow\; \mathbf{X} = \widetilde{\mathbf{X}}\mathbf{Z}$$

- Solving the original eigenproblem of $N \times N$ with this constraint results in a reduced eigenproblem of the same form but of $L \times L$ on $\widetilde{\mathbf{X}}$:

$$\min_{\widetilde{\mathbf{X}}} \operatorname{tr}\left(\widetilde{\mathbf{X}}\widetilde{\mathbf{A}}\widetilde{\mathbf{X}}^T\right) \text{ s.t. } \widetilde{\mathbf{X}}\widetilde{\mathbf{B}}\widetilde{\mathbf{X}}^T = \mathbf{I}$$

with reduced affinities $\widetilde{\mathbf{A}} = \mathbf{Z}\mathbf{A}\mathbf{Z}^T$, $\widetilde{\mathbf{B}} = \mathbf{Z}\mathbf{B}\mathbf{Z}^T$.

- After $\widetilde{\mathbf{X}}$ is found, the non-landmarks are predicted as $\mathbf{X} = \widetilde{\mathbf{X}}\mathbf{Z}$ (out-of-sample mapping).
- Advantages over Nyström method:
  - The reduced affinities $\widetilde{\mathbf{A}} = \mathbf{Z}\mathbf{A}\mathbf{Z}^T$ involve the entire dataset and contain much more information about the manifold that the landmark–landmark affinities, so fewer landmarks are needed.
  - Solving this smaller eigenproblem is faster.
  - The out-of-sample mapping requires less memory and is faster.

LLL: reduced affinities

Affinities between landmarks:

- Nyström (original affinities):

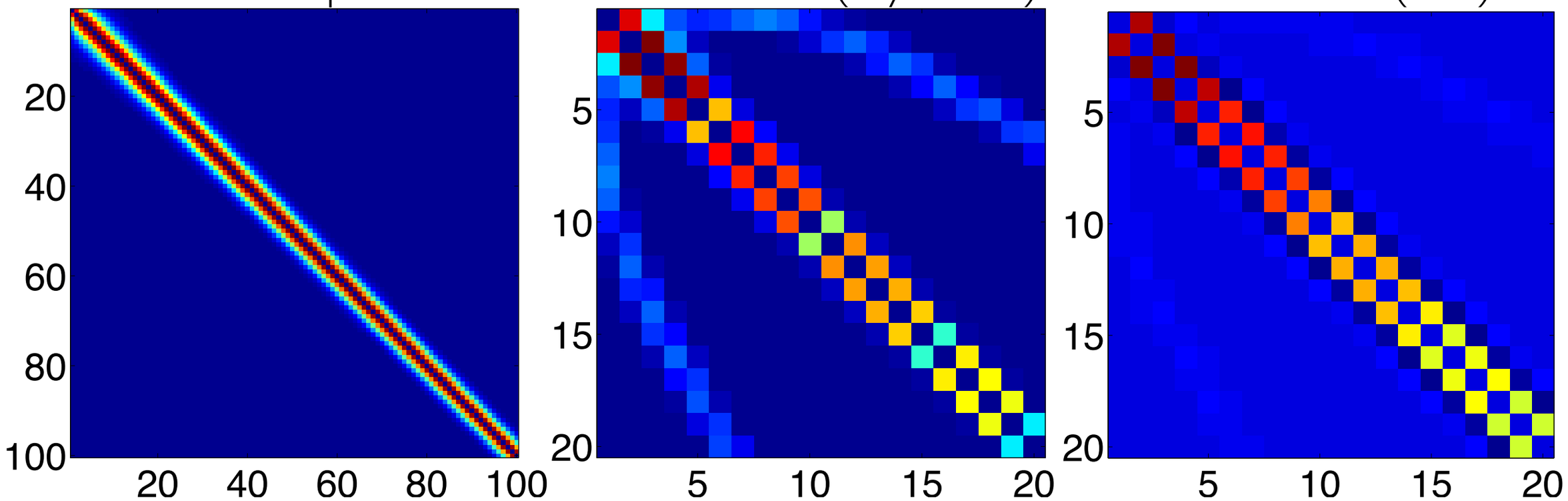
  $\mathbf{A} \Rightarrow a_{ij} \Rightarrow$ path i—j

- LLL (reduced affinities):

  $\widetilde{\mathbf{A}} = \mathbf{Z}\mathbf{A}\mathbf{Z}^T \Rightarrow \widetilde{a}_{ij} = \sum_{n,m=1}^{N} z_{in} a_{nm} z_{jm} \Rightarrow$ path i—n—m—j $\forall n, m$

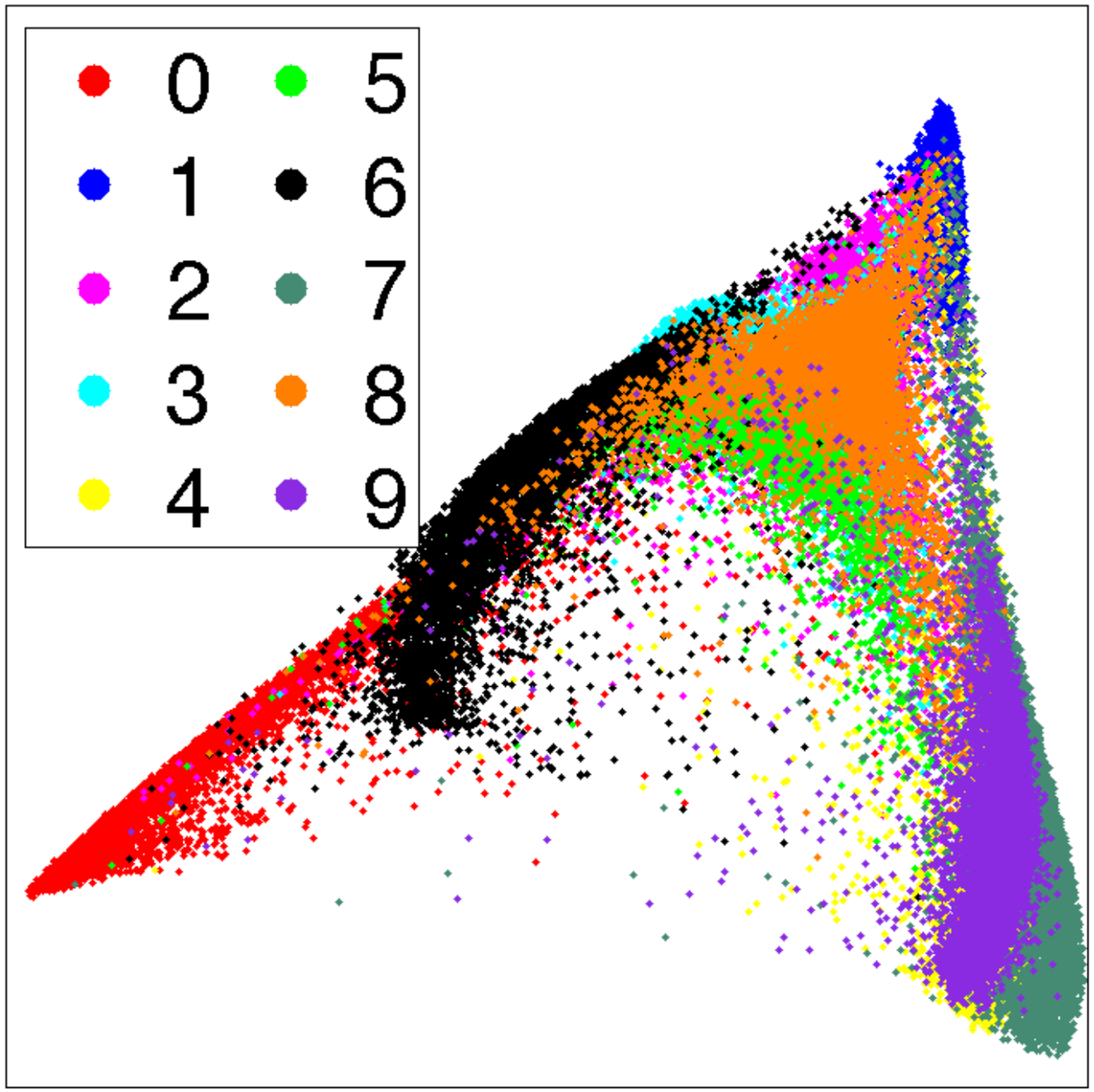
  So landmarks i and j can be farther apart and still be connected along the manifold.

Affinities between…

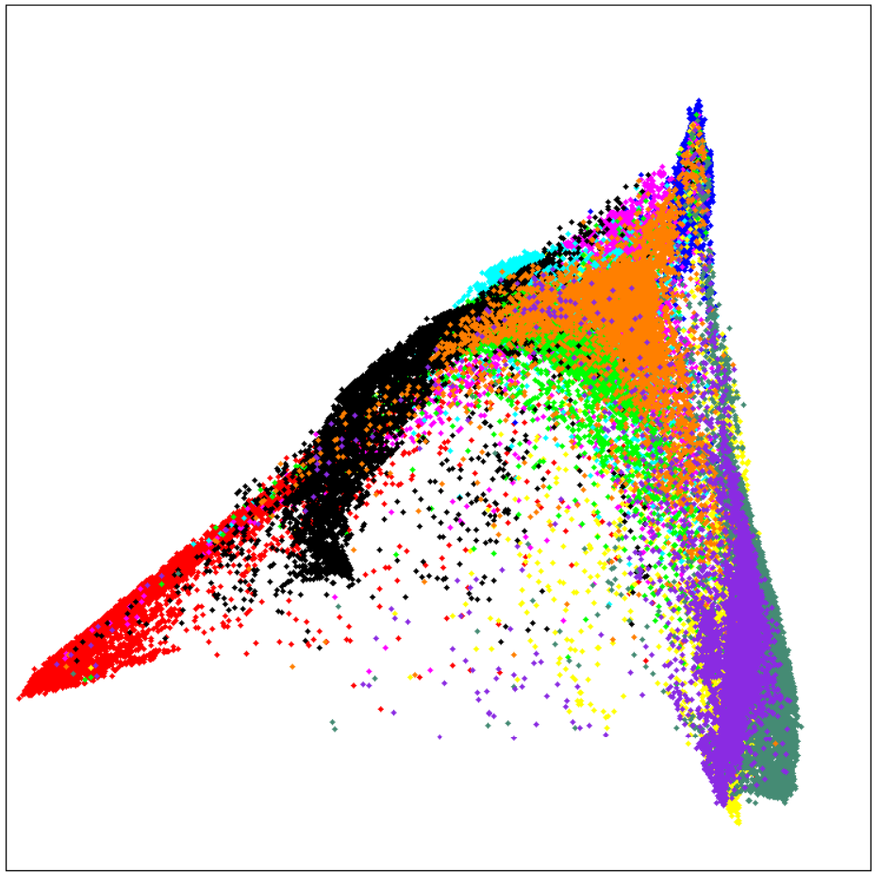


Experiments: MNIST dataset, $N = 60\,000$

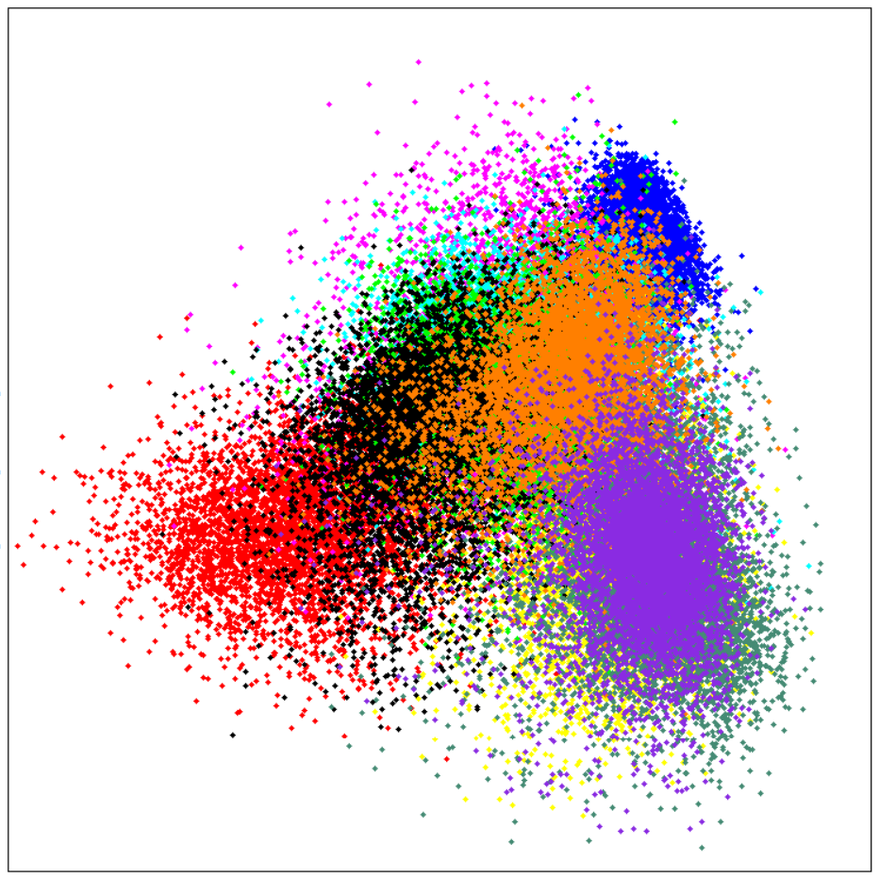
Exact LE, 80 s.



LLL, 5 s.



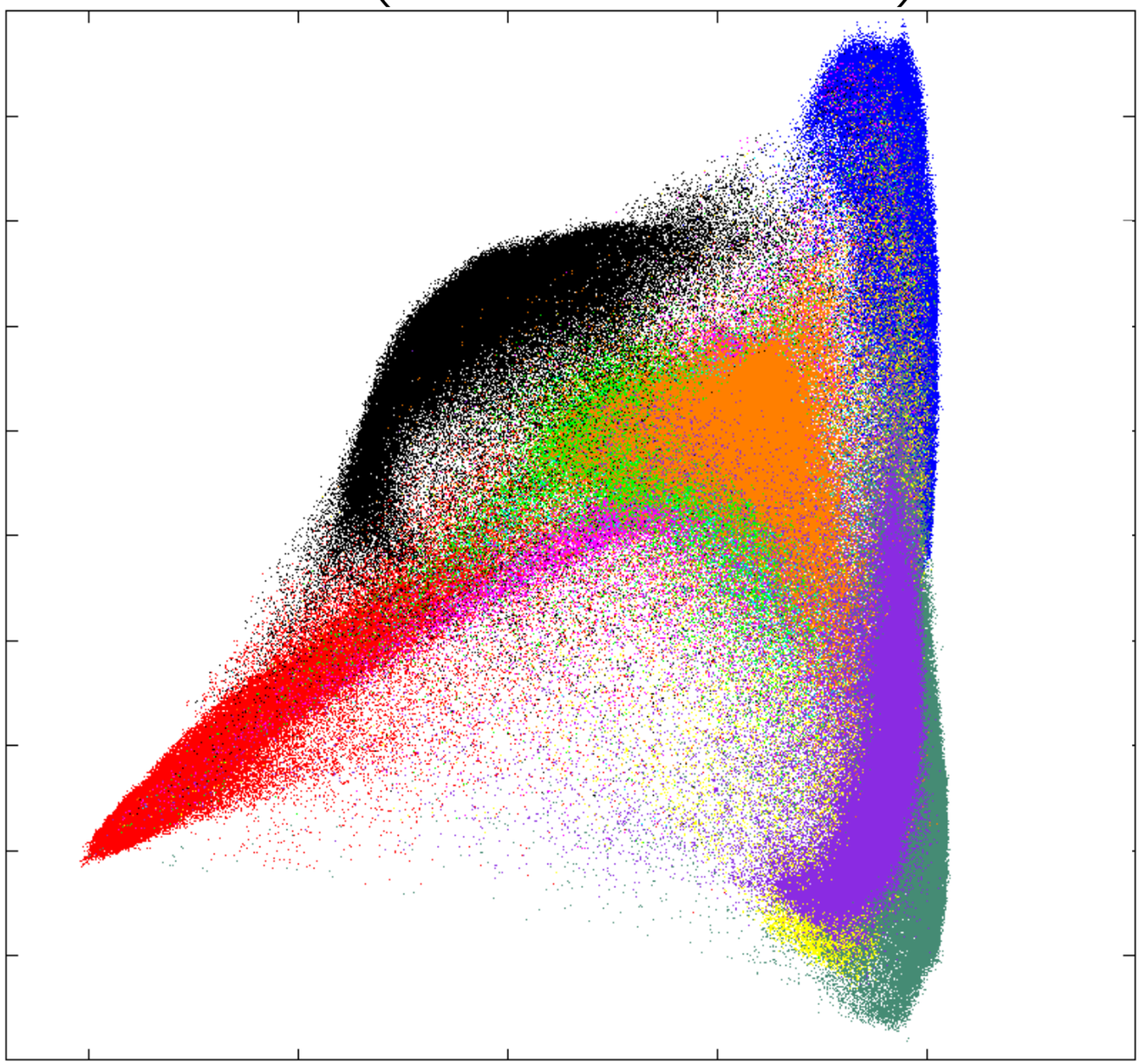
LE (Nys.), 5 s.



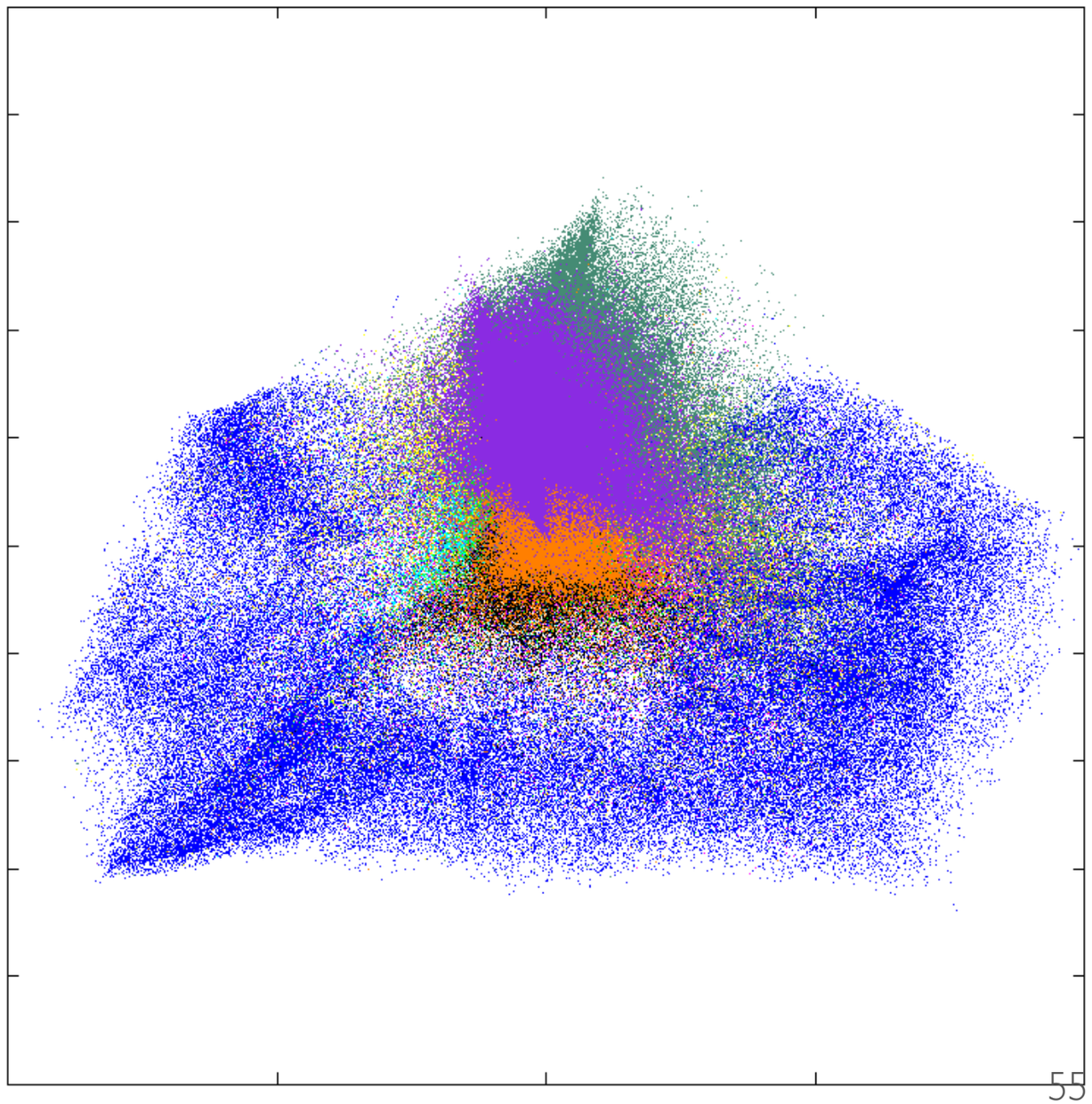
Experiments: large-scale dataset

- $N = 1\,020\,000$ points from infiniteMNIST.
- $L = 10^4$ random landmarks (1%).

LLL (18 min runtime)



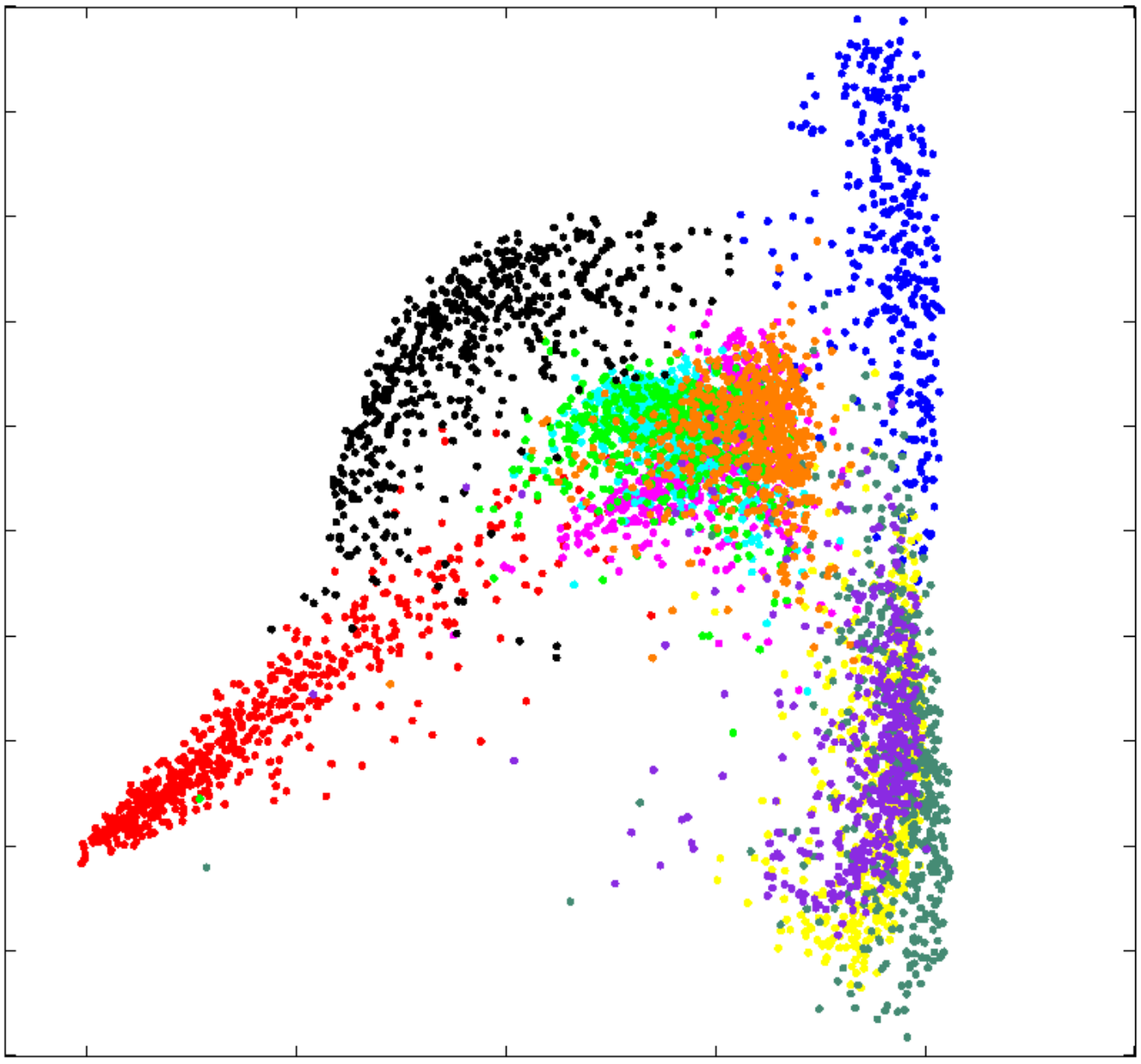
LE (with $\mathbf{Z}$ as an out of sample)



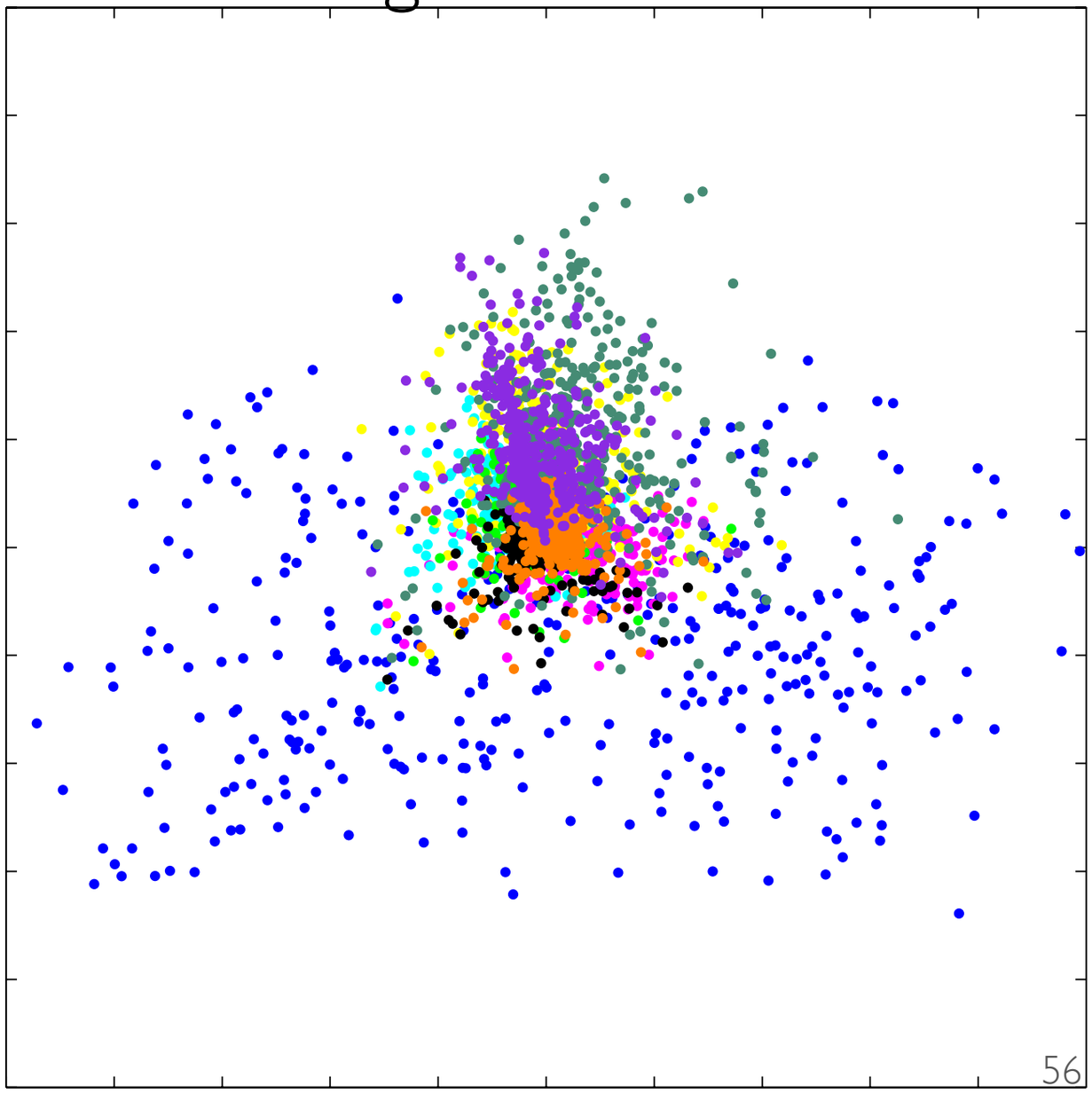
Experiments: large-scale dataset

The reason for the improved result with LLL is that it uses better affinities, so the landmarks are better projected.

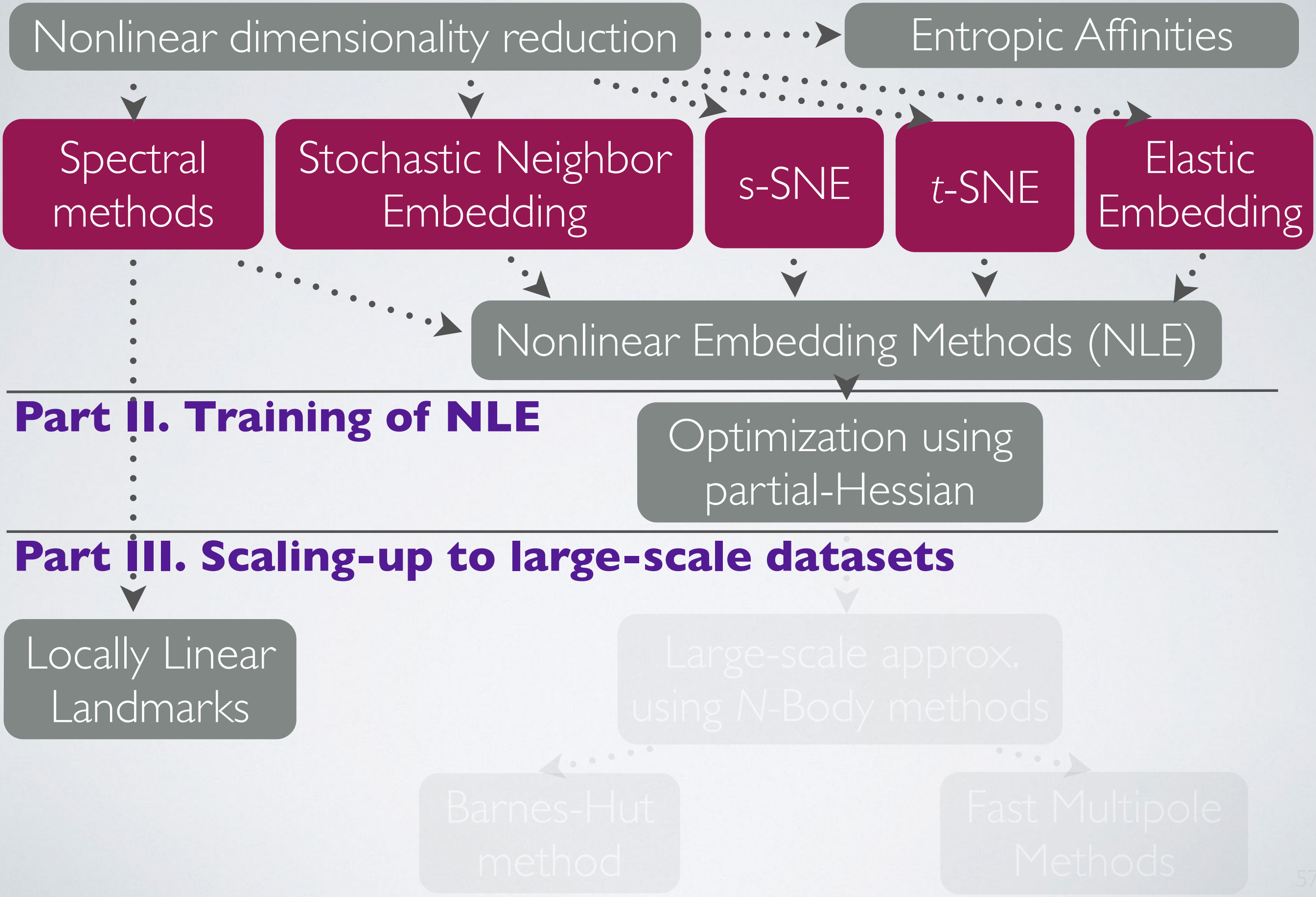
Landmarks with
LLL reduced affinities



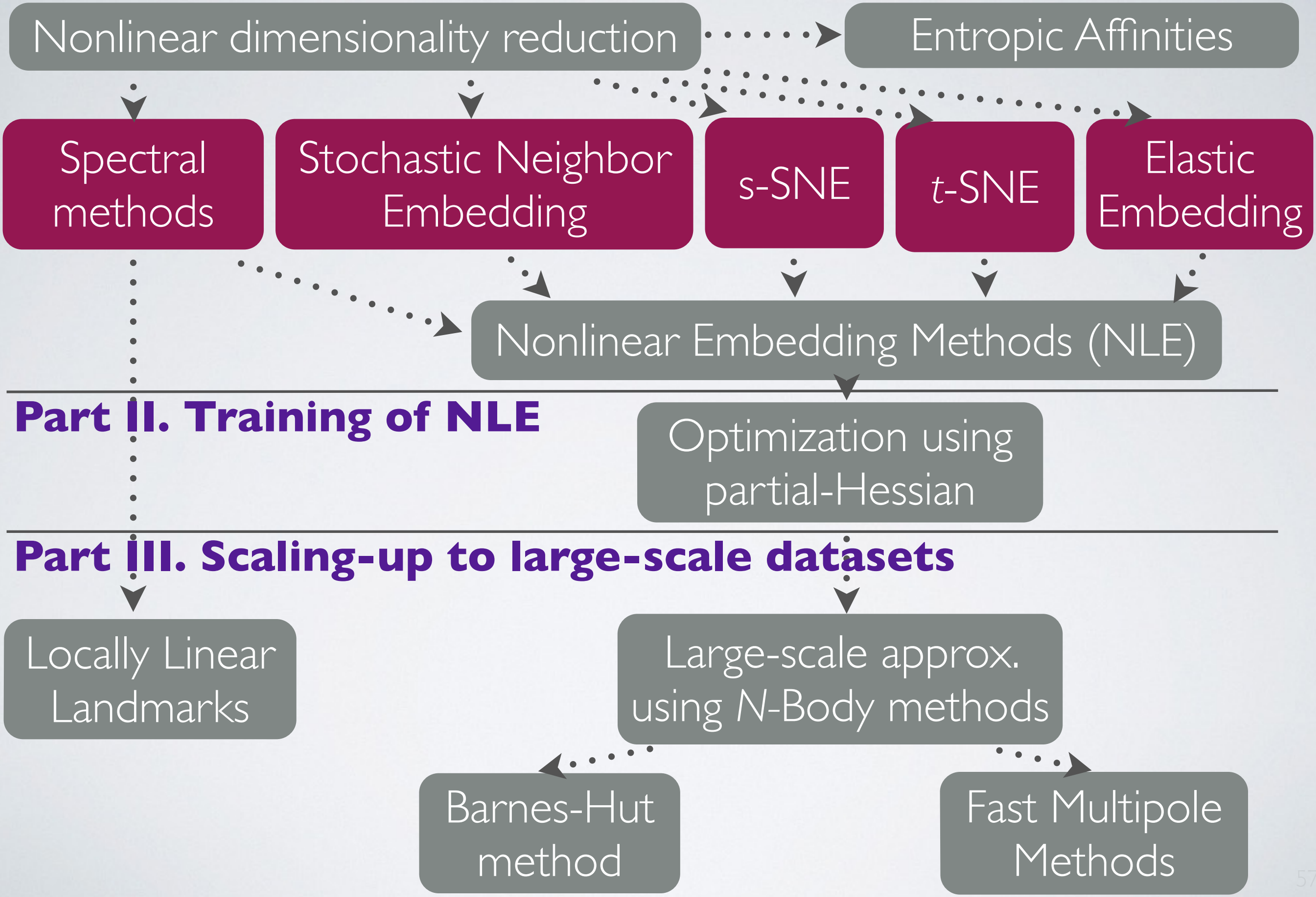
Landmarks with
original affinities



Part I. Nonlinear dimensionality reduction



Part I. Nonlinear dimensionality reduction

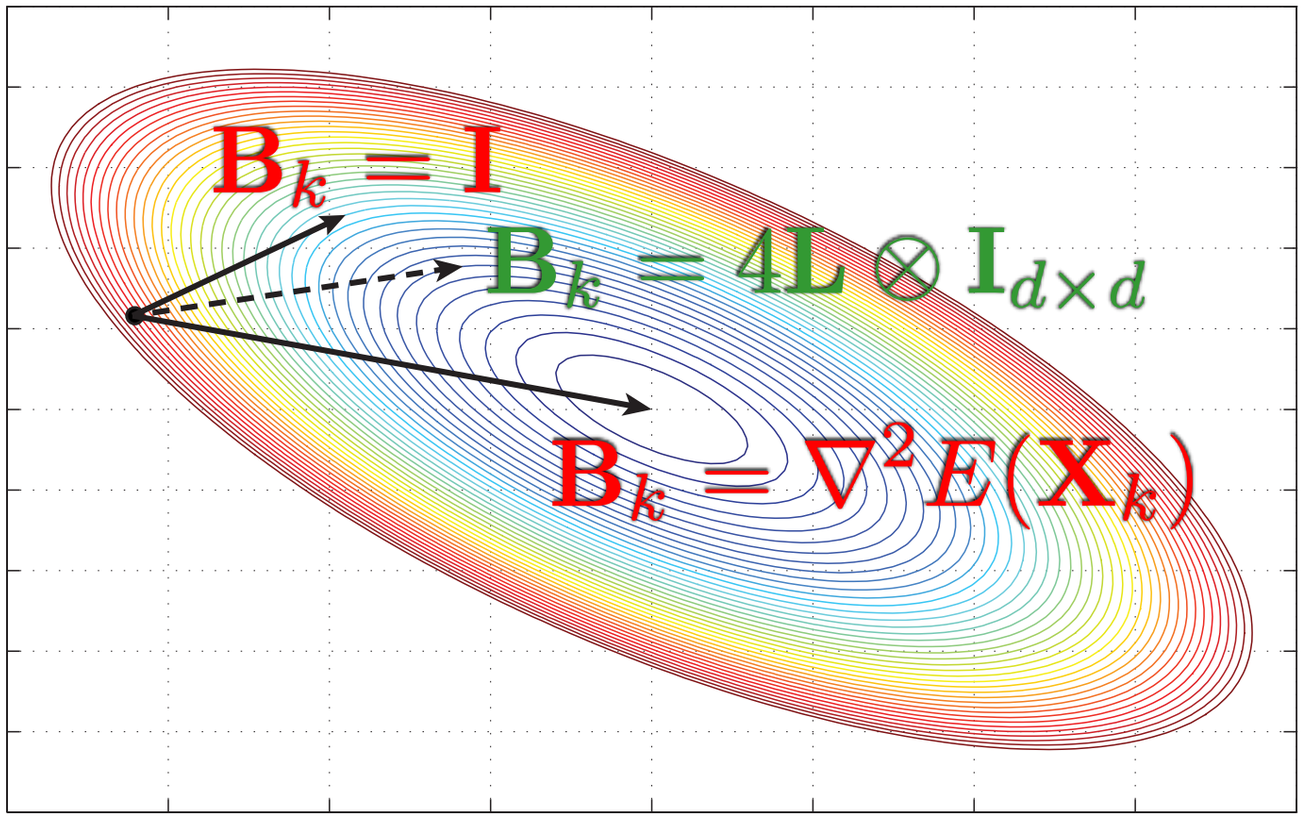


Optimization of NLE

For every iteration k:

- compute the gradient $\mathbf{G}_k$,
- find search direction $\mathbf{P}_k$,
- use line search to find a step size η for the next iteration:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta \mathbf{P}_k$$



Spectral direction, as well as other gradient-based methods require gradient and objective function evaluations for every iteration.

Computational bottleneck of NLE

In elastic embedding algorithm objective function and the gradient are given by:

$$E_{EE}(\mathbf{X}) = \sum_{n,m=1}^{N} w_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \lambda \sum_{n=1}^{N} S(\mathbf{x}_n)$$

$$G_{EE}(\mathbf{X}) = 4\mathbf{X}\mathbf{L} - 4\lambda\mathbf{X}\,\mathrm{diag}\left(S(\mathbf{X})\right) + 4\lambda S^x(\mathbf{X})$$

with

$$S(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{e}^{-\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2} \qquad S^x(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{x}_m \mathbf{e}^{-\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2}$$

Computing $S^x(\mathbf{x}_n)$ and $S(\mathbf{x}_n)$ for every $n = 1, \ldots, N$ is $\mathcal{O}(N^2)$.

No matter how fast is the optimization, it just decreases the number of iterations required for convergence. Each iteration is still $\mathcal{O}(N^2)$ because of the gradient and objective function evaluations!

Computational bottleneck of NLE

- The bottleneck of the algorithm is computation of the pairwise interaction between data points (*N*-body problem).

$$S(\mathbf{x}_n) = \sum_{m=1}^{N} e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2} \qquad S^x(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{x}_m e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2}$$

- Solution: use approximate methods to compute these interactions!
  - tree-based methods;
  - fast multipole methods.

Computational bottleneck of NLE

- The bottleneck of the algorithm is computation of the pairwise interaction between data points (*N*-body problem).

$$S(\mathbf{x}_n) = \sum_{m=1}^{N} e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2} \qquad S^x(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{x}_m e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2}$$

- Solution: use approximate methods to compute these interactions!
  - tree-based methods;
  - fast multipole methods.

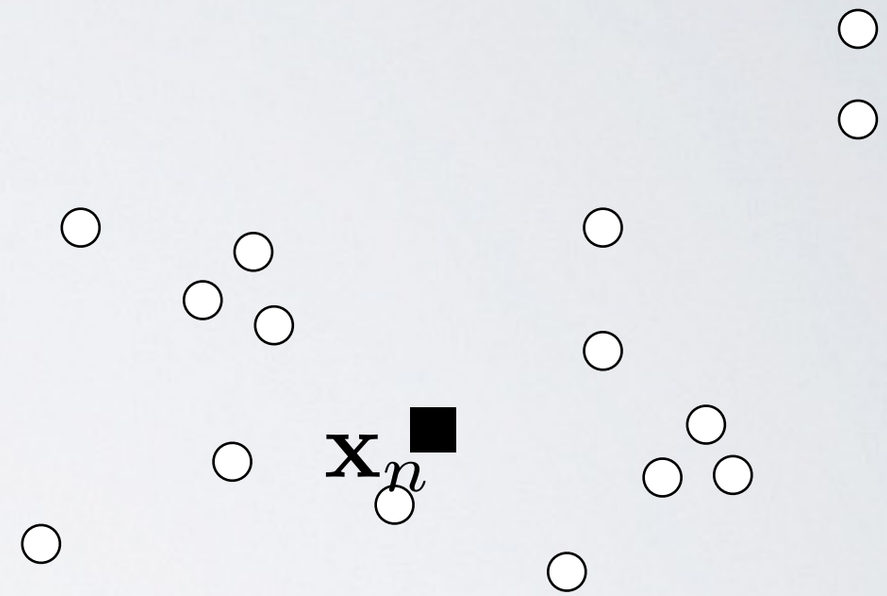
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.



- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually

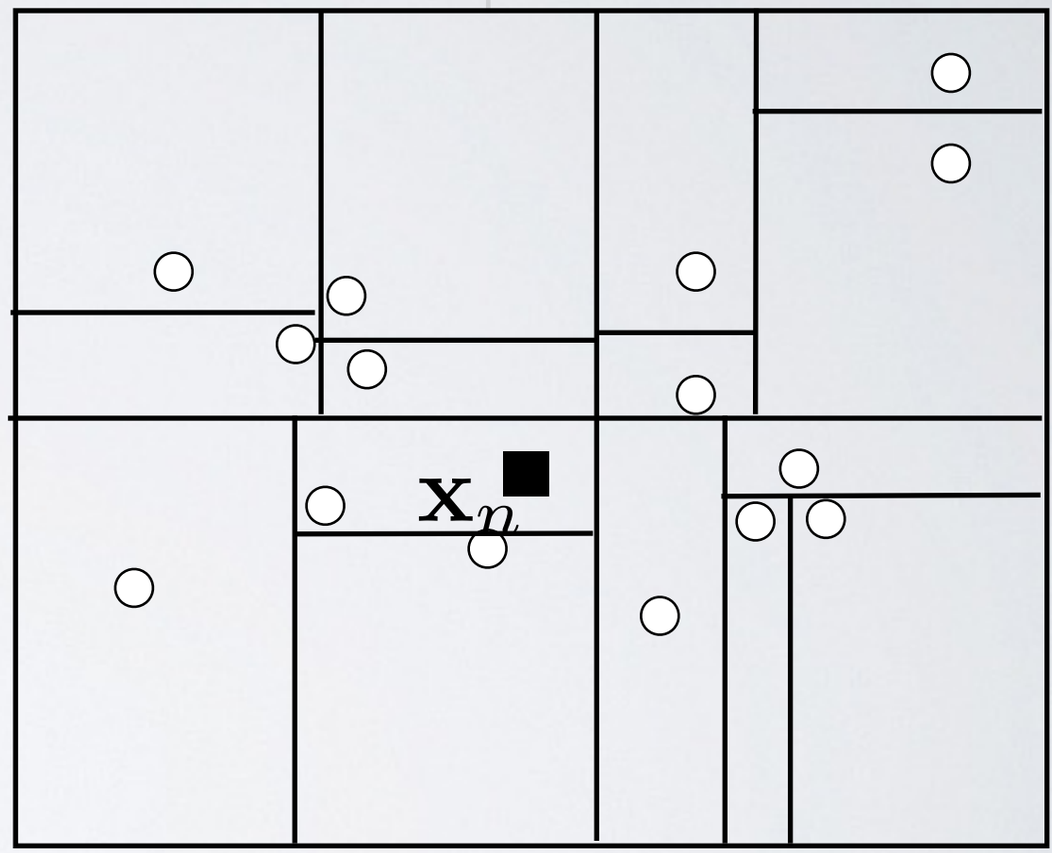
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.
- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually



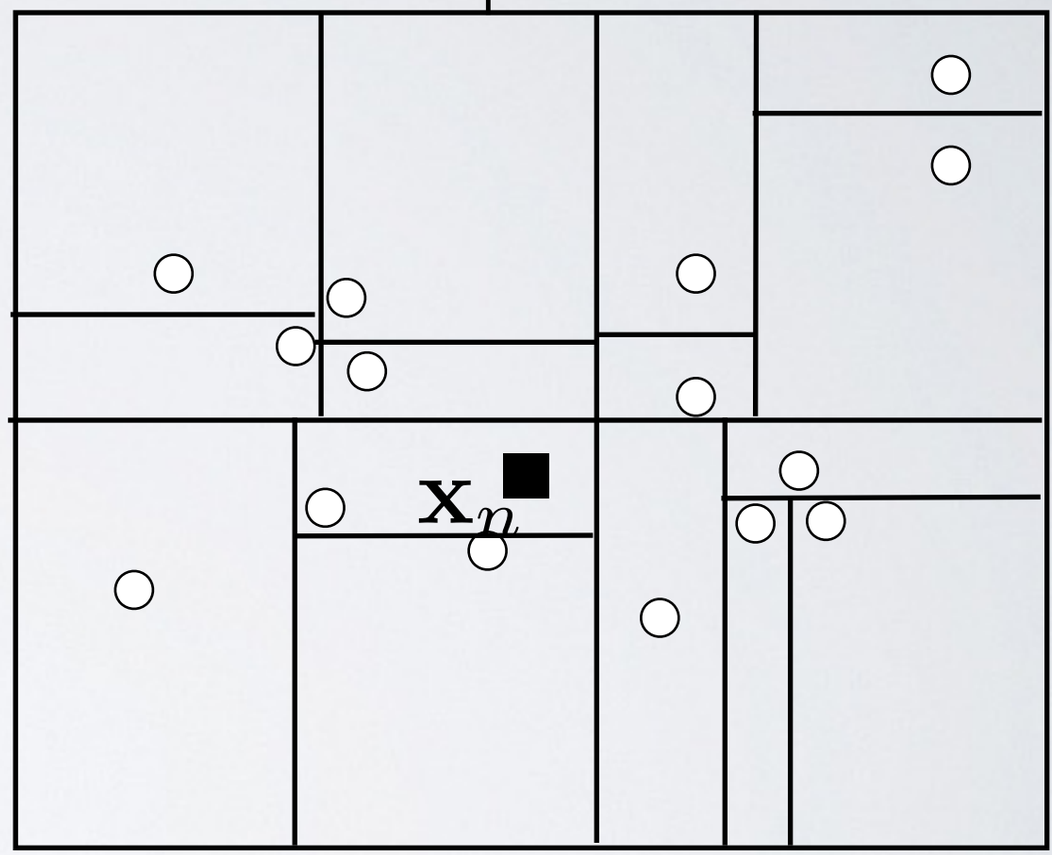
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.
- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually



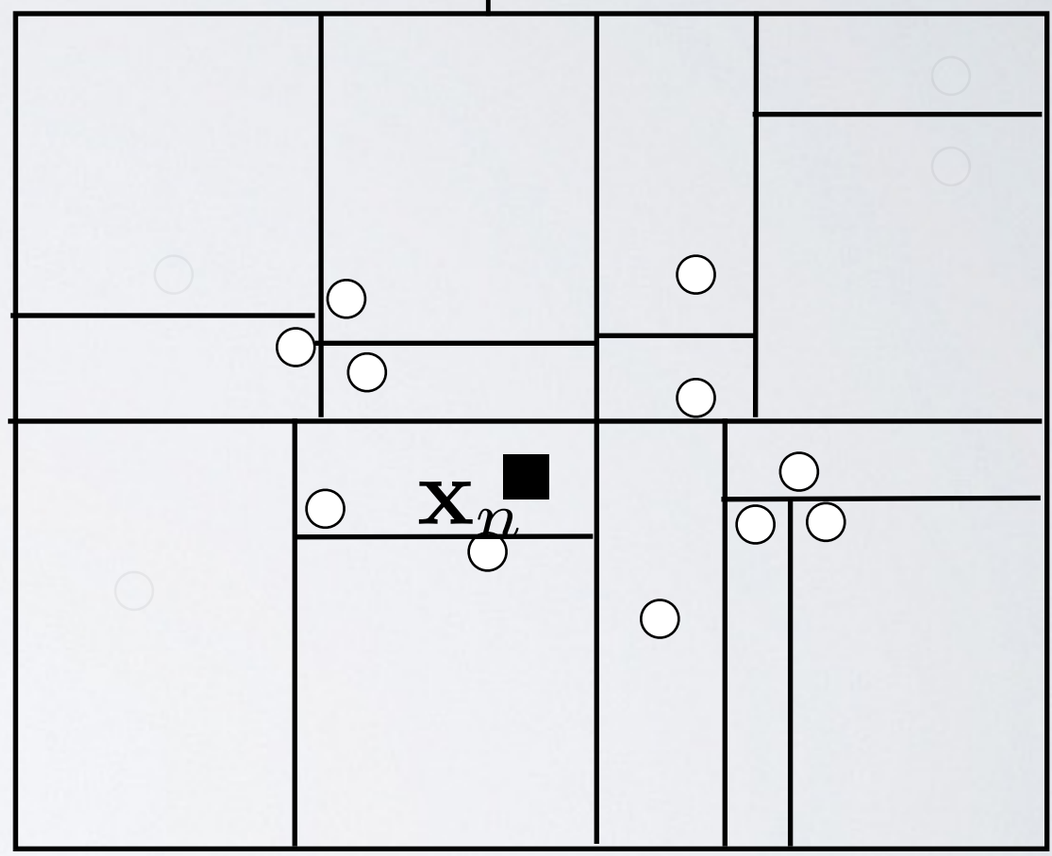
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.
- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually



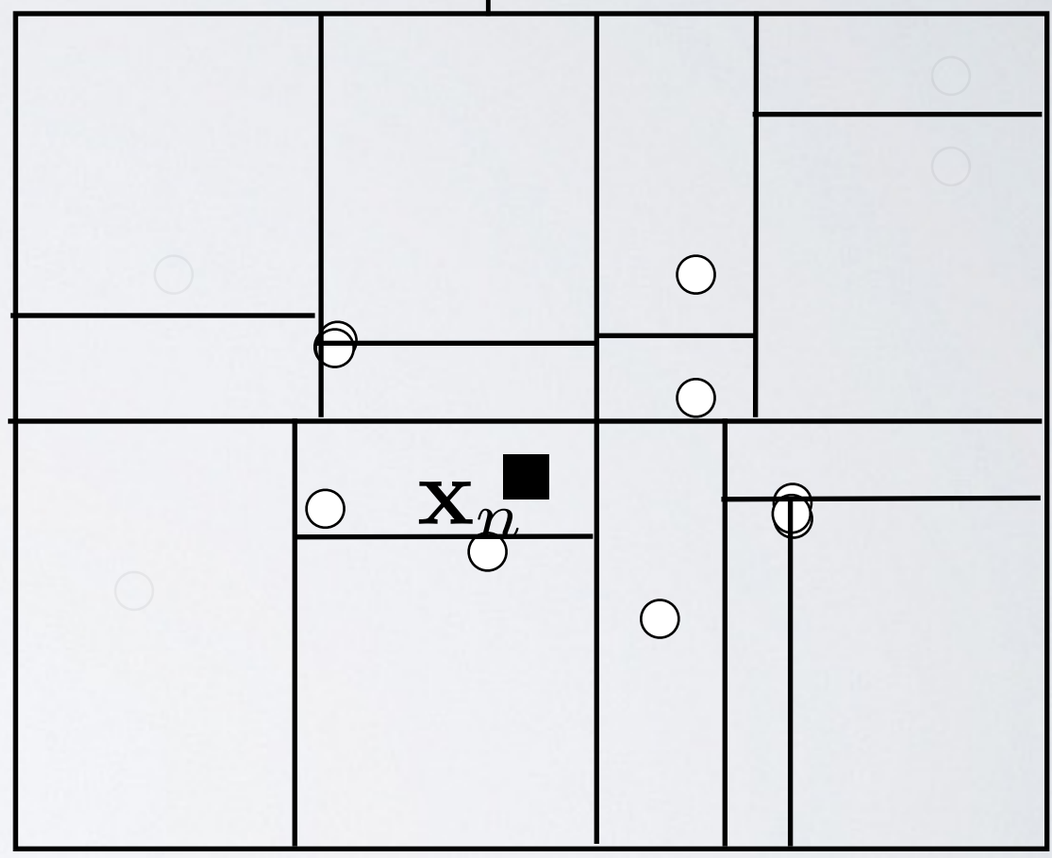
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.



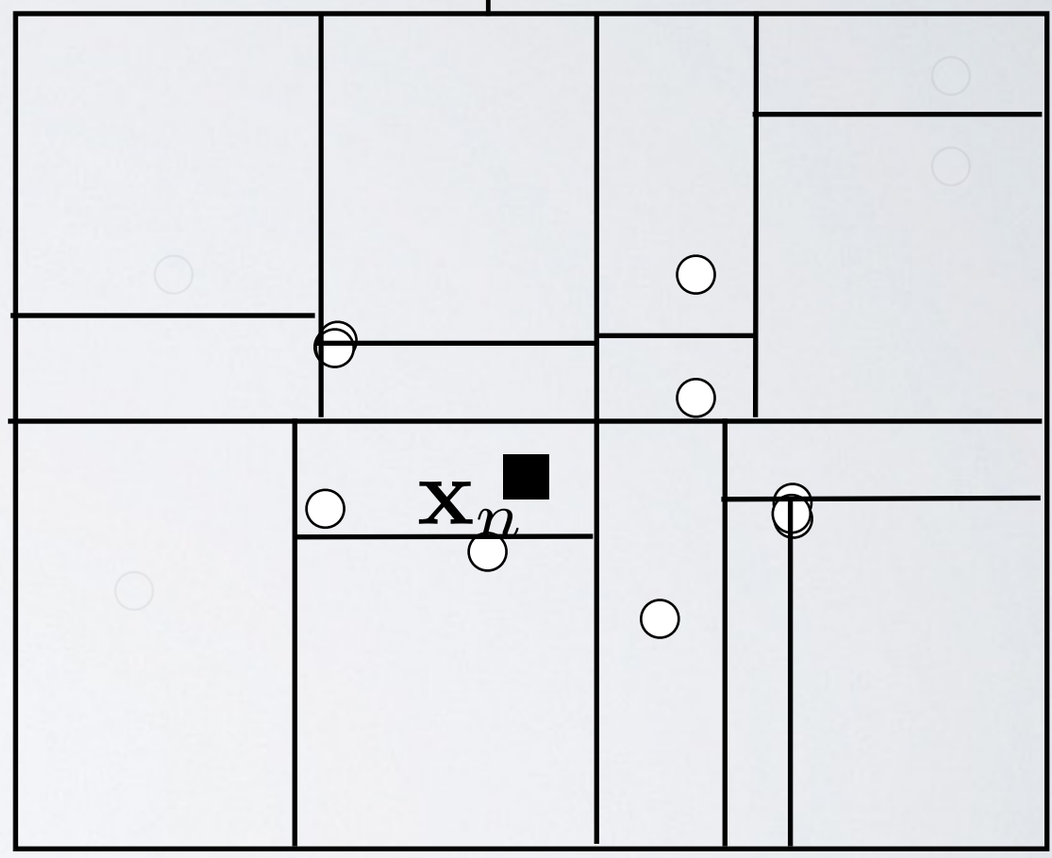
- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually

Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.
  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.
- Complexity is usually $\mathcal{O}(N \log N)$.
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually

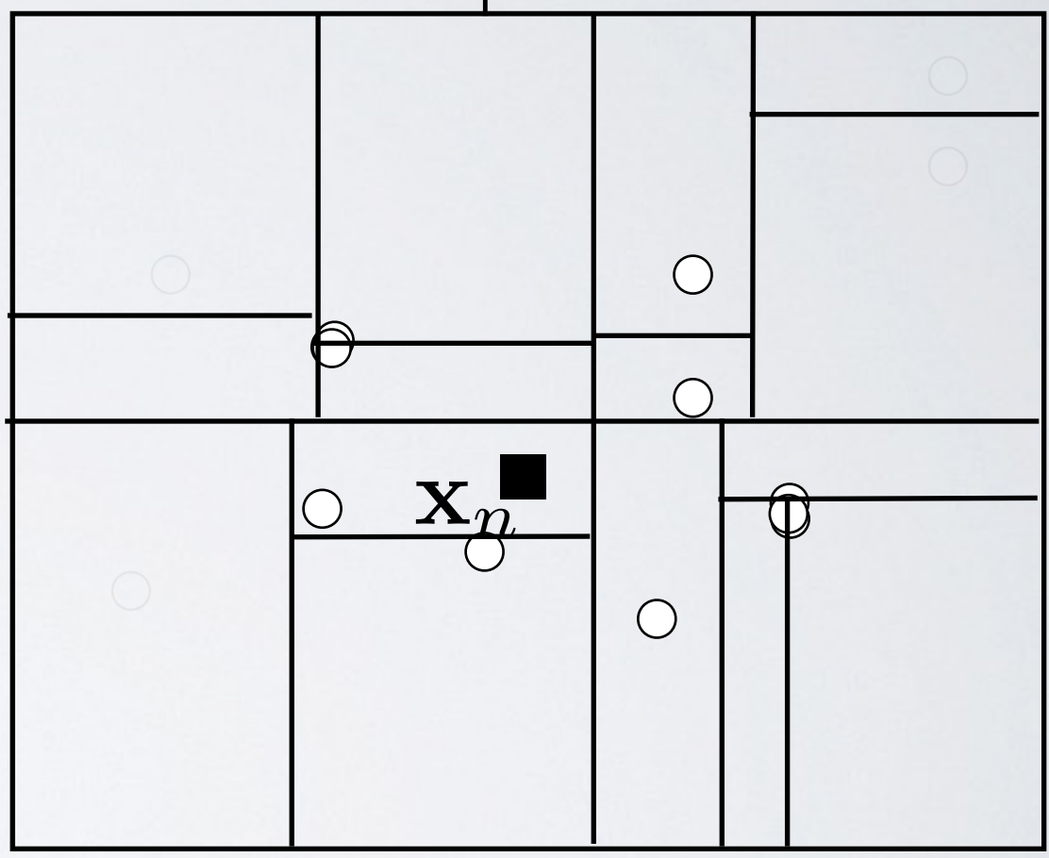


Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.
  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.



- Complexity is usually $\mathcal{O}(N \log N)$.
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually hard to derive.

Barnes-Hut algorithm (Barnes and Hut '86)

☺ Can be applicable to any kind of interaction (Euclidean distances, Gaussian distances, etc).

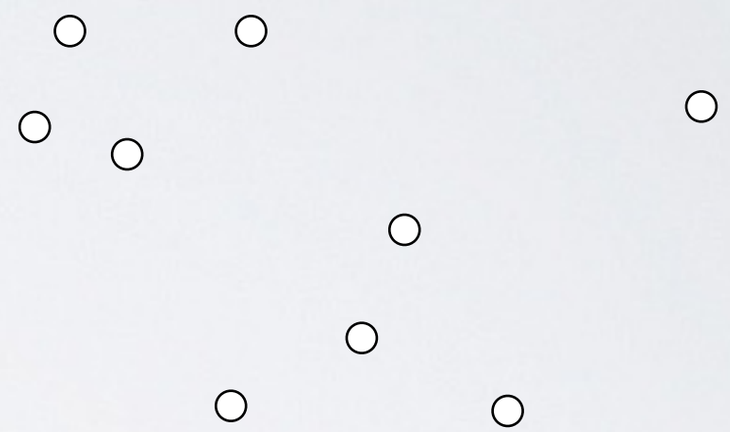
☺ Single parameter to control the trade-off between speed and approximation error.

☹ No clearly defined error bounds.

Was applied to speed up NLE algorithms by Yang et al. (2013) and Maaten (2013).

Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

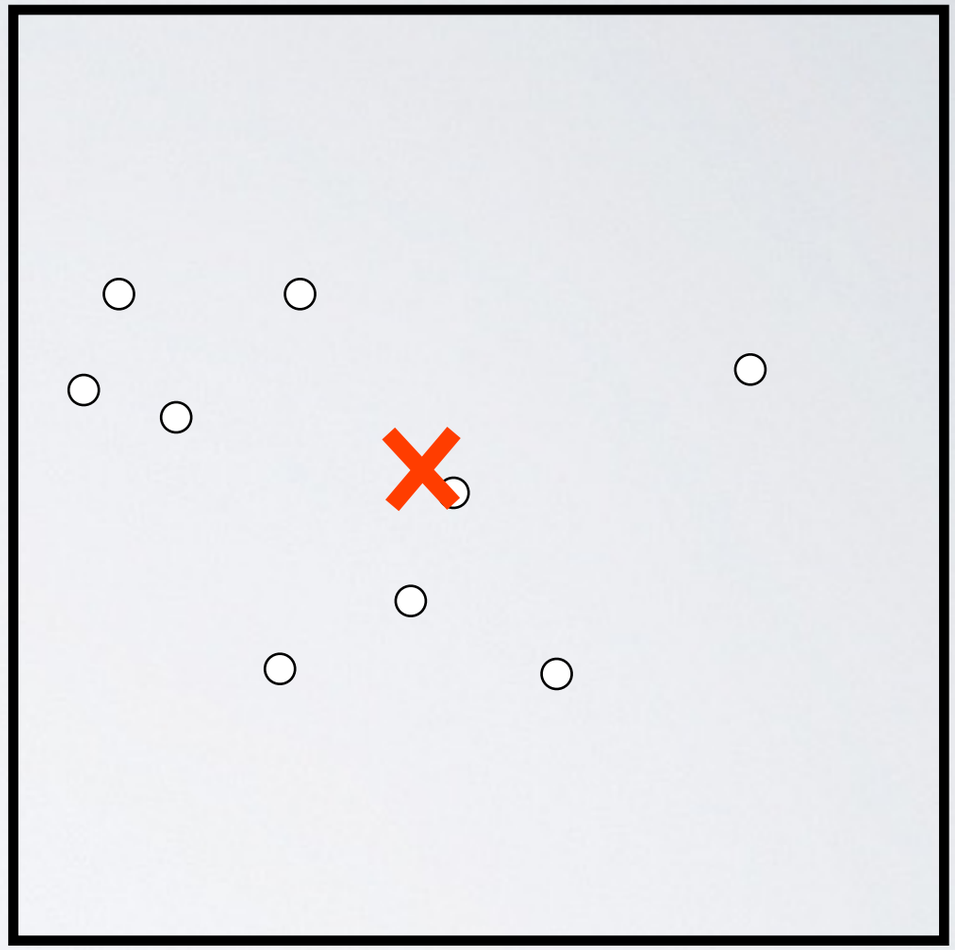


Complexity: $\mathcal{O}(N \log N)$

Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

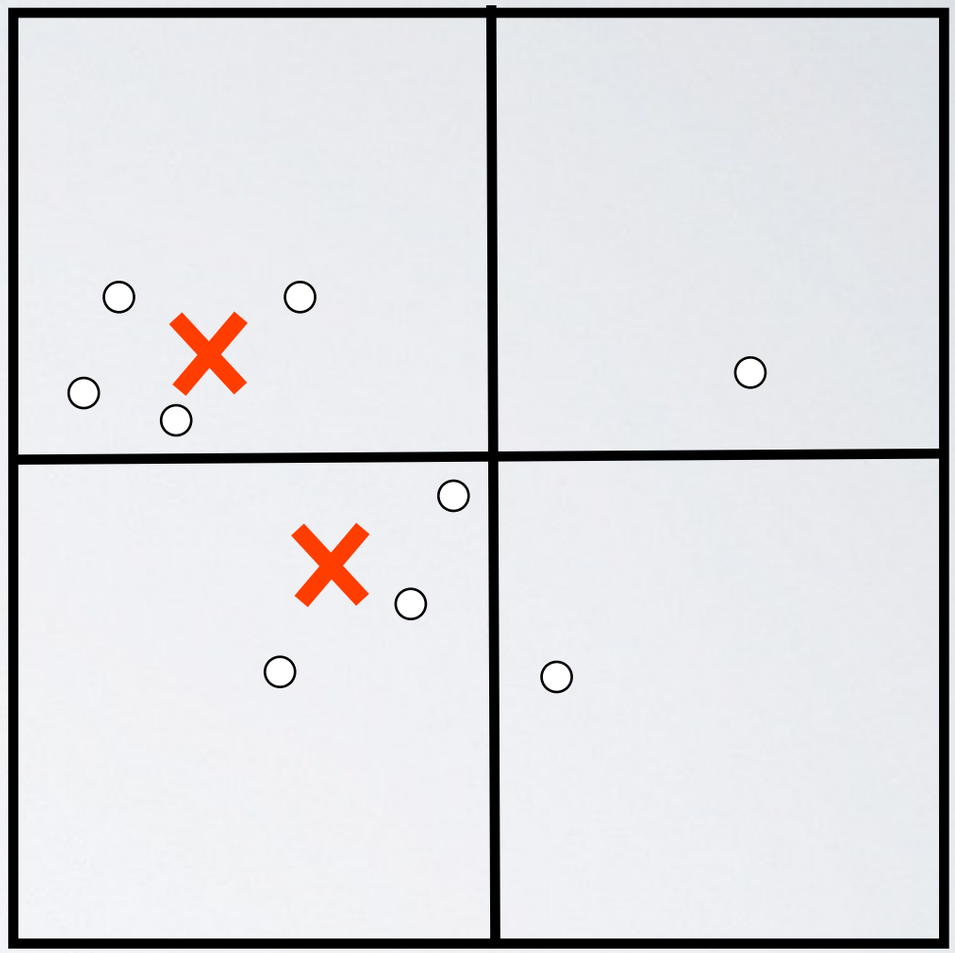
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0, 1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

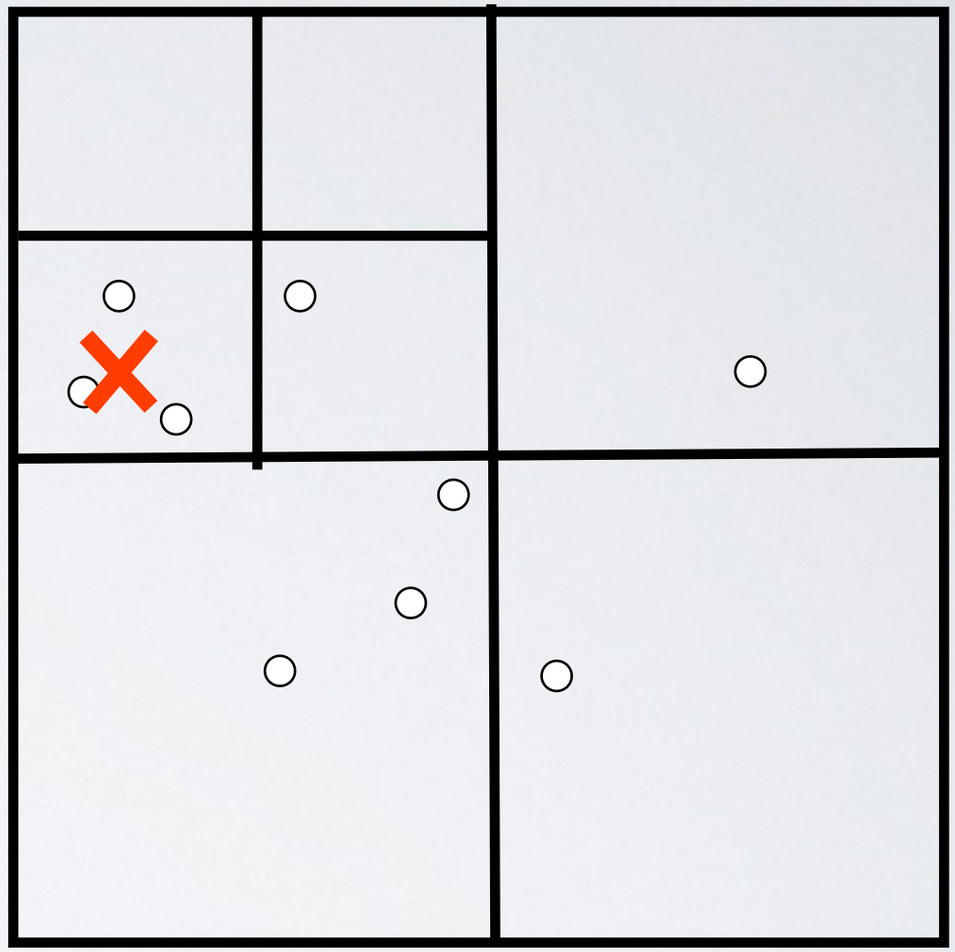
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

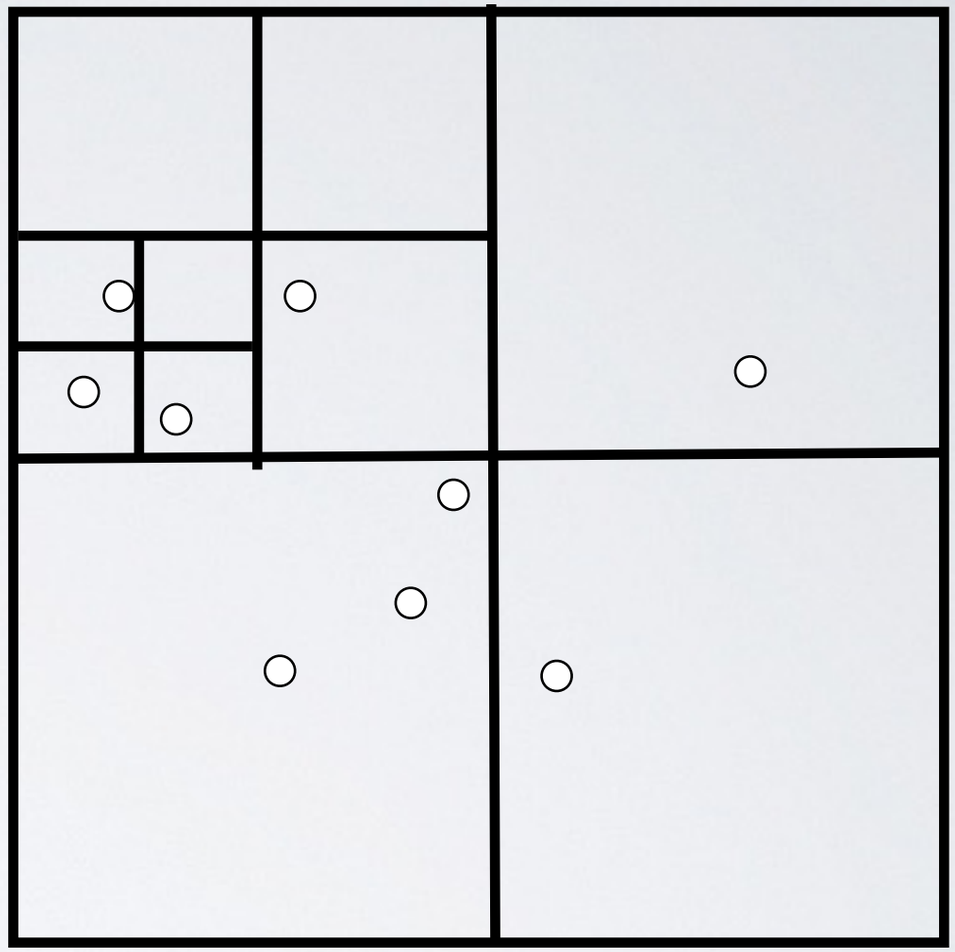
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

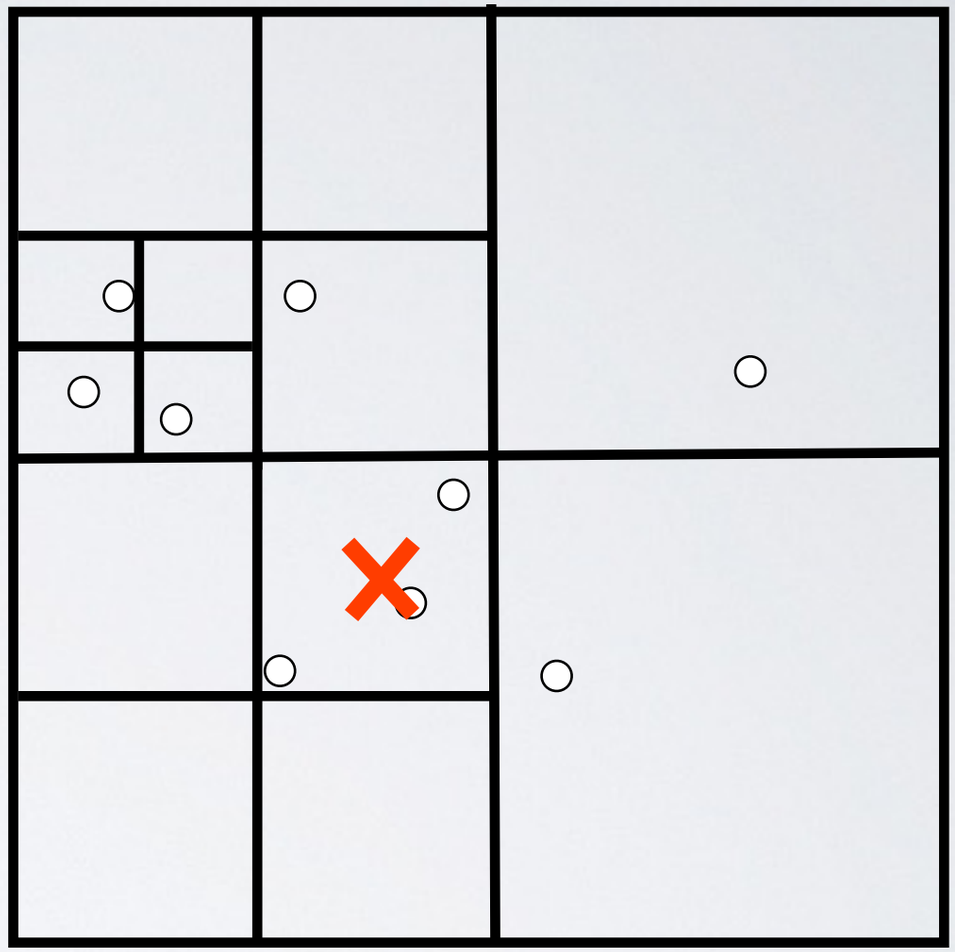
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

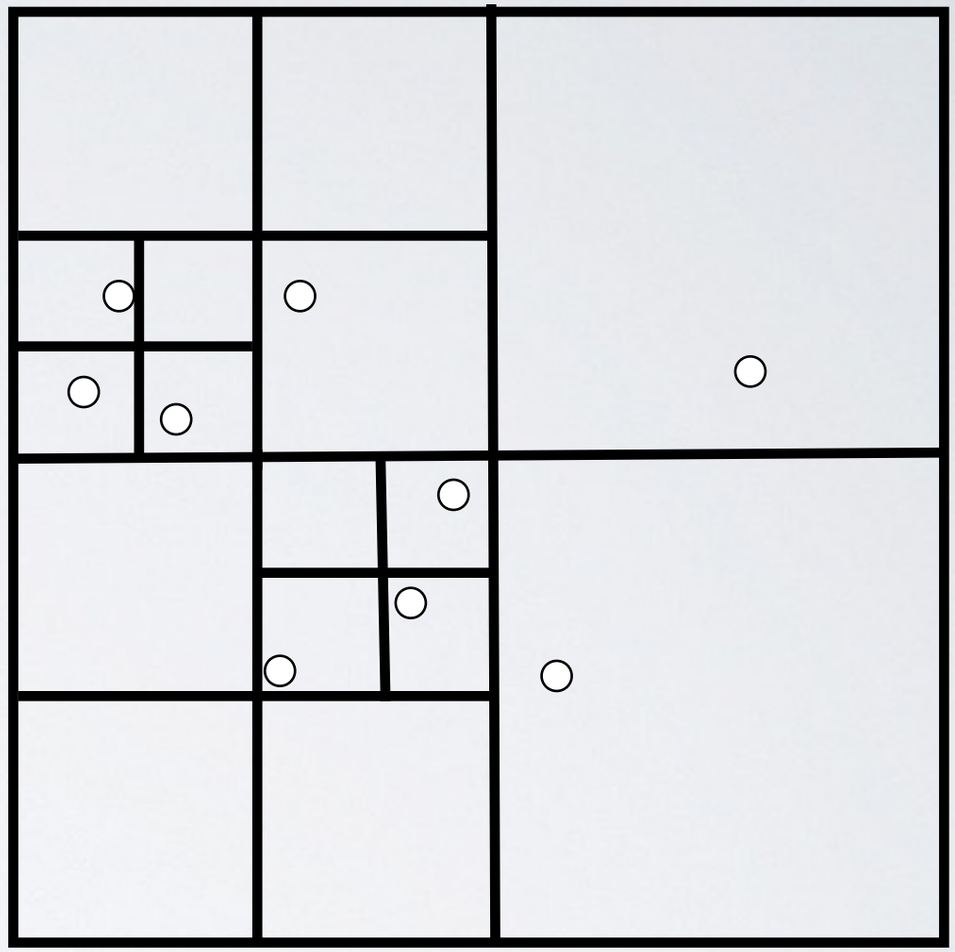
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

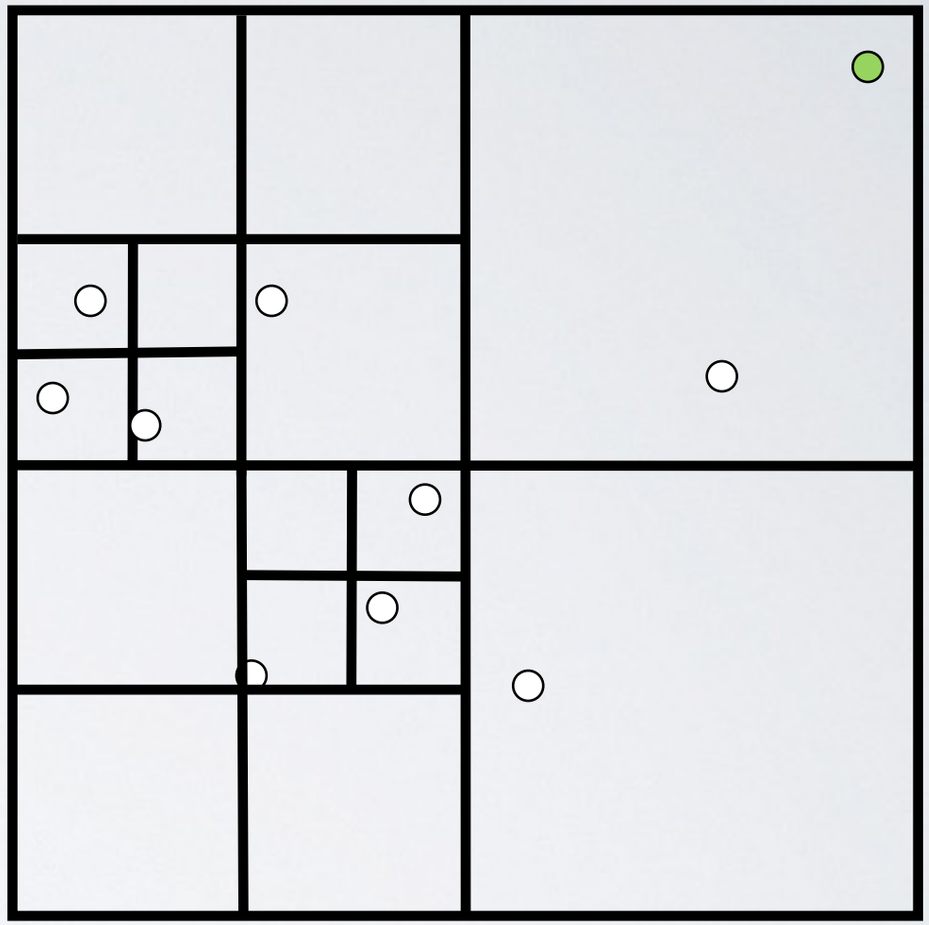
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

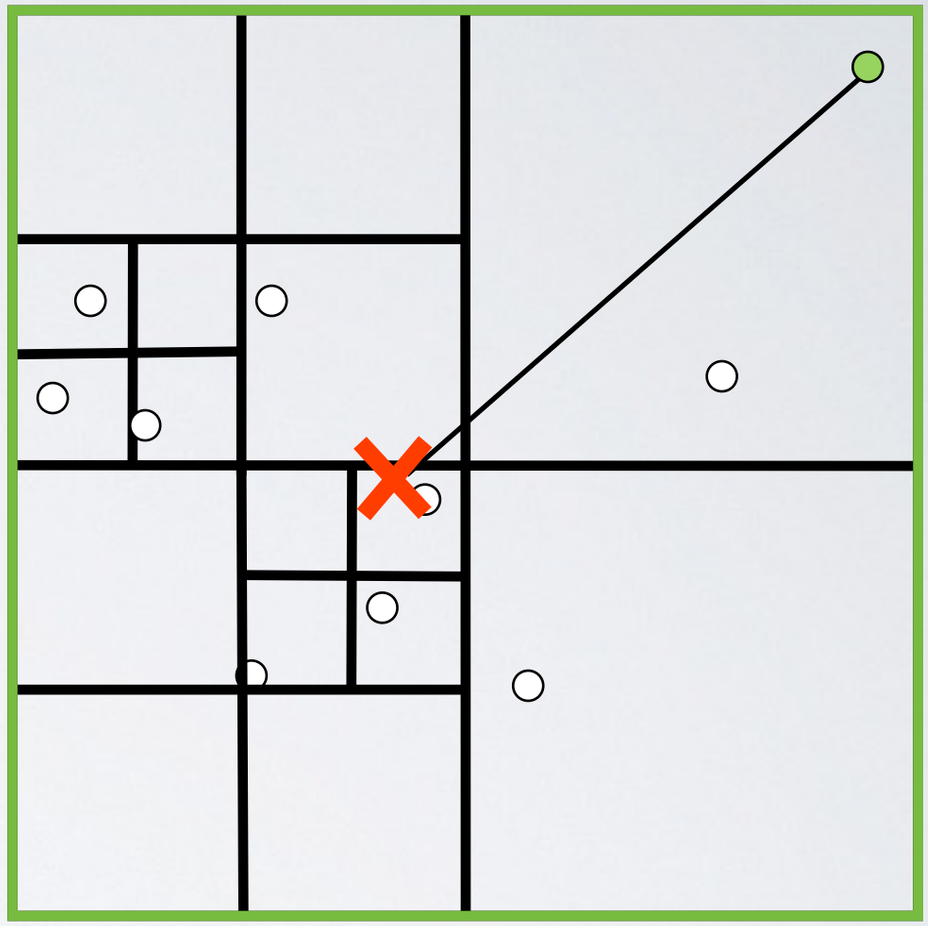
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

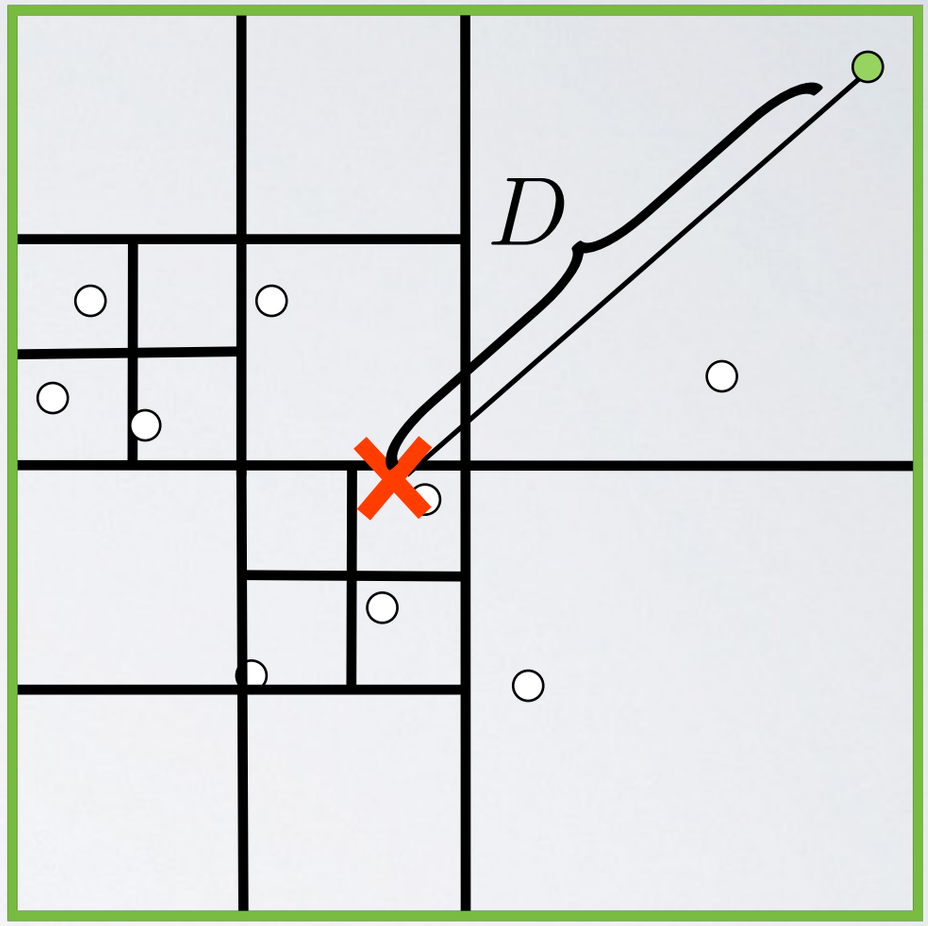
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

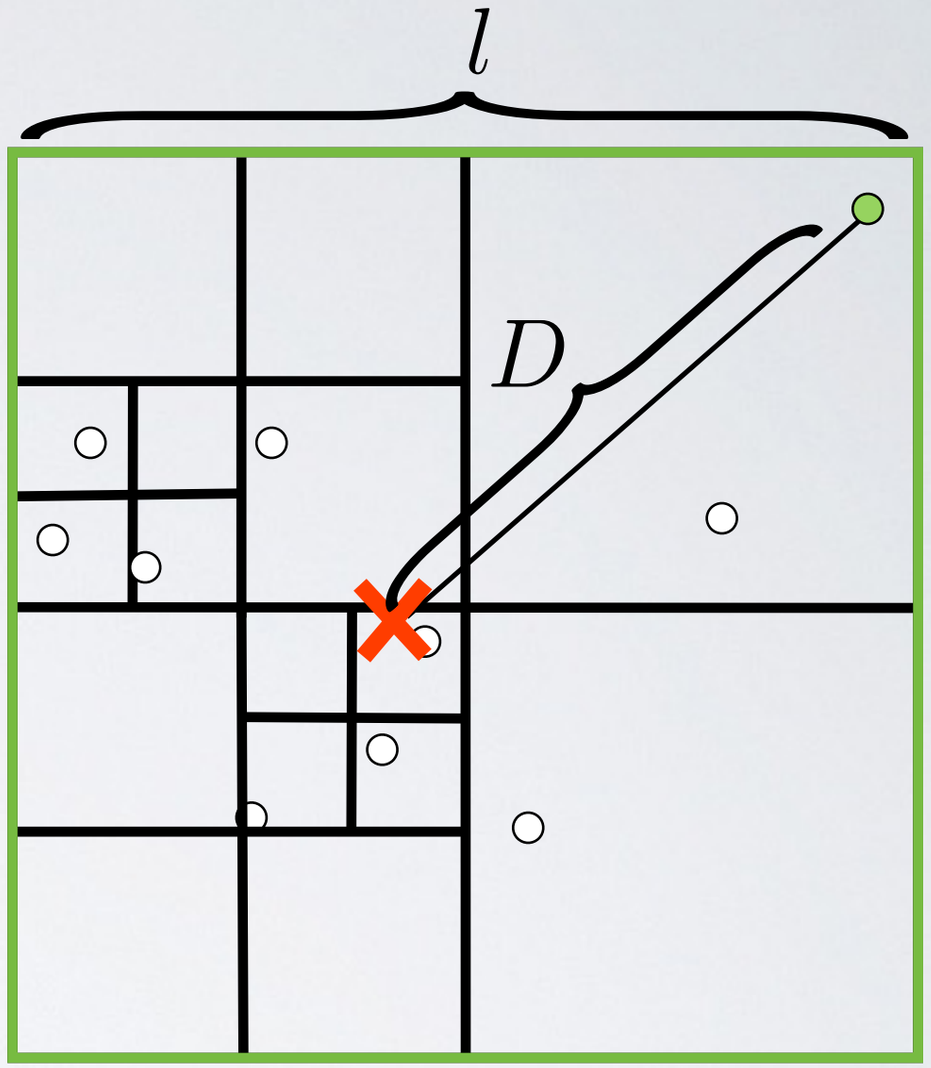
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

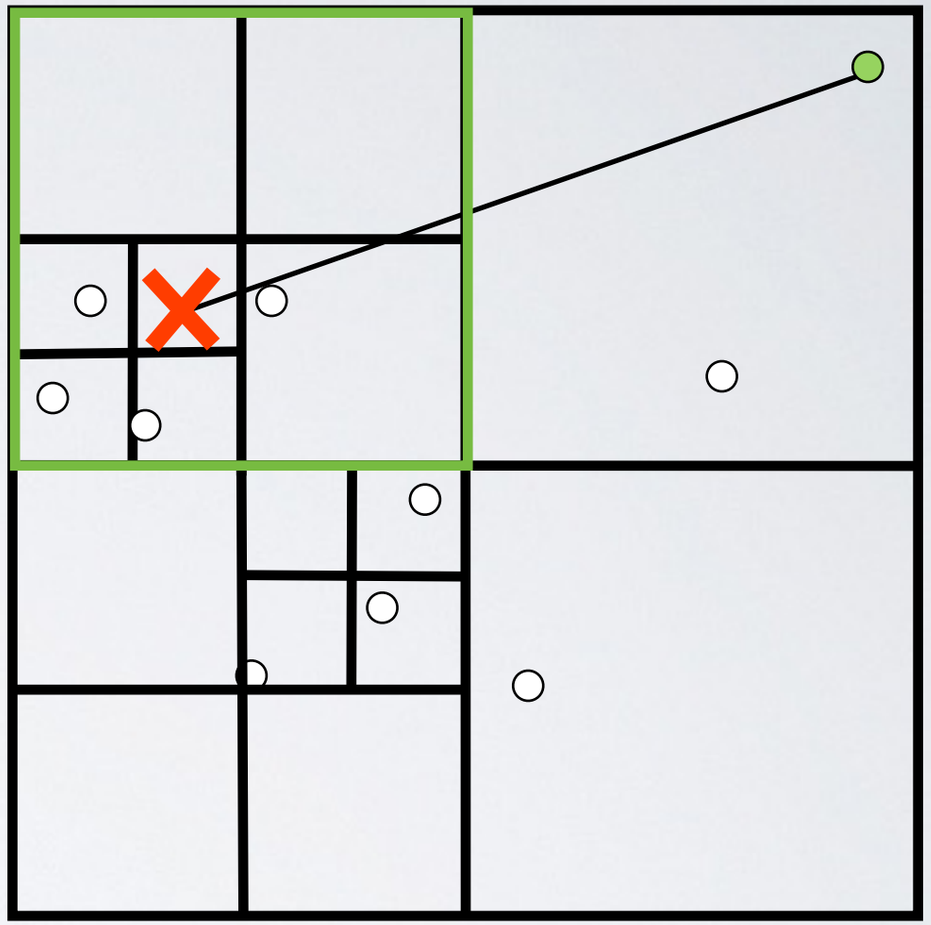
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

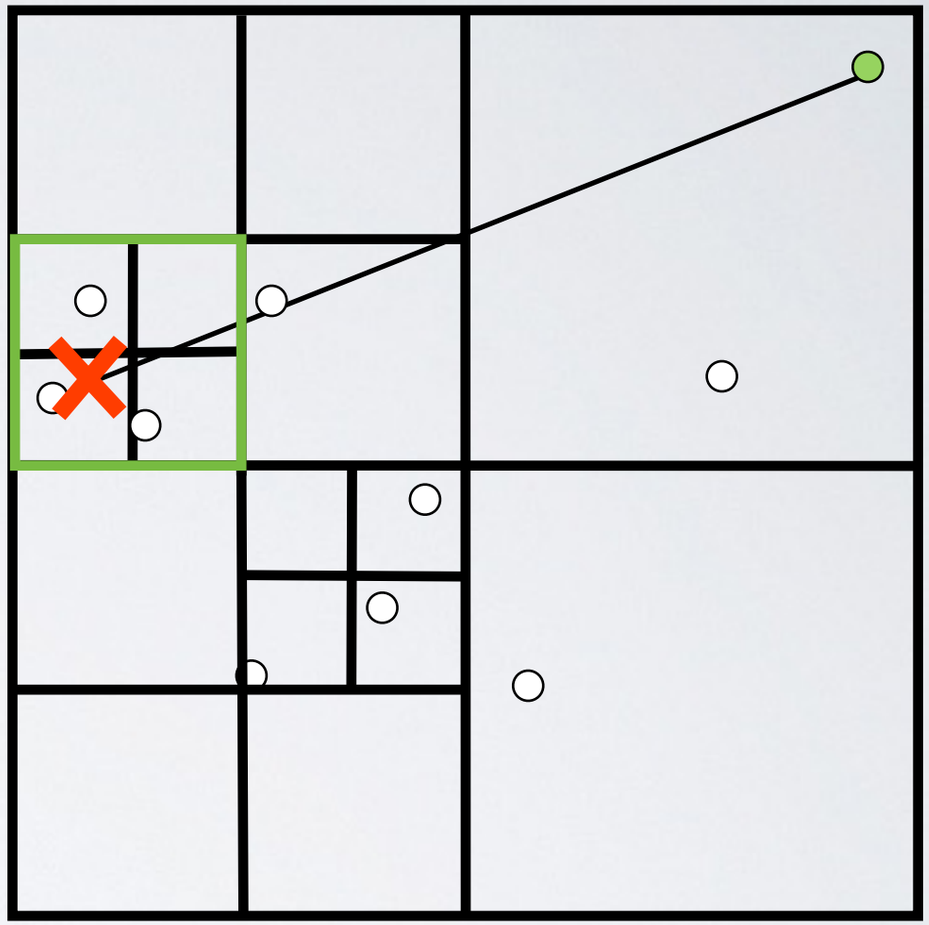
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

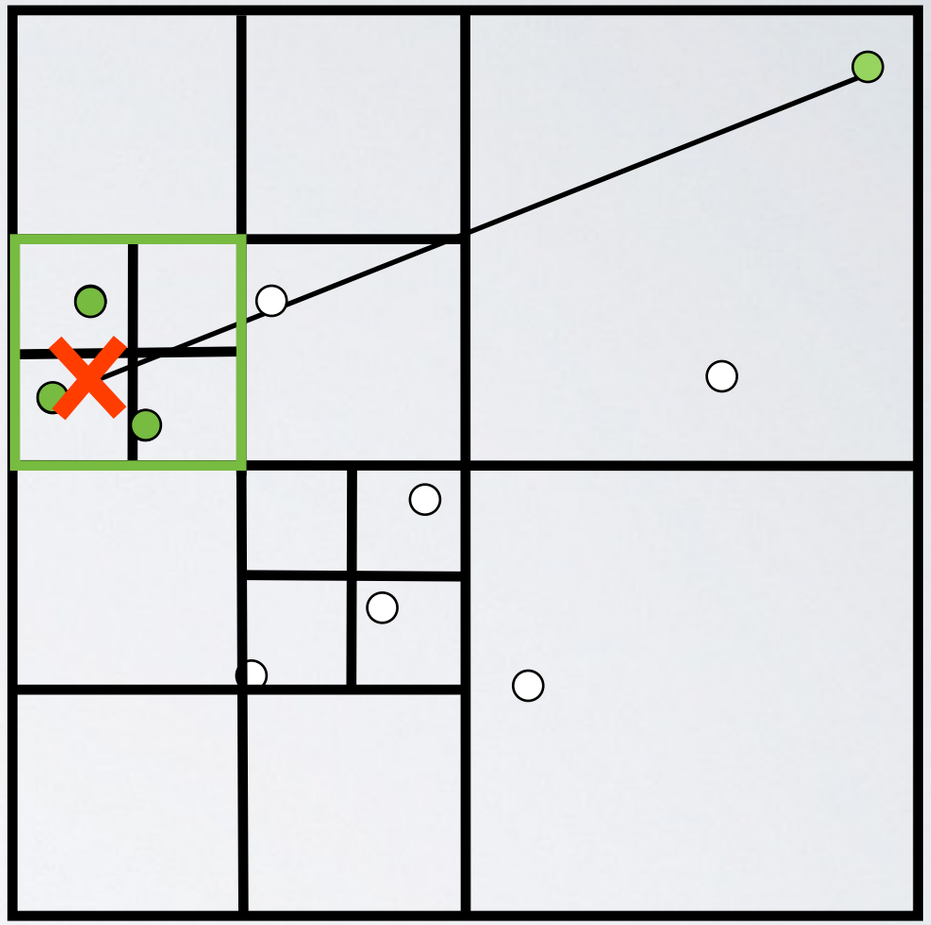
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

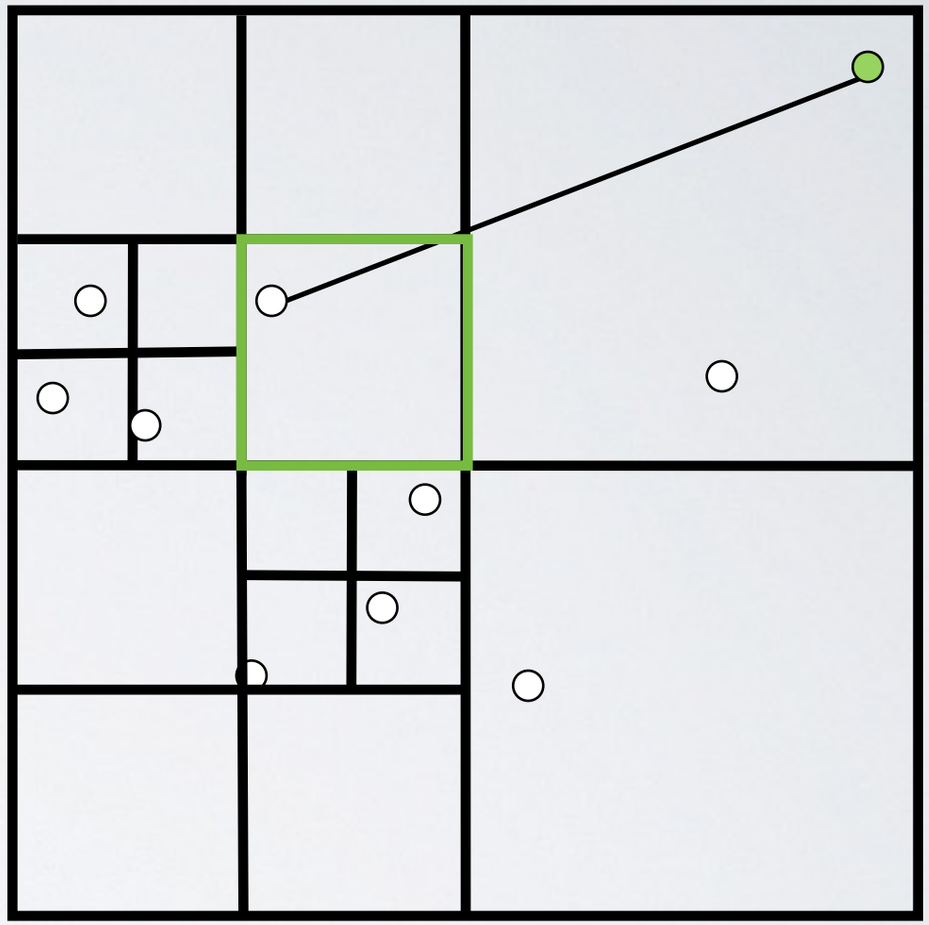
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

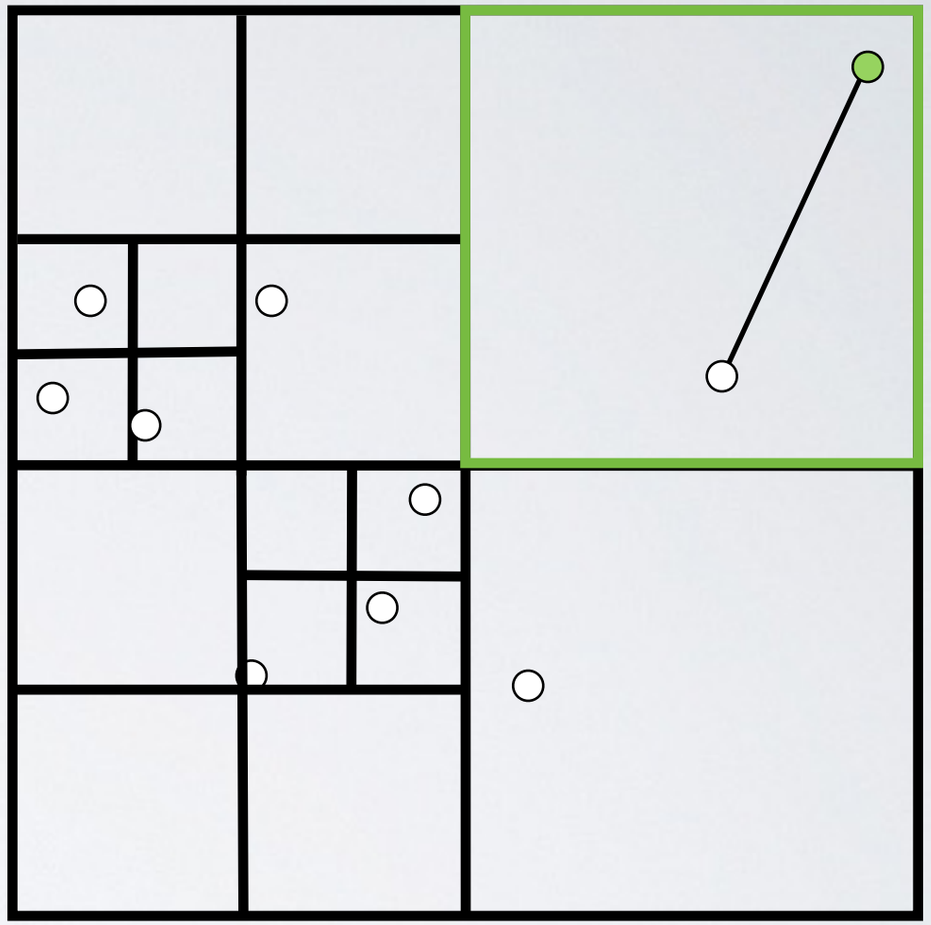
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

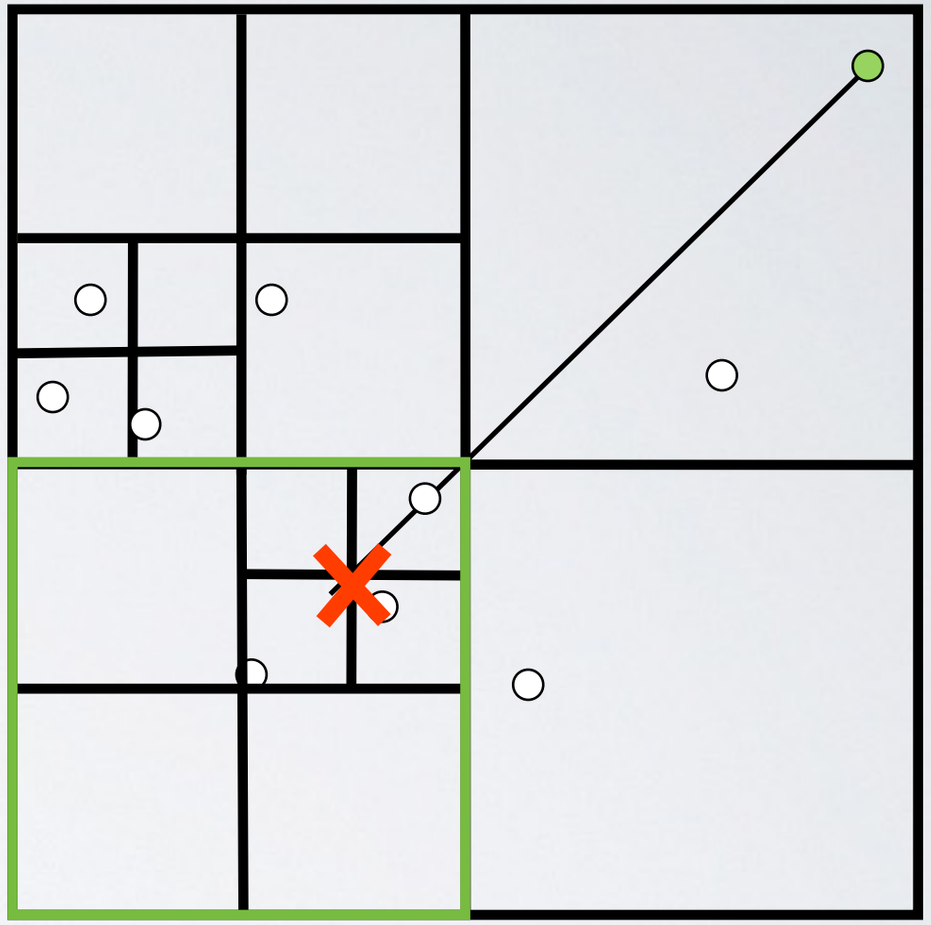
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

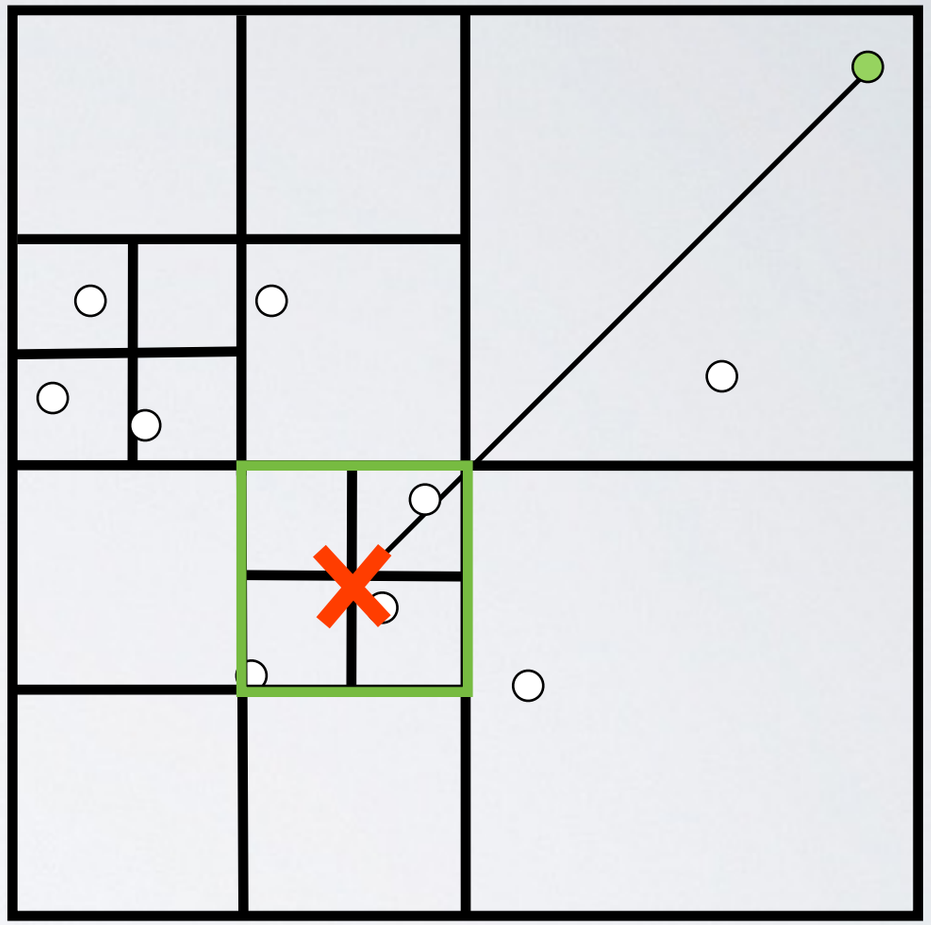
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

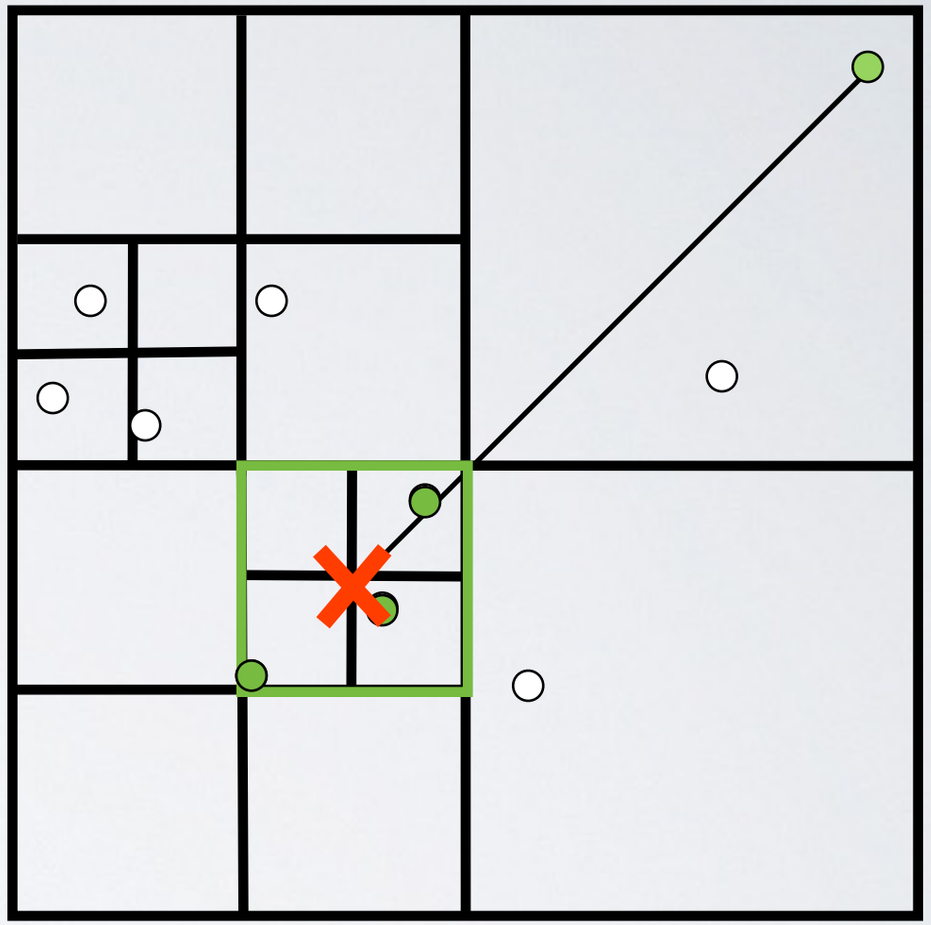
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

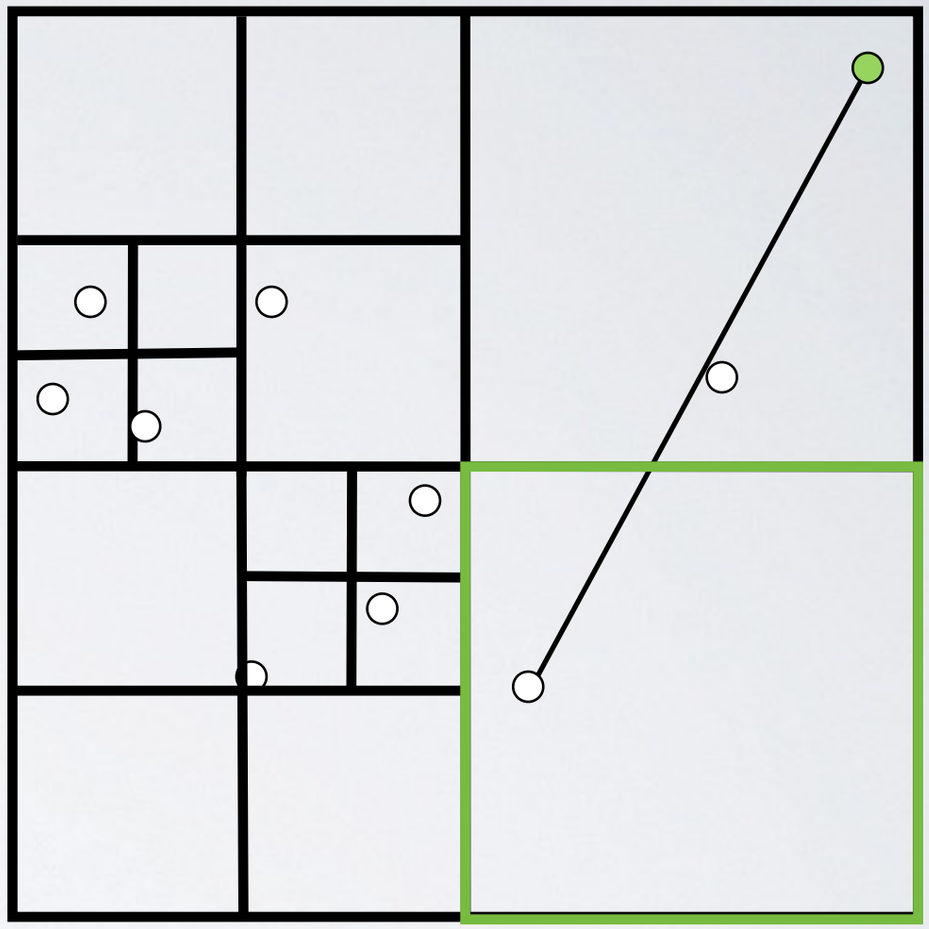
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Fast multipole methods (Greengard and Rokhlin '87)

Properties:

☺ Time complexity $\mathcal{O}(N)$.

☺ Well defined error bounds.

☹ Expansion for each new kernel needs to be derived separately. The performance may vary.

☹ Computational cost grows exponentially with number of dimensions.

Fast multipole methods (Greengard and Rokhlin '87)

Approximate the interactions of the form:

$$Q(\mathbf{x}_n) = \sum_{m=1}^{N} q_m K(\|(\mathbf{x}_n - \mathbf{x}_m)/\sigma\|^2)$$

The idea is to do a series expansion of the kernel K, such that the sum decouples over $\mathbf{x}_n$ and $\mathbf{x}_m$:

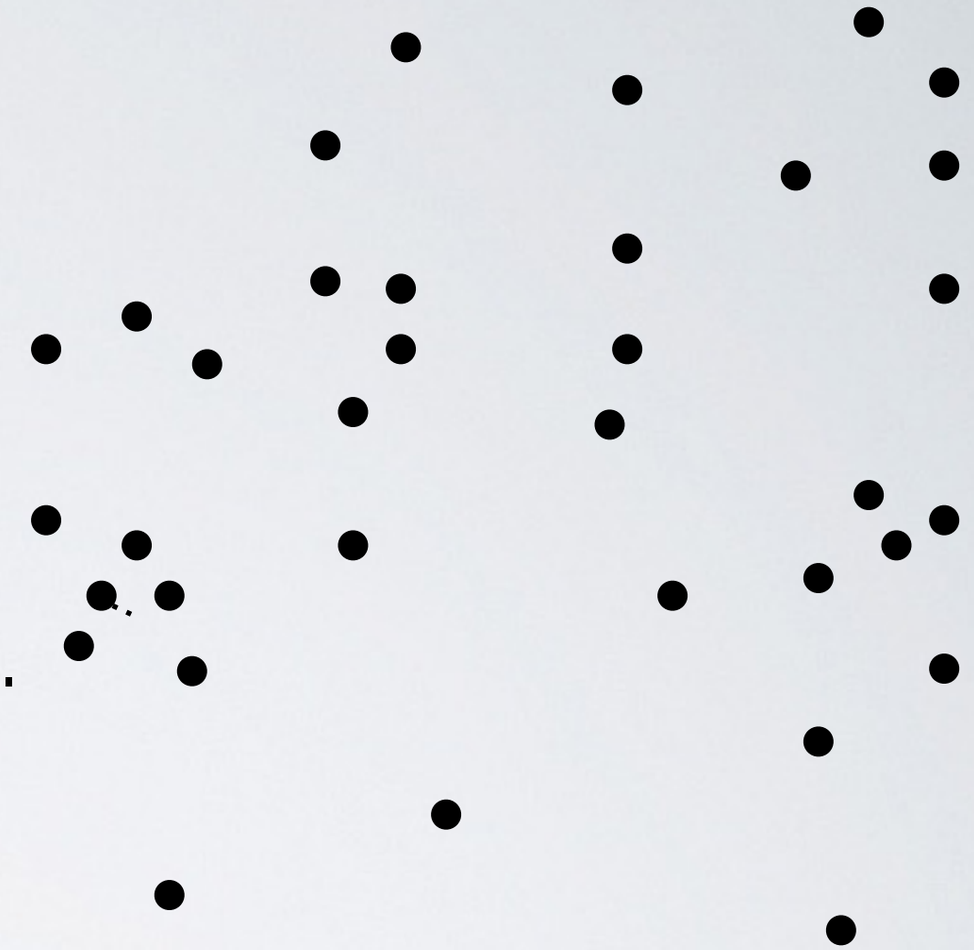
$$K(\|(\mathbf{x}_n - \mathbf{x}_m)/\sigma\|^2) = \sum_{\boldsymbol{\alpha} \geq 0} f_{\boldsymbol{\alpha}}(\mathbf{x}_n)\, g_{\boldsymbol{\alpha}}(\mathbf{x}_m)$$

using multi-index notation $\boldsymbol{\alpha} \geq 0 \Rightarrow \alpha_1, \ldots, \alpha_d \geq 0$

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

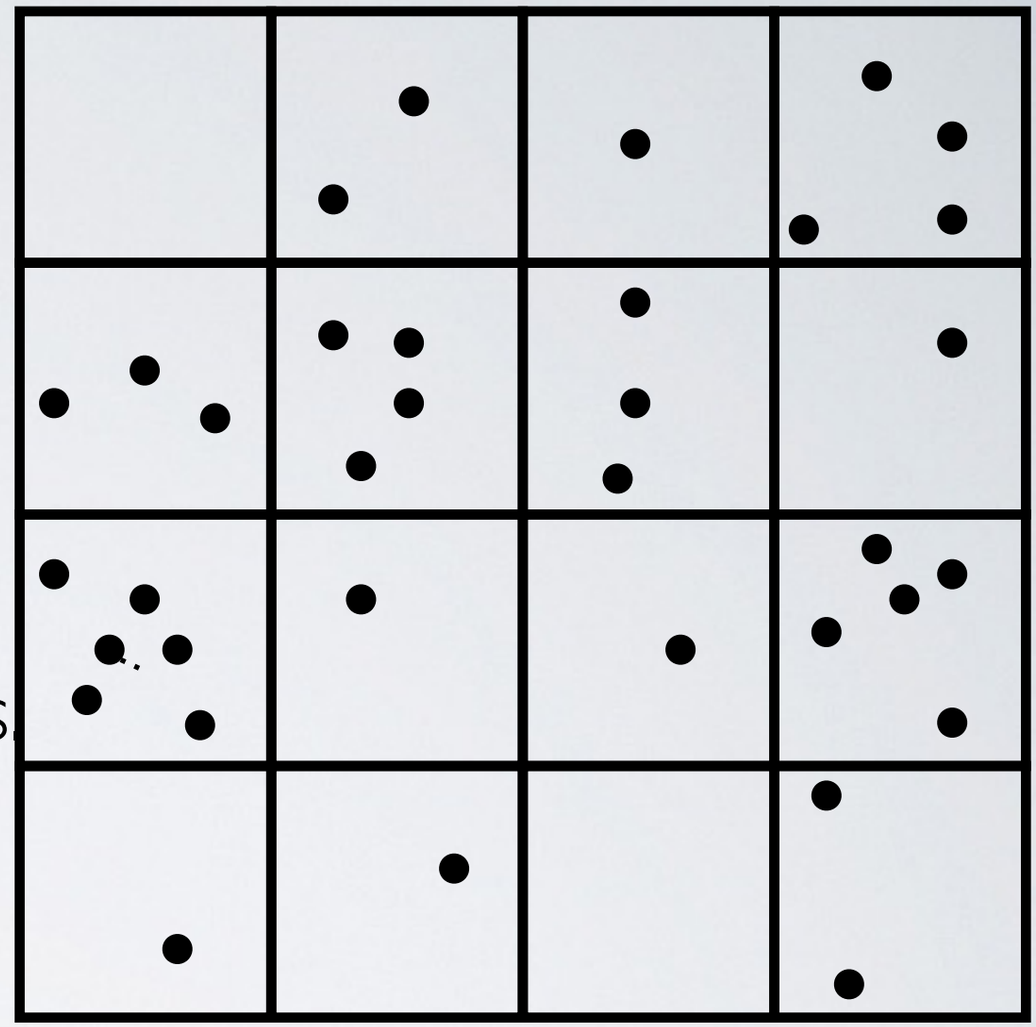
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

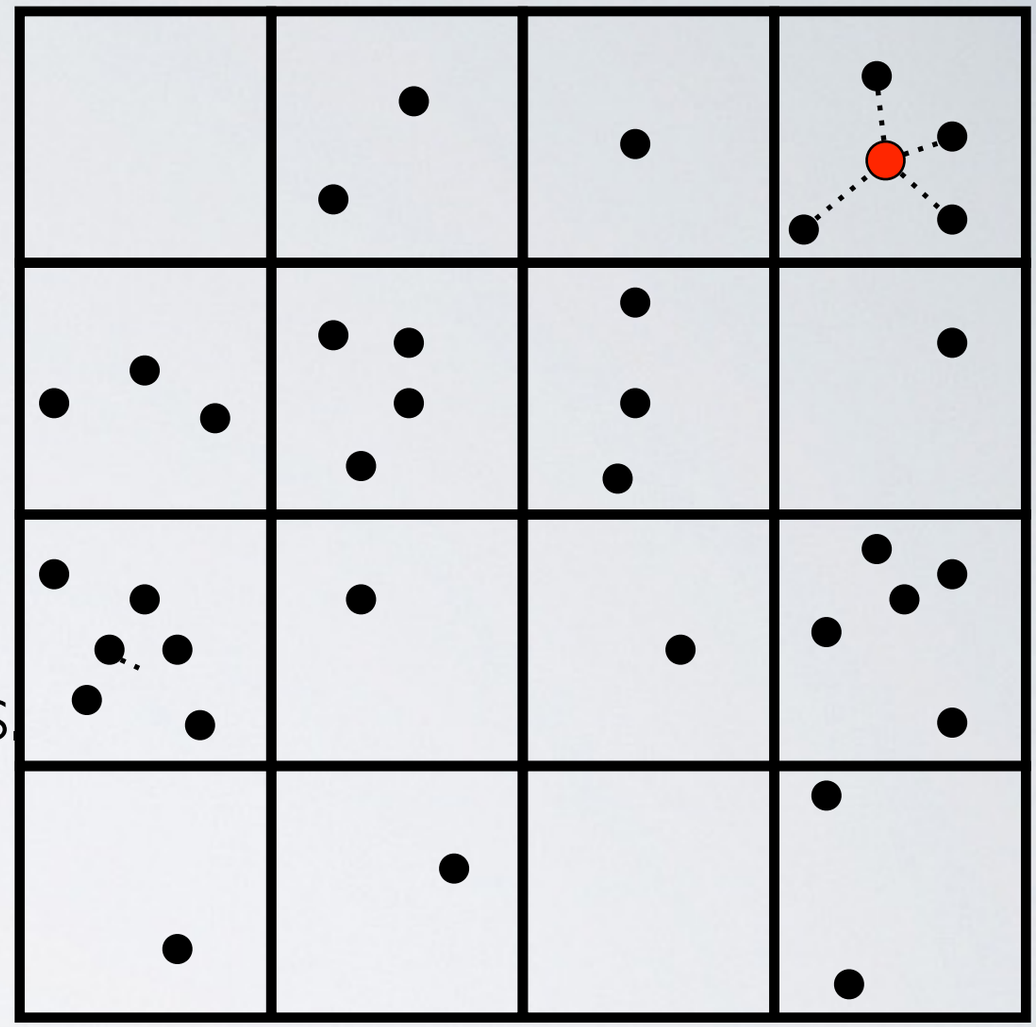
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

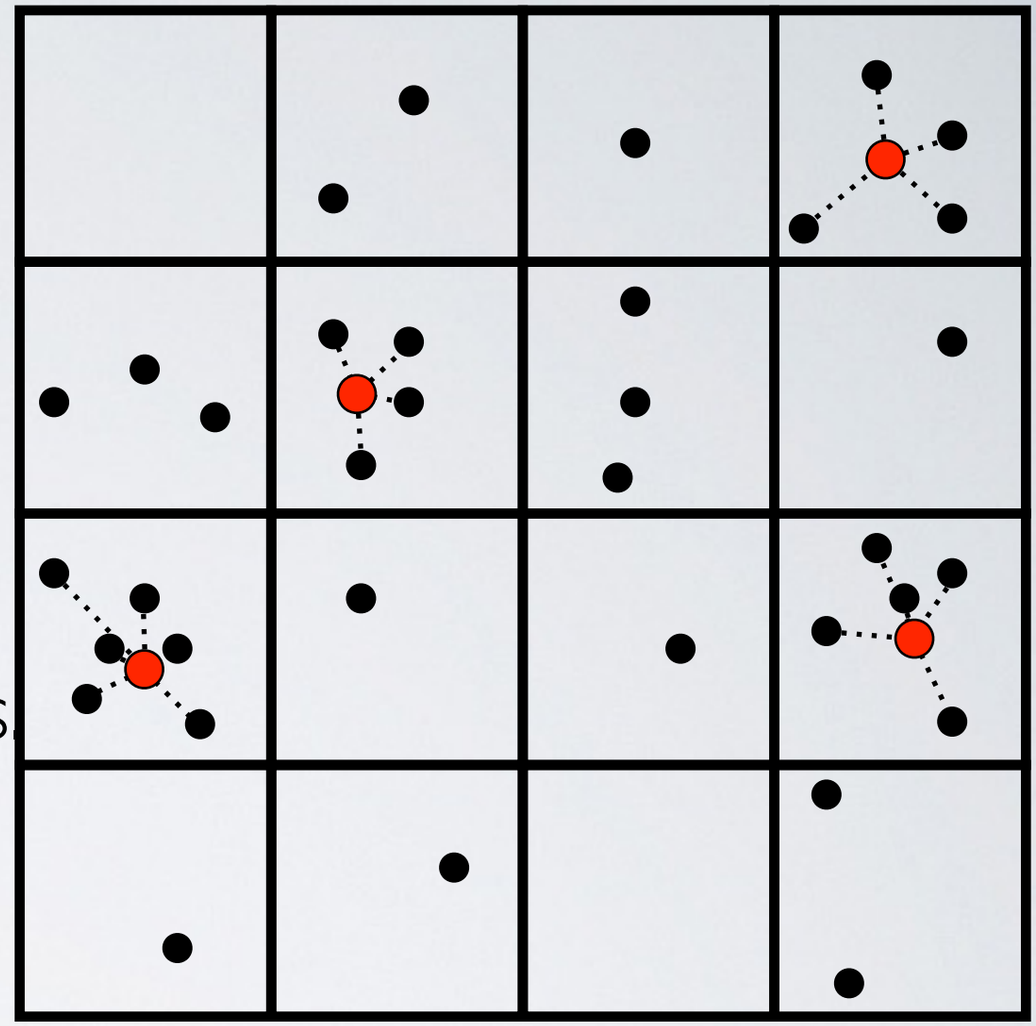
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

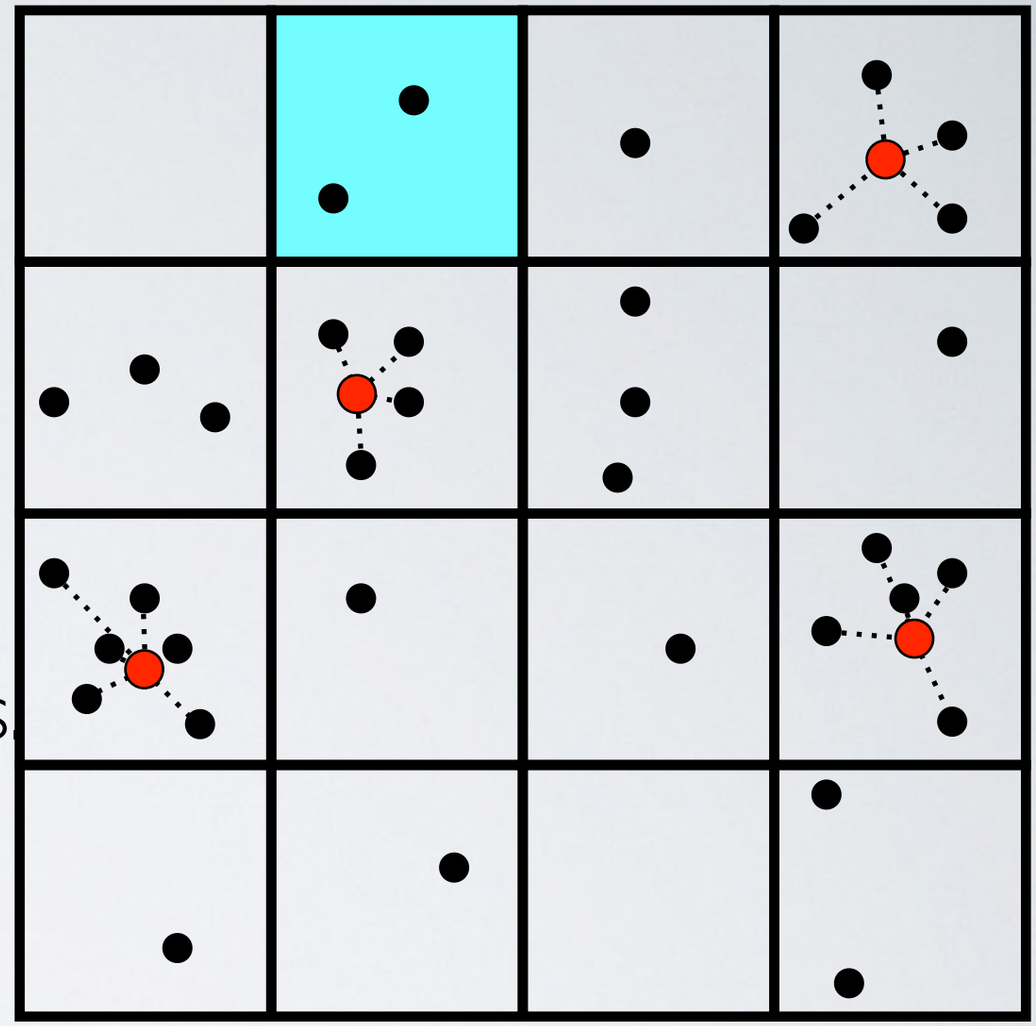
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

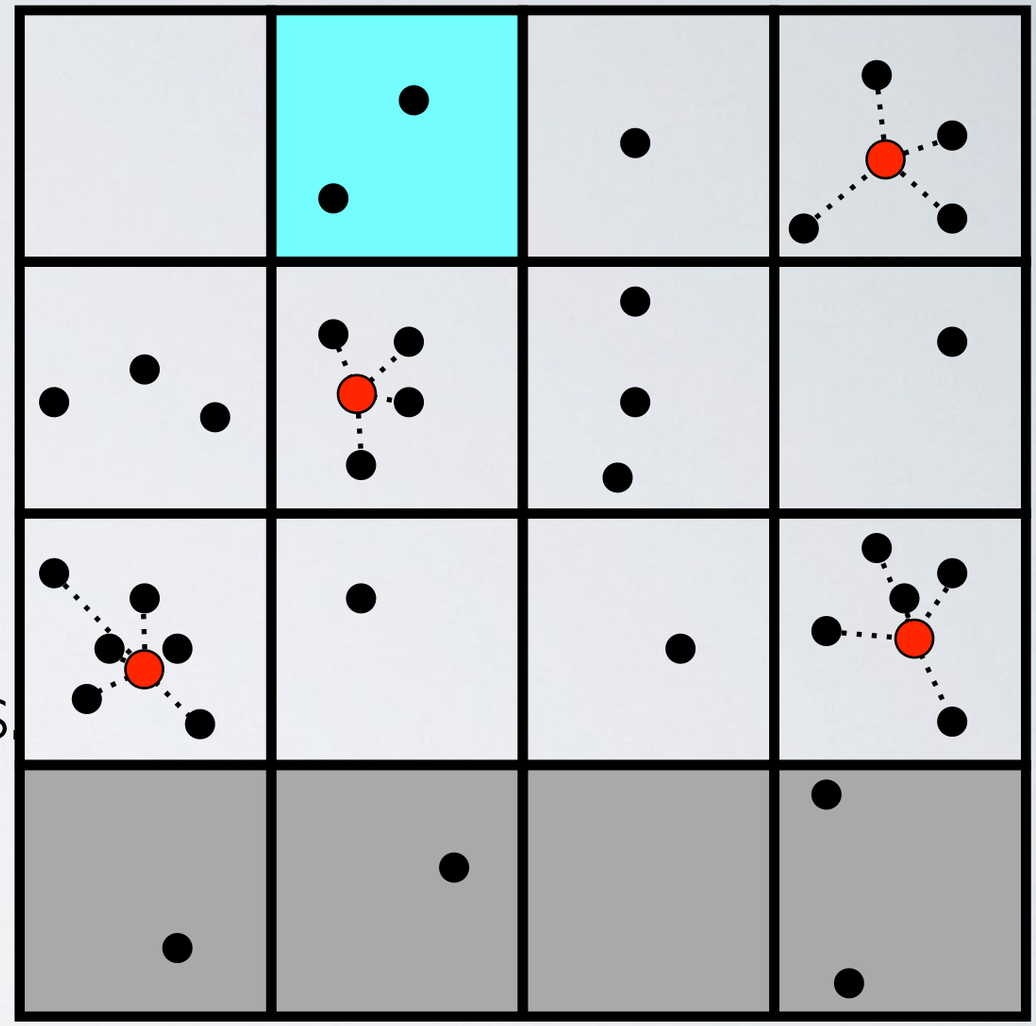
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

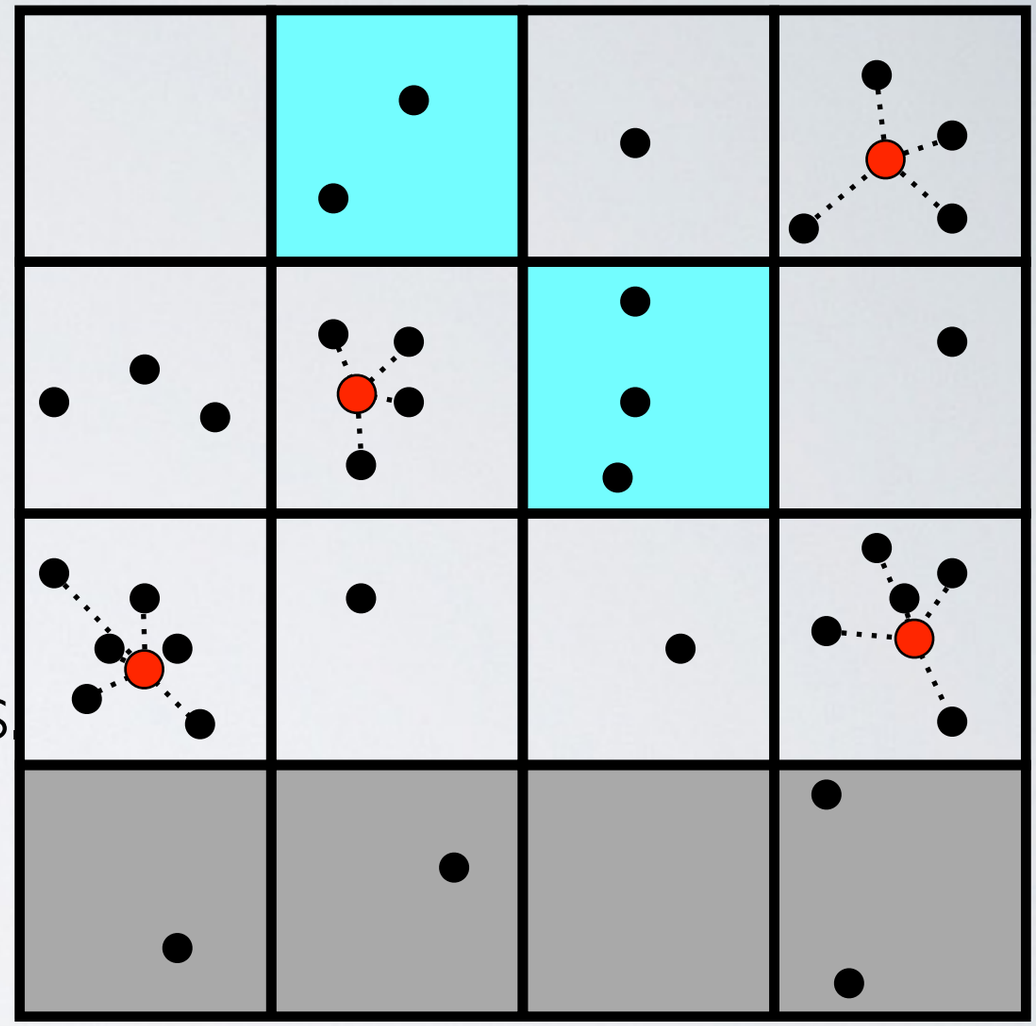
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

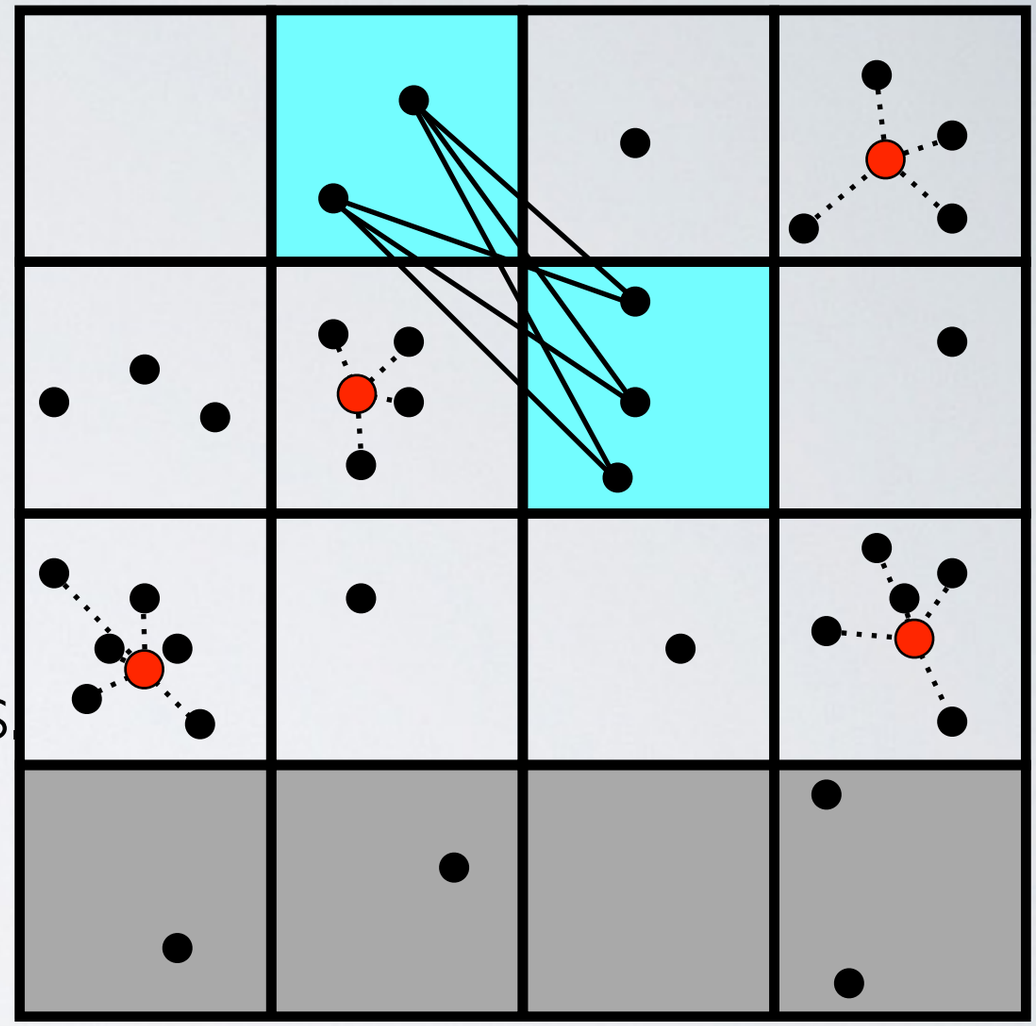
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

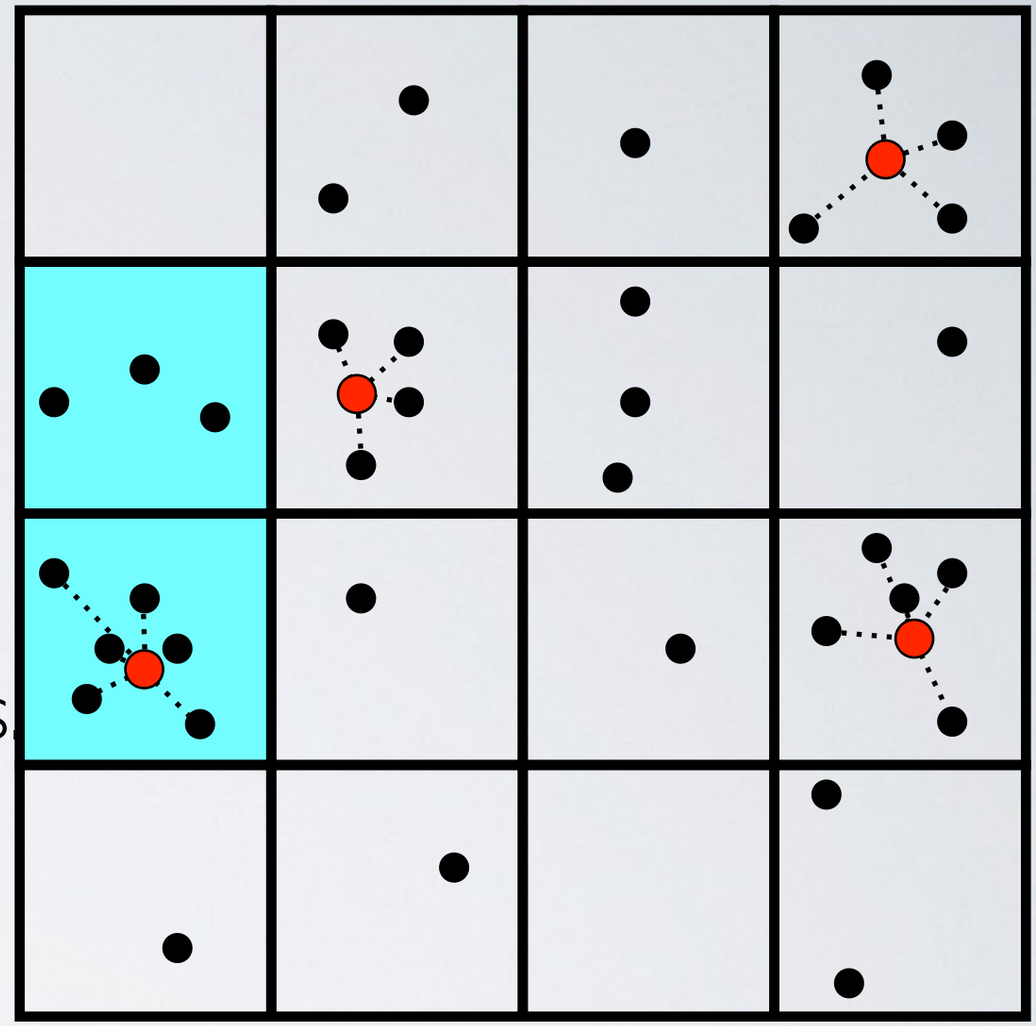
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

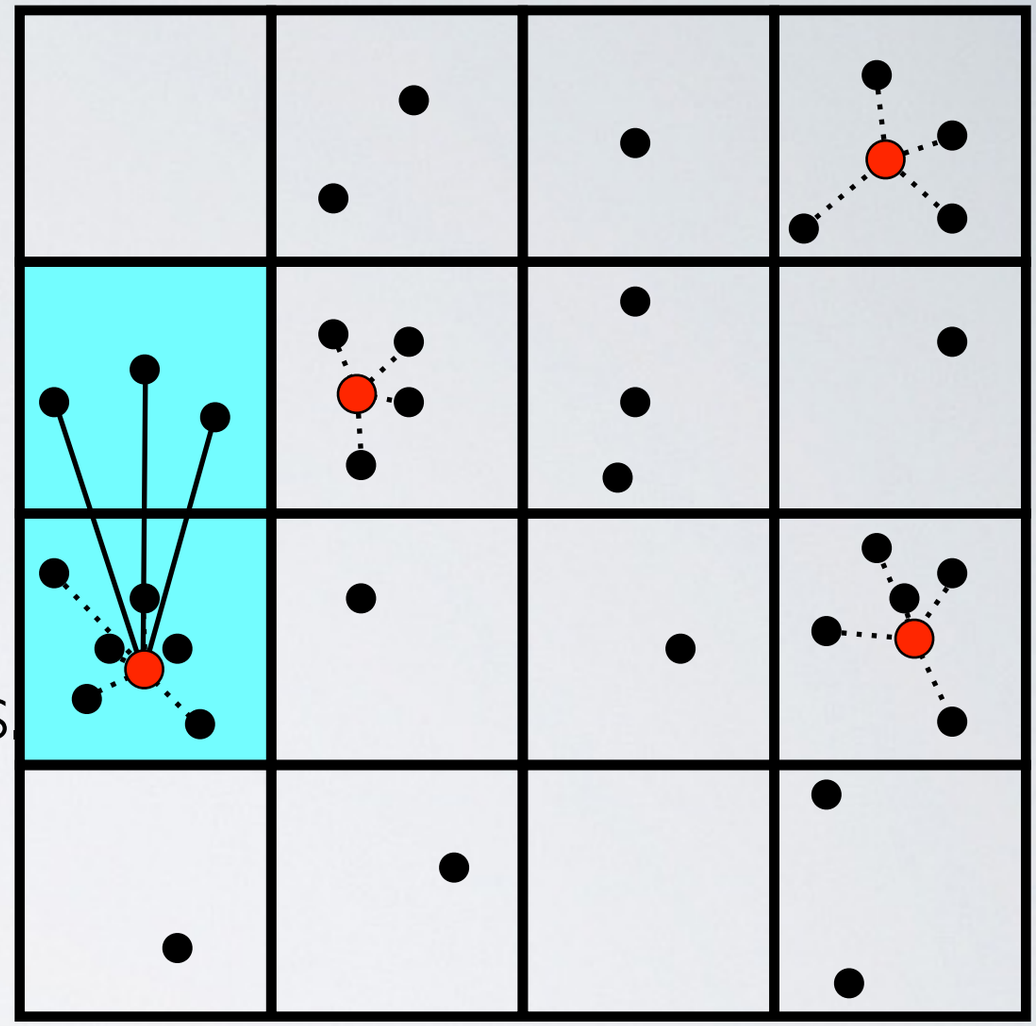
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

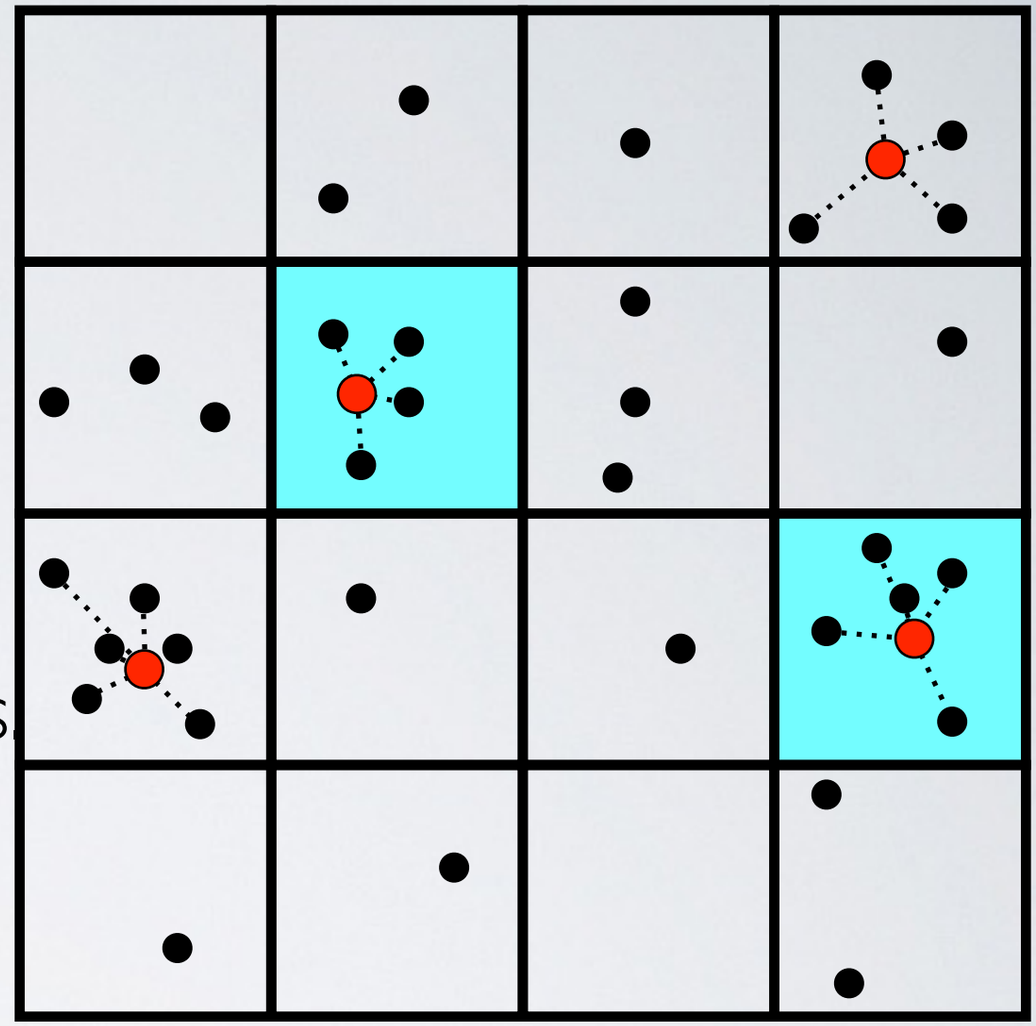
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

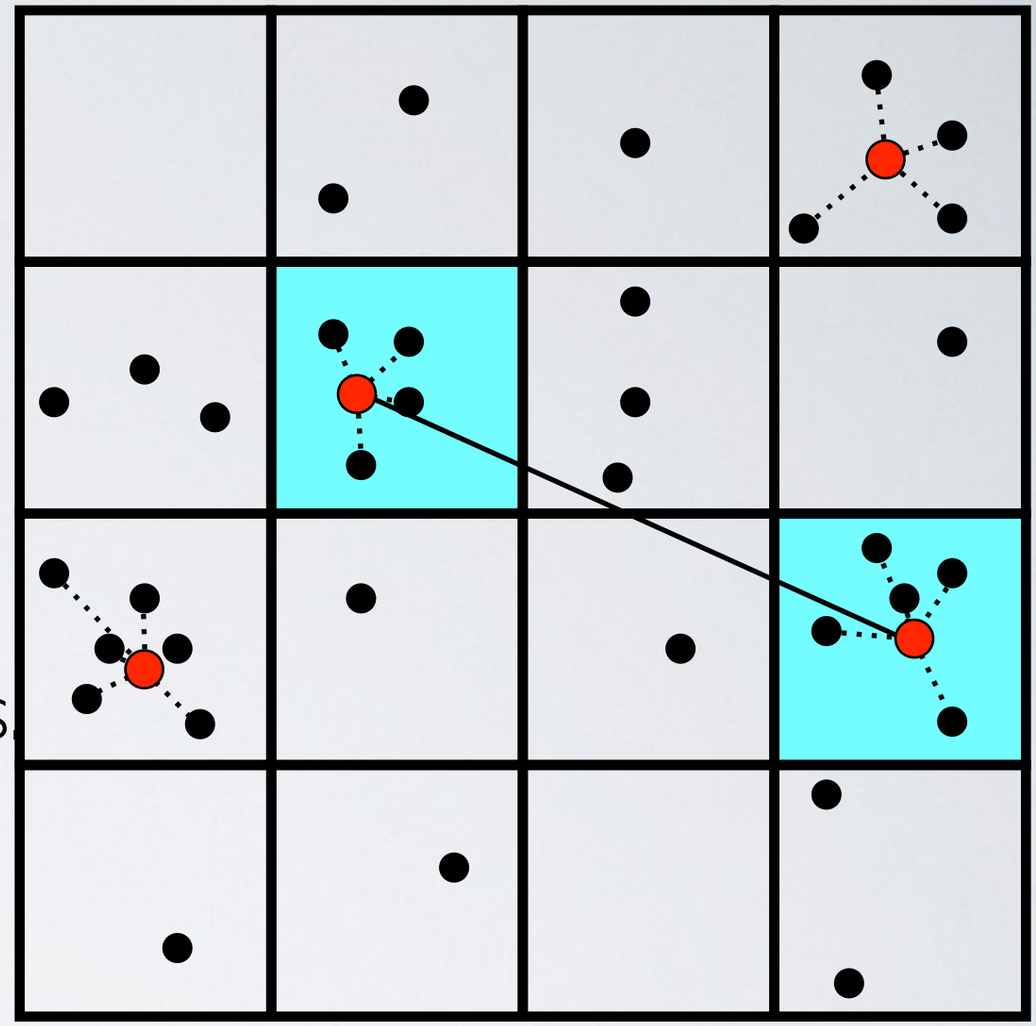
1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Application of *N*-Body to NLE

- We can approximate the following interaction with *N*-Body methods

$$S(\mathbf{x}_n) = \sum_{m=1}^{N} K(||\mathbf{x}_n - \mathbf{x}_m||^2) \qquad S^x(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{x}_m K(||\mathbf{x}_n - \mathbf{x}_m||^2)$$

- The objective function and the gradient of EE:

$$E_{EE}(\mathbf{X}) = \sum_{n,m=1}^{N} w_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \lambda \sum_{n=1}^{N} S(\mathbf{x}_n)$$

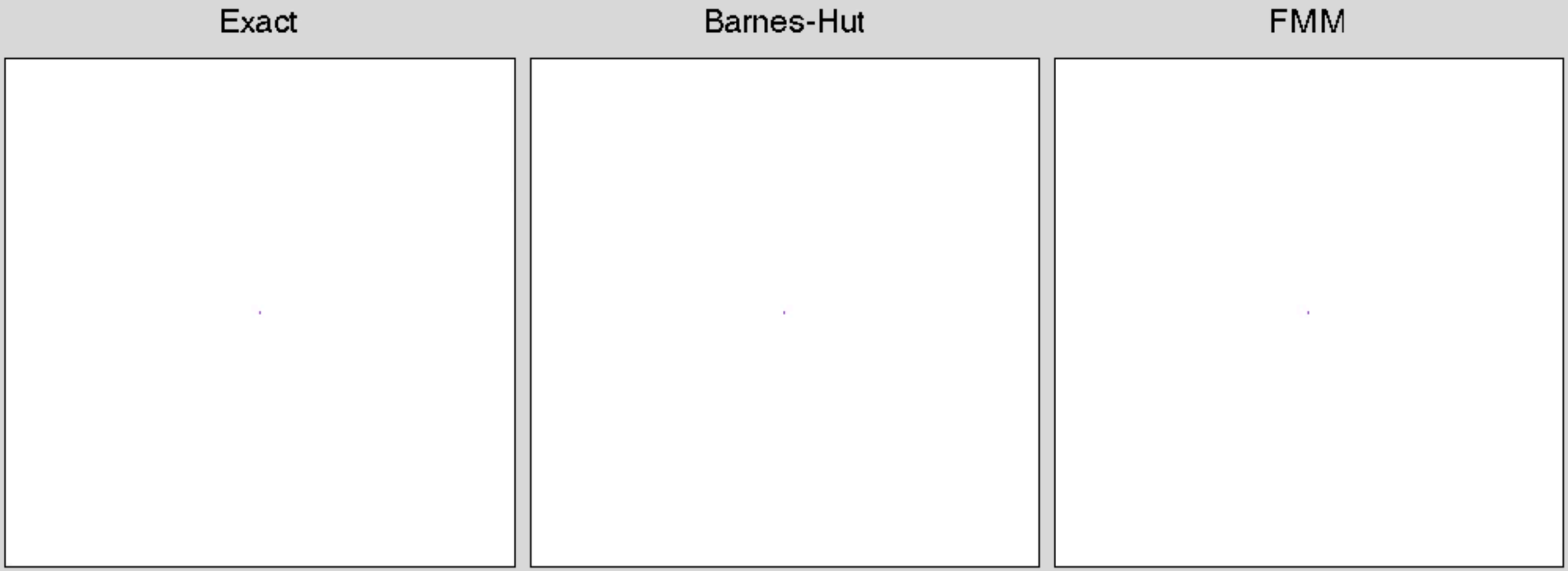
$$G_{EE}(\mathbf{X}) = 4\mathbf{X}\mathbf{L} - 4\lambda\mathbf{X}\,\mathrm{diag}\left(S(\mathbf{X})\right) + 4\lambda S^x(\mathbf{X})$$

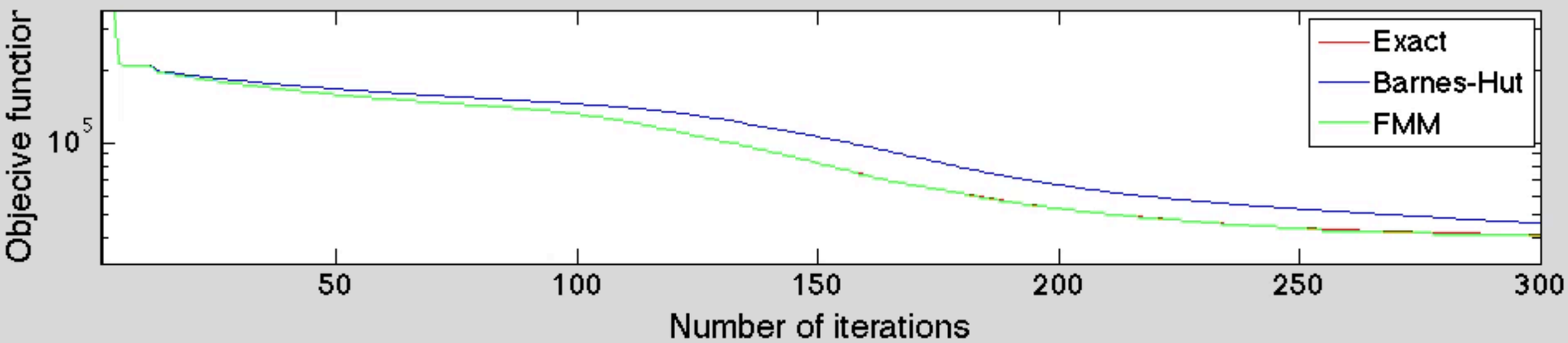
- Given $S(\mathbf{x}_n)$and $S^x(\mathbf{x}_n)$, each term is can be computed in $\mathcal{O}(N)$.

- Objective function and the gradient of other NLE methods can be defined analogously.

Experiments: 60 000 handwritten digits

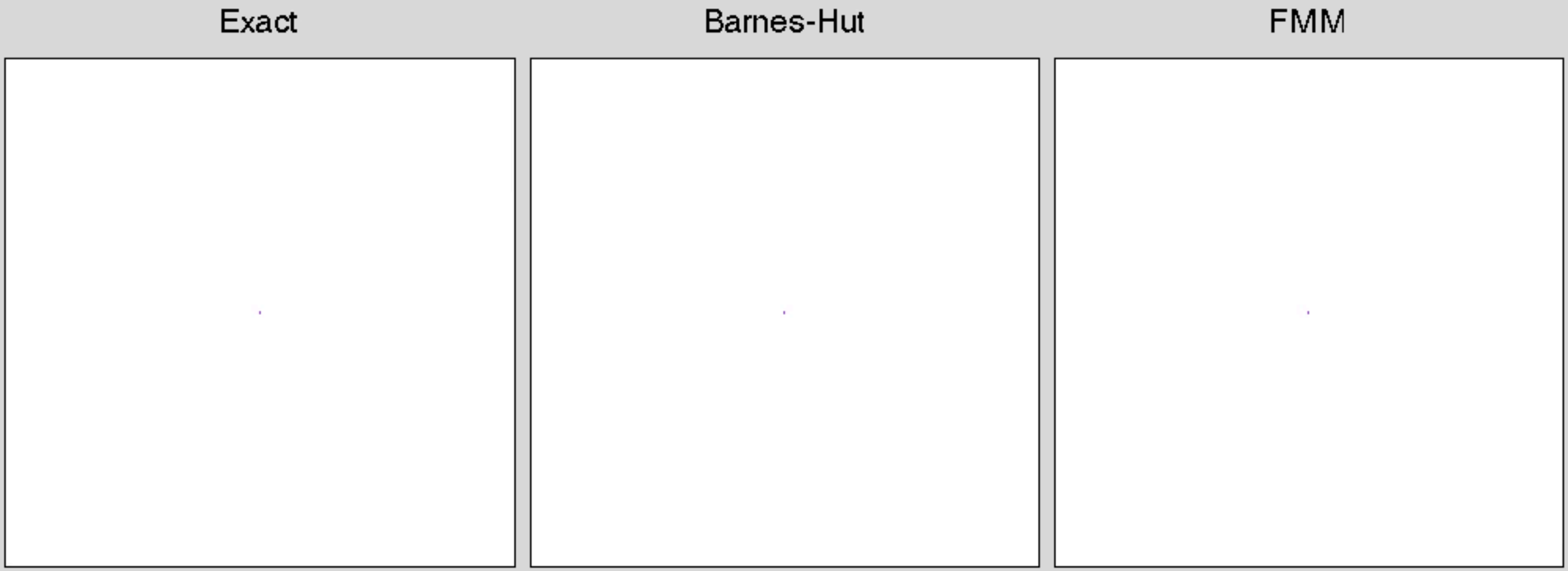
All methods show similar decrease in the objective function *per iteration.*

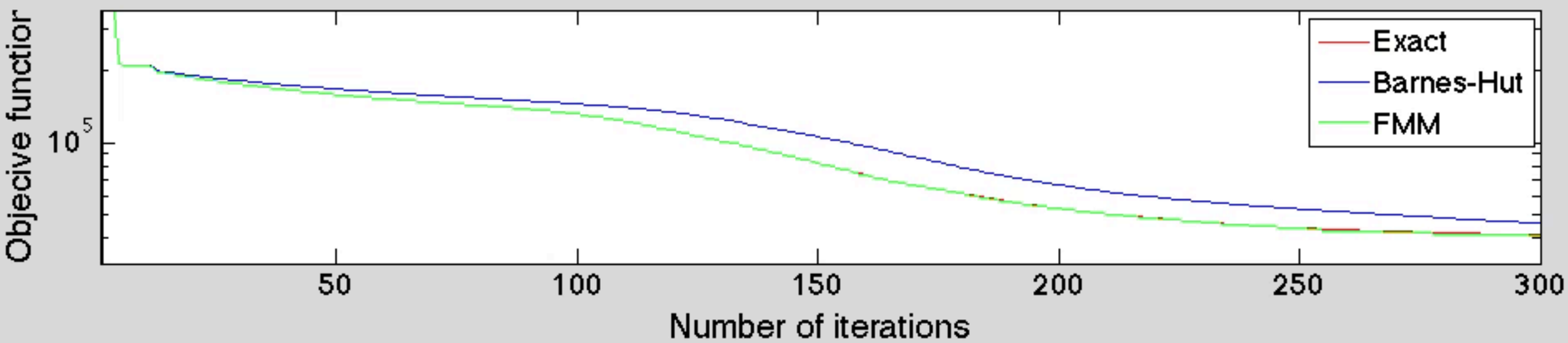




Experiments: 60 000 handwritten digits

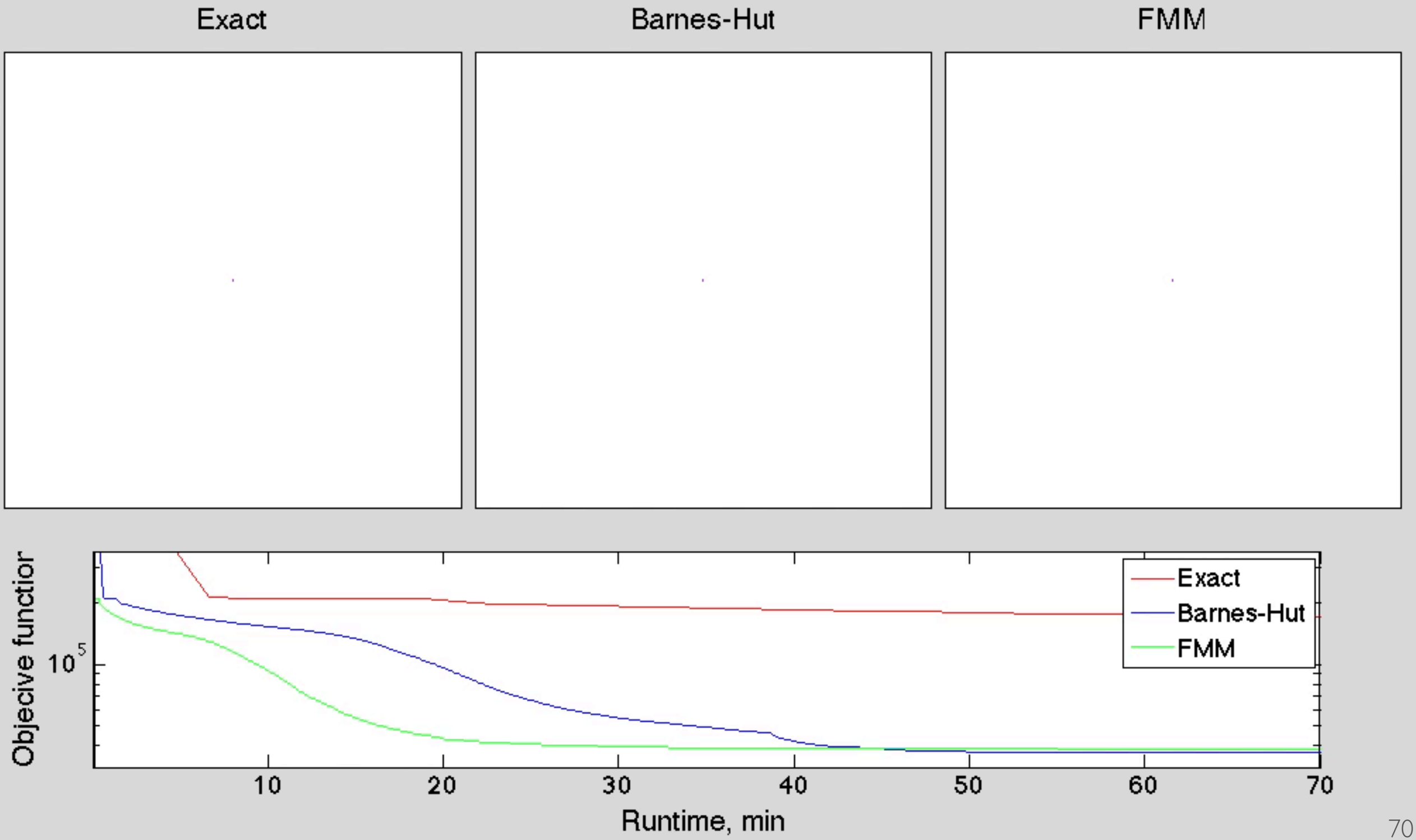
All methods show similar decrease in the objective function *per iteration.*





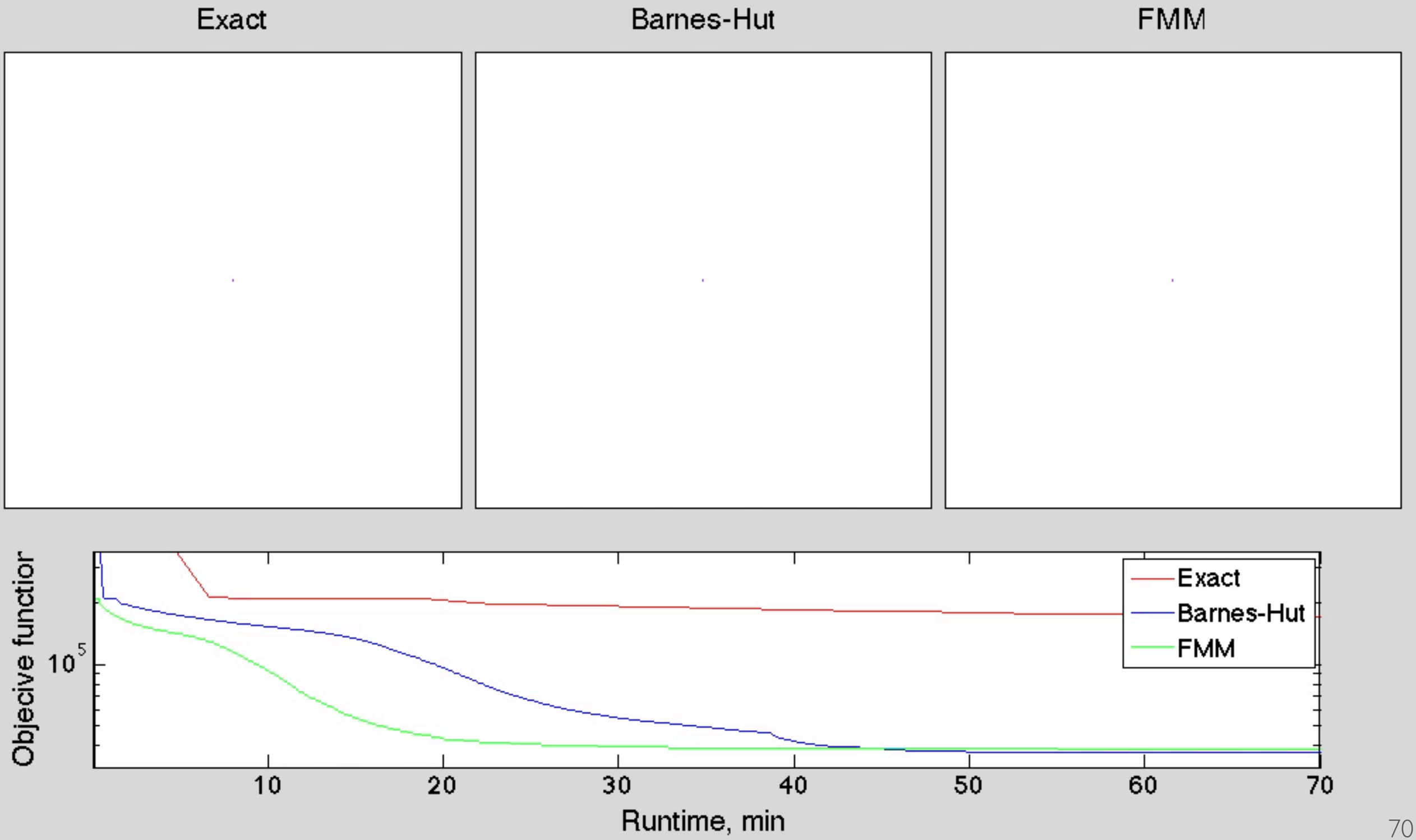
Experiments: 60 000 handwritten digits

The decrease is very different if considered *per minute of runtime.*

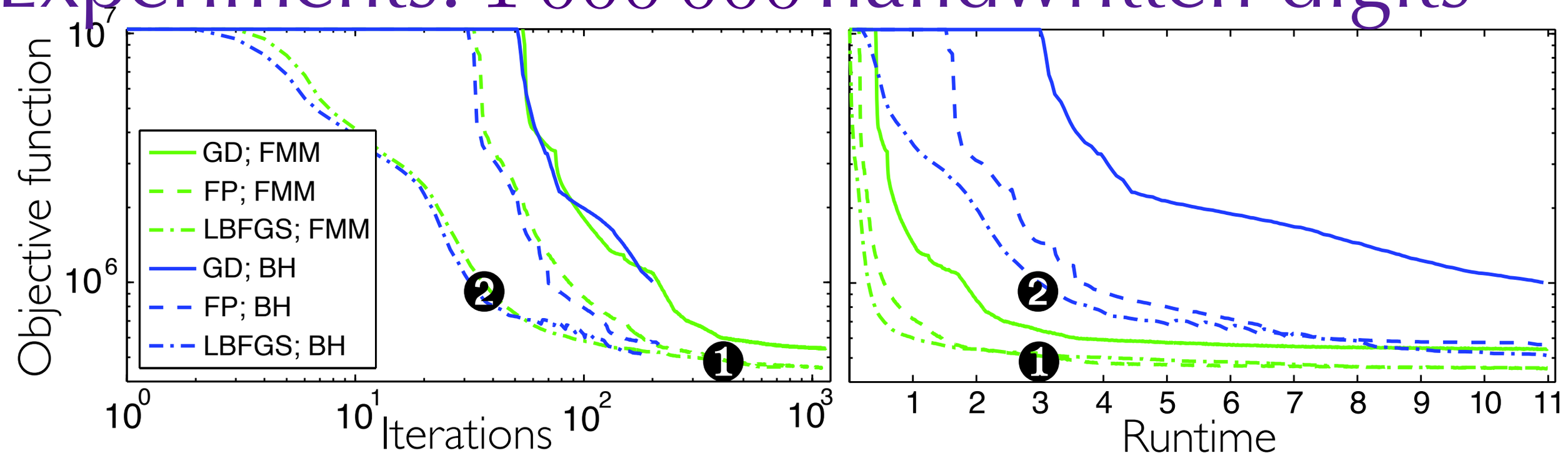


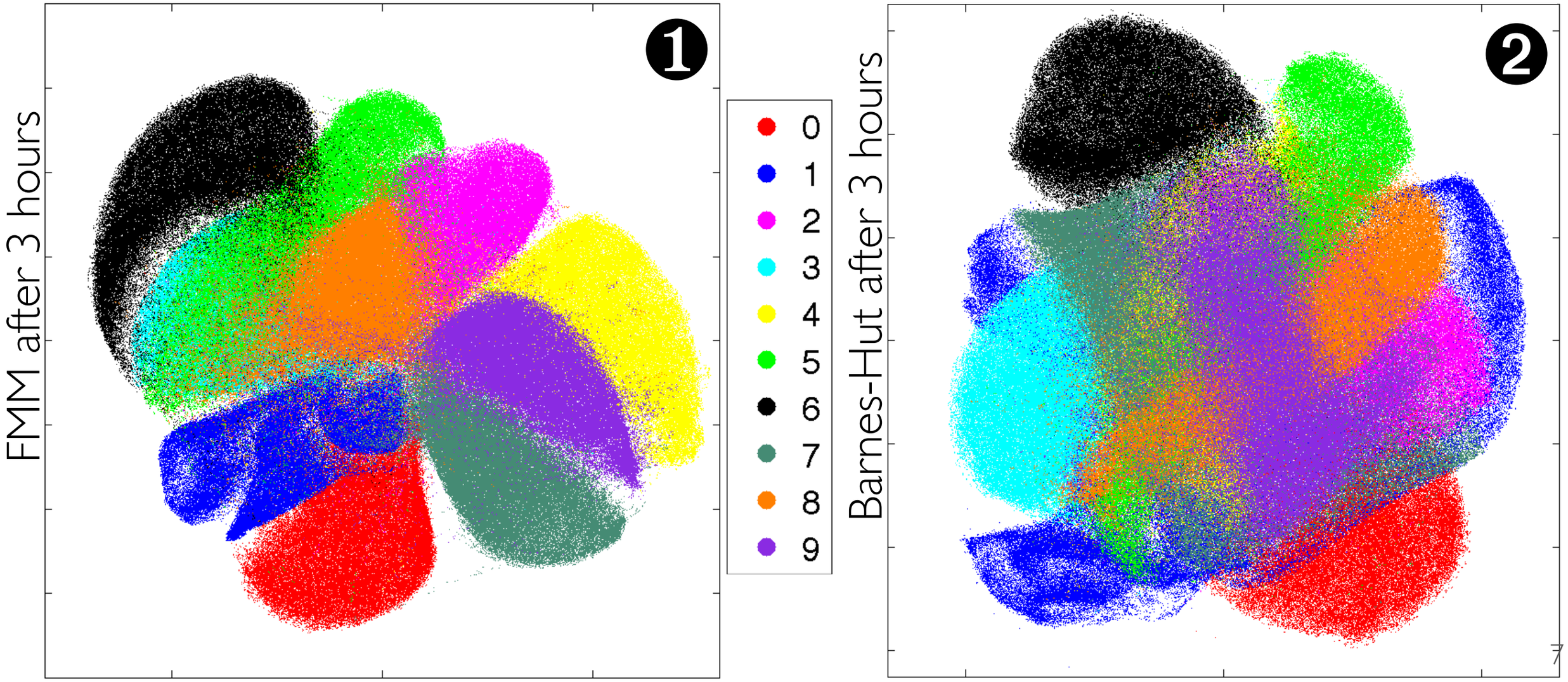
Experiments: 60 000 handwritten digits

The decrease is very different if considered *per minute of runtime.*



Experiments: 1 000 000 handwritten digits





Experiments: 1 000 000 handwritten digits

Experiments: 1 000 000 handwritten digits

infiniteMNIST,N=1020000,EE/L-BFGS/FMM,it#1,t=0.03 hours

infiniteMNIST,N=1020000,EE/L-BFGS/FMM,it#1,t=0.03 hours

Conclusions

- Nonlinear dimensionality reduction gives good results, but usually expensive to train.
- New ways to scale-up NLE algorithms to datasets with $> 10^6$ points (on a single core with moderate memory requirements):
  - For spectral learning methods (LE, LLE, PCA, Spec. clustering):
    - Locally Linear Landmarks (LLL) reformulates the problem on a subset, while retaining the structure of the whole dataset.
  - For nonlinear embedding methods (SNE, *t*-SNE, EE):
    - spectral direction gives 10-100x speedup comparing to the traditional optimization methods.
    - *N*-Body approximations using Barnes-Hut or FMM reduces the complexity of the algorithms to $\mathcal{O}(N \log N)$ and $\mathcal{O}(N)$ respectively.

Papers

- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2014): "Fast, accurate spectral clustering using locally linear landmarks", in submission.
- M. Á. Carreira-Perpiñán and **Max Vladymyrov** (2014): "A fast, universal algorithm to learn parametric nonlinear embeddings", in submission.
- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2014): "Linear-time training of nonlinear low-dimensional embeddings", 17th International Conference on Artificial Intelligence and Statistics (AISTATS 2014), pp. 968–977. Acceptance rate: 35.8% (120/335), poster.
- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2013): "Locally linear landmarks for large-scale manifold learning". 24th European Conference on Machine Learning (ECML 2013), pp. 256–271. Acceptance rate: 25.0% (111/443), oral.
- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2013): "Entropic affinities: properties and efficient numerical computation". 30th International Conference on Machine Learning (ICML 2013), pp. 477–485. Acceptance rate: 23.5% (283/1204), oral.
- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2012): "Partial-Hessian strategies for fast learning of nonlinear embeddings". 29th International Conference on Machine Learning (ICML 2012), pp. 345–352. Acceptance rate: 27.2% (242/890), oral.

Software

- Code for all the methods presented is available online:
  https://eng.ucmerced.edu/people/vladymyrov

Papers

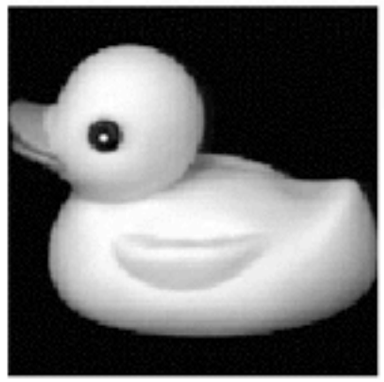
- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2014): "Fast, accurate spectral clustering using locally linear landmarks", in submission.
- M. Á. Carreira-Perpiñán and **Max Vladymyrov** (2014): "A fast, universal algorithm to learn parametric nonlinear embeddings", in submission.
- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2014): "Linear-time training of nonlinear low-dimensional embeddings", 17th International Conference on Artificial Intelligence and Statistics (AISTATS 2014), pp. 968–977. Acceptance rate: 35.8% (120/335), poster.
- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2013): "Locally linear landmarks for large-scale manifold learning". 24th European Conference on Machine Learning (ECML 2013), pp. 256–271. Acceptance rate: 25.0% (111/443), oral.
- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2013): "Entropic affinities: properties and efficient numerical computation". 30th International Conference on Machine Learning (ICML 2013), pp. 477–485. Acceptance rate: 23.5% (283/1204), oral.
- **Max Vladymyrov** and M. Á. Carreira-Perpiñán (2012): "Partial-Hessian strategies for fast learning of nonlinear embeddings". 29th International Conference on Machine Learning (ICML 2012), pp. 345–352. Acceptance rate: 27.2% (242/890), oral.

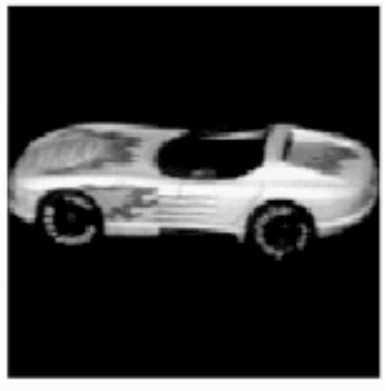
Software

- Code for all the methods presented is available online:
  https://eng.ucmerced.edu/people/vladymyrov

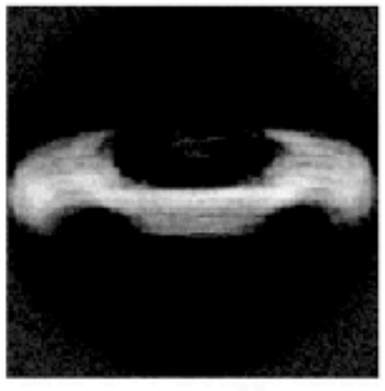
Out-of-sample mapping: example

Original

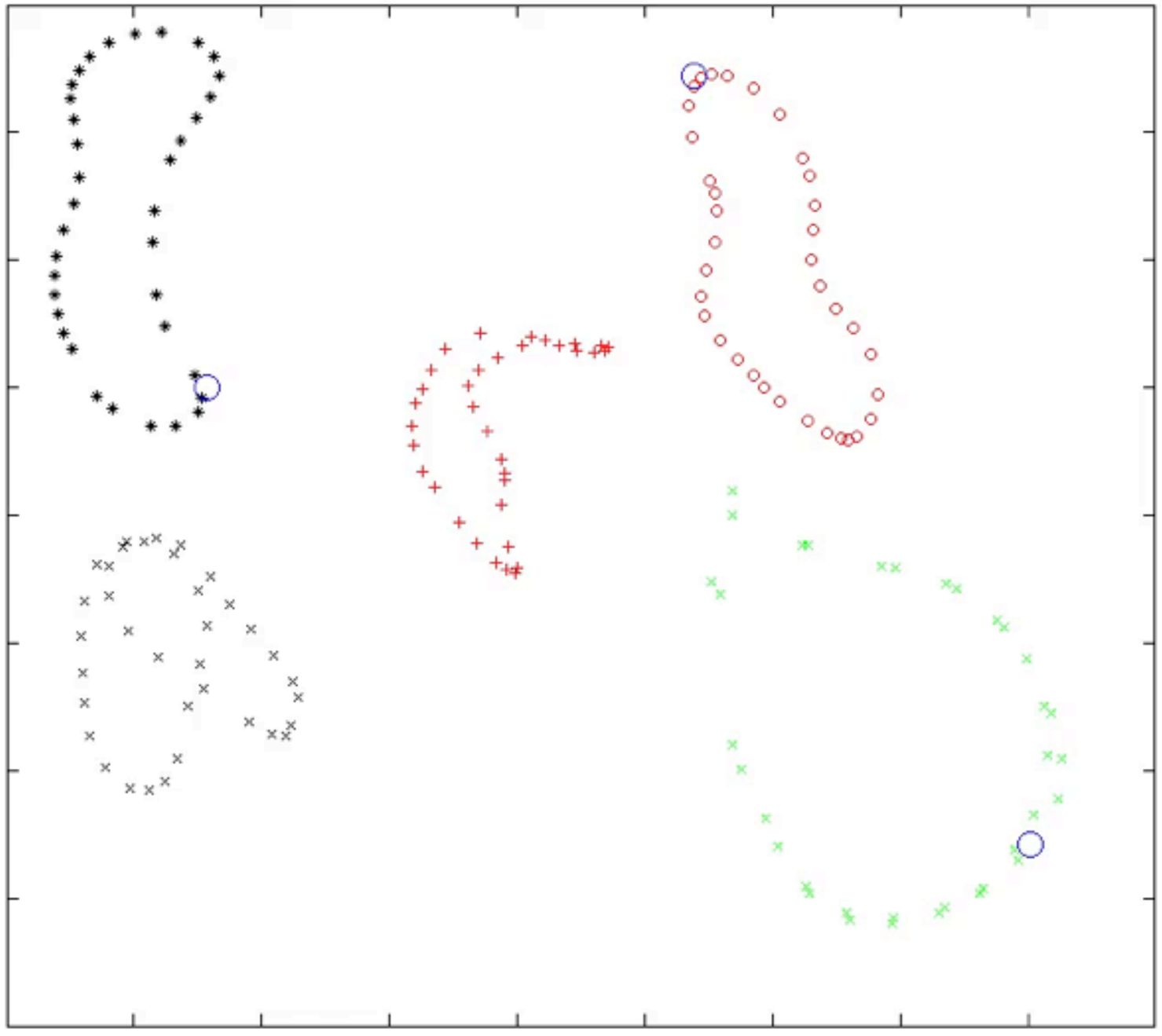




Reconstructed

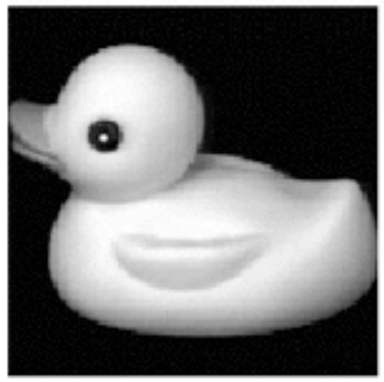


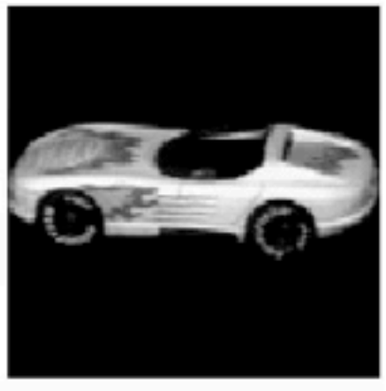
Low-dimensional space



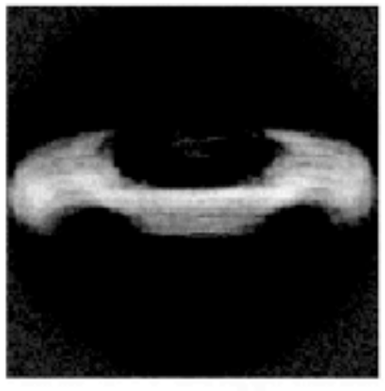
Out-of-sample mapping: example

Original

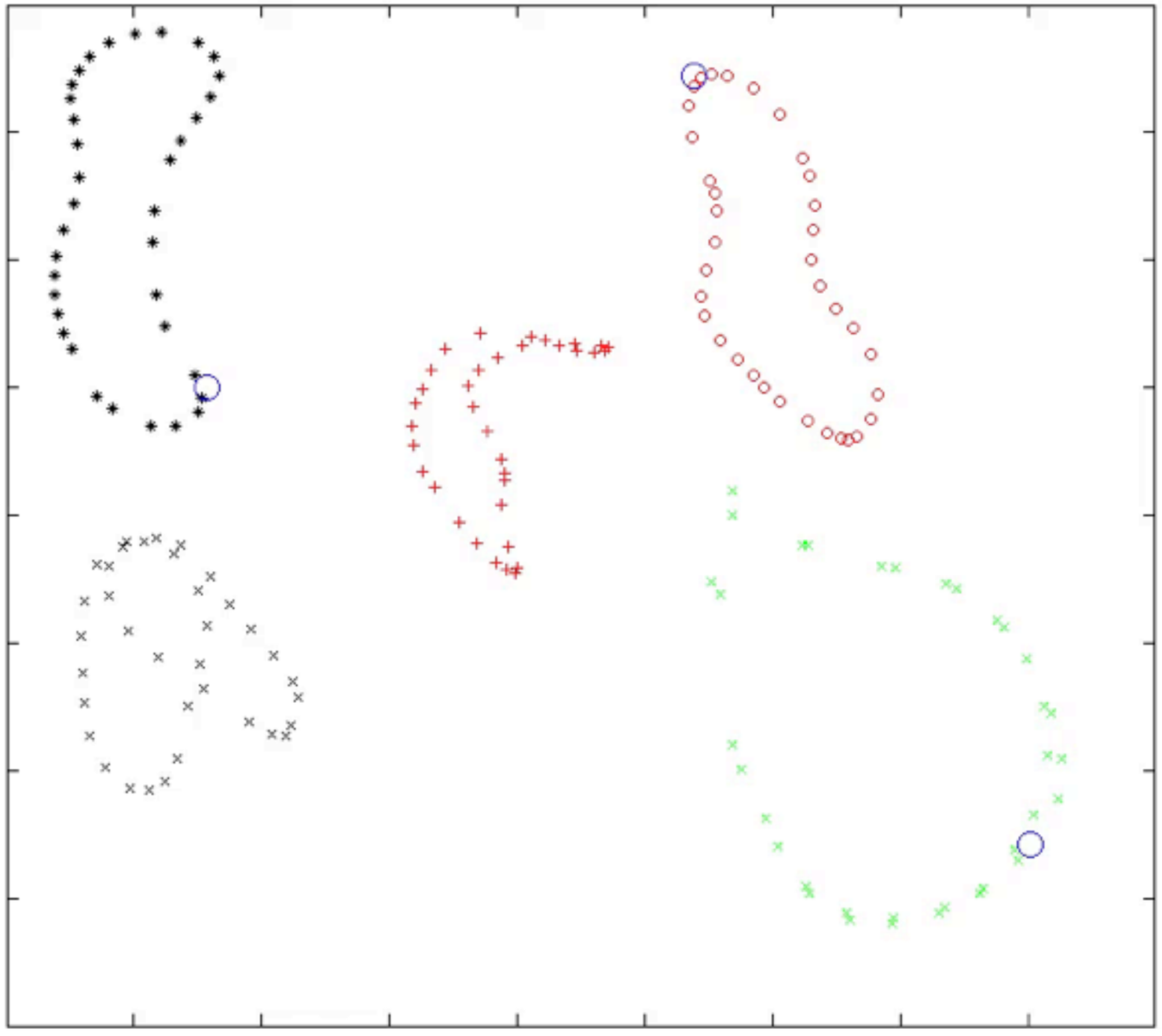




Reconstructed



Low-dimensional space



Fast out-of-sample mapping

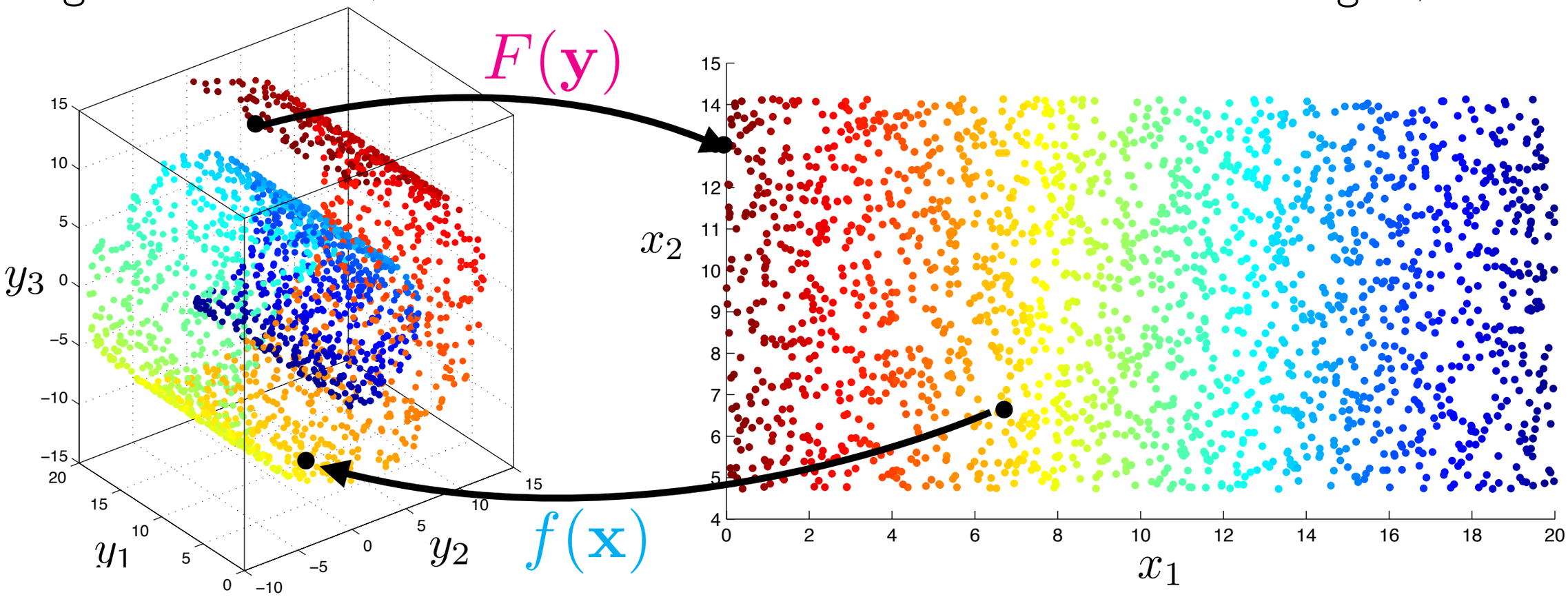
- Given a new point $\mathbf{y} \in \mathbb{R}^D$, we solve the original problem over $(\mathbf{X}\,\mathbf{x})$ and $(\mathbf{Y}\,\mathbf{y})$, subject to keeping the embedding $\mathbf{X}$ fixed:

$$E'(\mathbf{x}, \mathbf{y}, \lambda) = E^+(\mathbf{x}, \mathbf{y}) + \lambda E^-(\mathbf{x}, \mathbf{y}) \qquad \lambda \geq 0$$

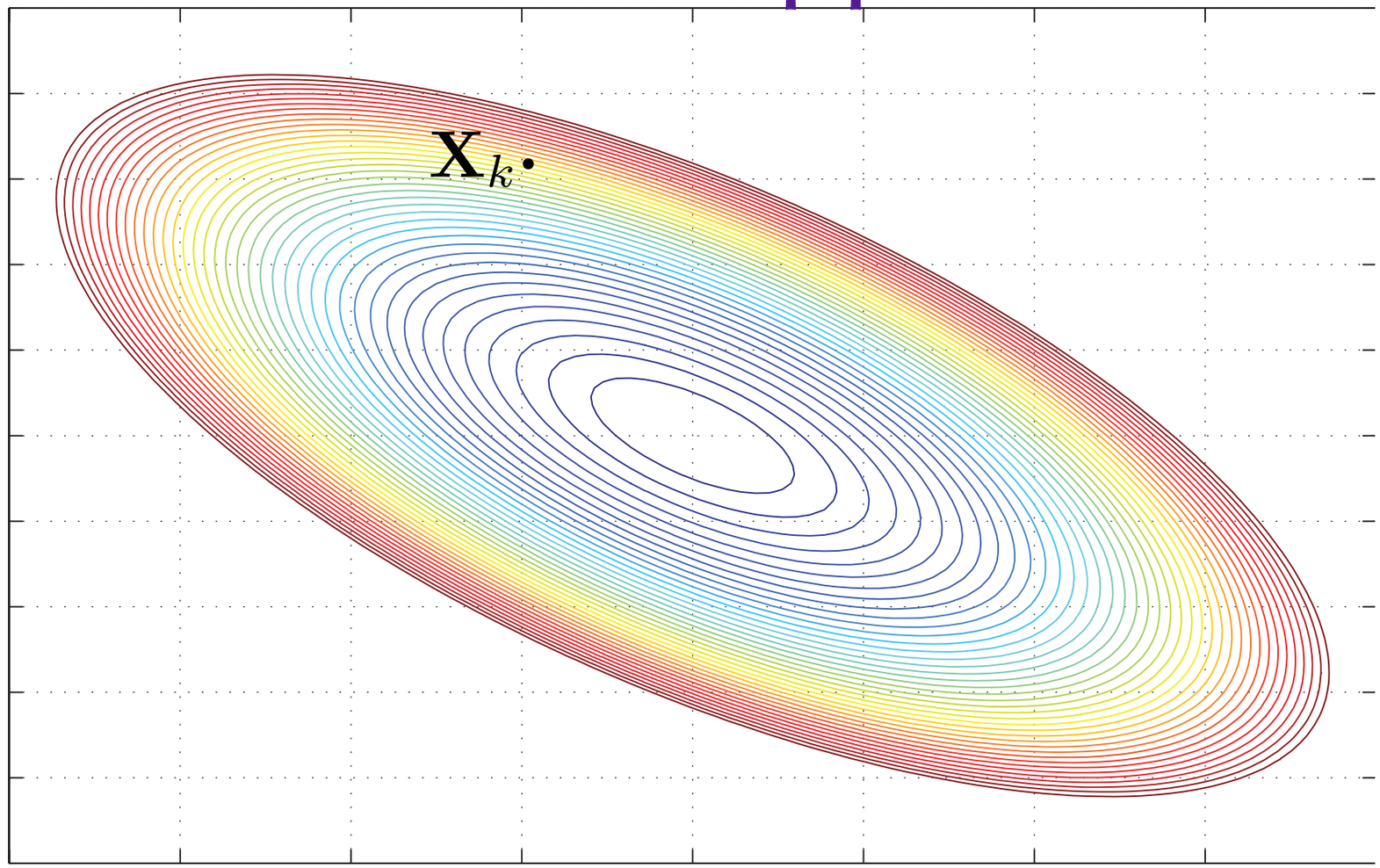
- Project new high-d point $\mathbf{y}$: $F(\mathbf{y}) = \arg\min_{\mathbf{x}} E'(\mathbf{x}, \mathbf{y})$
- Reconstruct new low-d point $\mathbf{x}$: $f(\mathbf{x}) = \arg\min_{\mathbf{y}} E'(\mathbf{x}, \mathbf{y})$

Original dataset $\mathbf{Y}, D = 3$

Low-dimensional embedding $\mathbf{X}$, $d = 2$

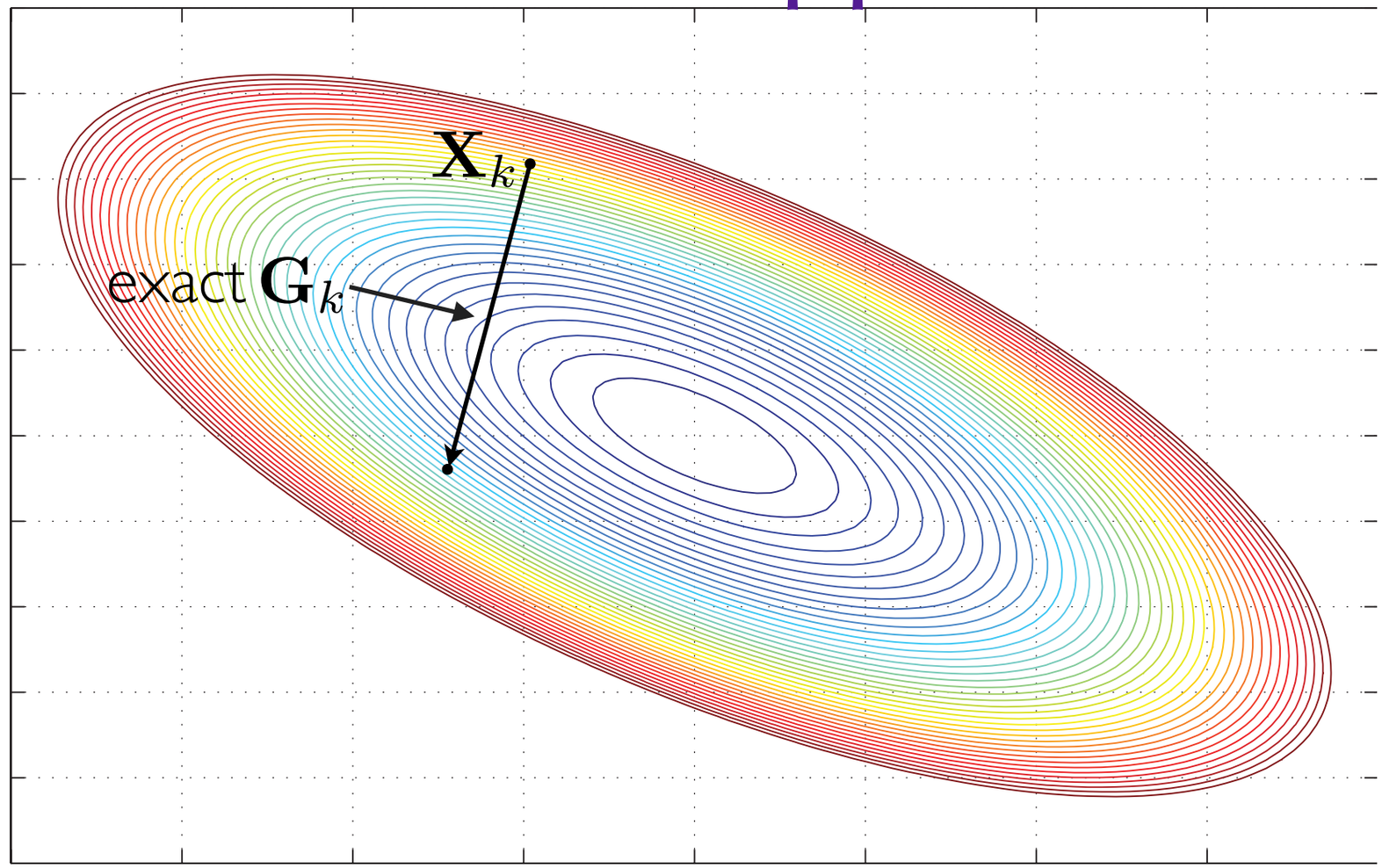


Model the effect of the approximate gradient



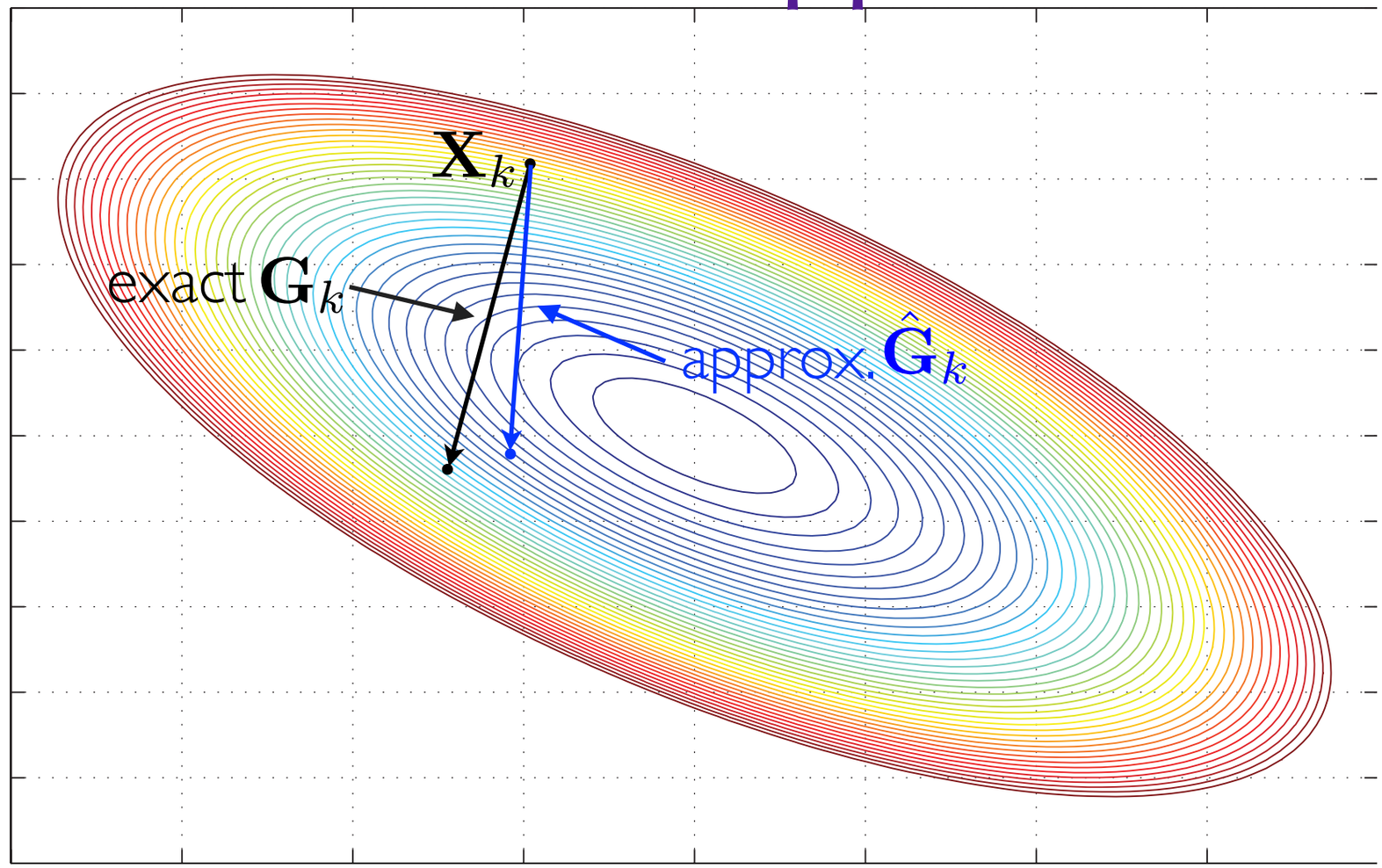
- For each iteration we incur the error $\mathbf{X}_{k+1} = \mathbf{X}_k + \boldsymbol{\epsilon}_k$.
- Approximation the error with the model $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$.
- σ is a model parameter and represents the accuracy of the approximation.

Model the effect of the approximate gradient



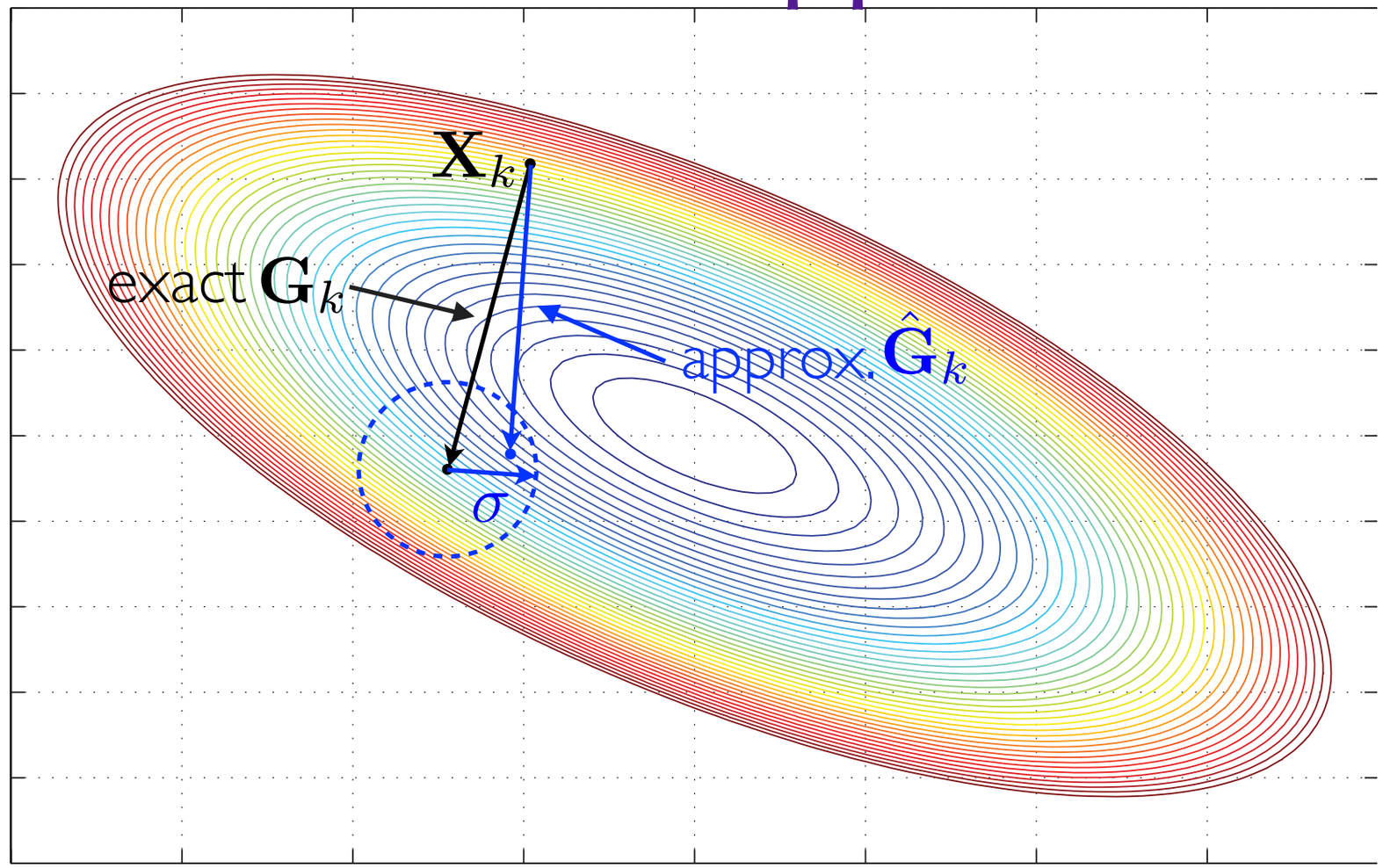
- For each iteration we incur the error $\mathbf{X}_{k+1} = \mathbf{X}_k + \boldsymbol{\epsilon}_k$.
- Approximation the error with the model $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$.
- σ is a model parameter and represents the accuracy of the approximation.

Model the effect of the approximate gradient



- For each iteration we incur the error $\mathbf{X}_{k+1} = \mathbf{X}_k + \boldsymbol{\epsilon}_k$.
- Approximation the error with the model $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$.
- σ is a model parameter and represents the accuracy of the approximation.

Model the effect of the approximate gradient



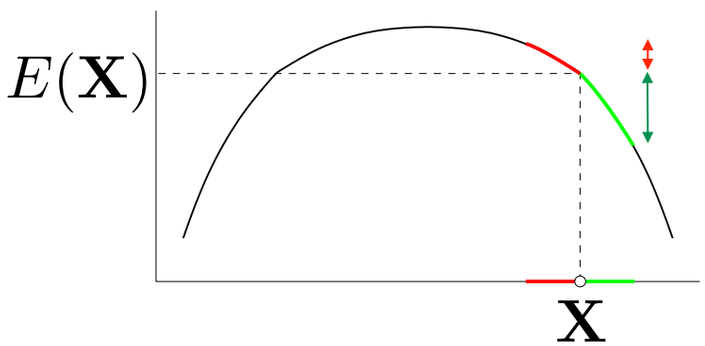
- For each iteration we incur the error $\mathbf{X}_{k+1} = \mathbf{X}_k + \boldsymbol{\epsilon}_k$.
- Approximation the error with the model $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$.
- σ is a model parameter and represents the accuracy of the approximation.

Model the effect of the approximate gradient

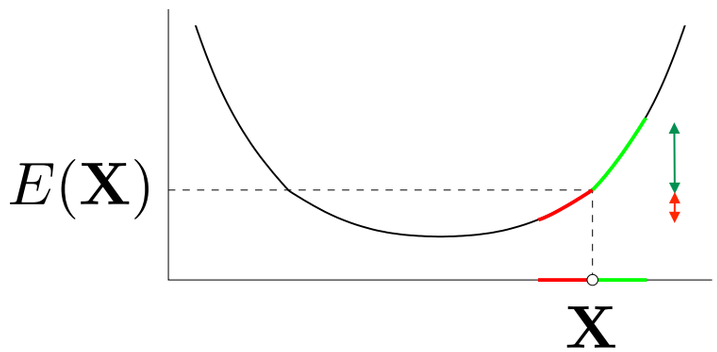
Mean of the absolute error:

$$\langle E(\mathbf{X}+\boldsymbol{\epsilon}) - E(\mathbf{X})\rangle = \frac{1}{2}\sigma^2 \operatorname{tr}\left(\nabla^2 E(\mathbf{X})\right) + \mathcal{O}(\sigma^4)$$

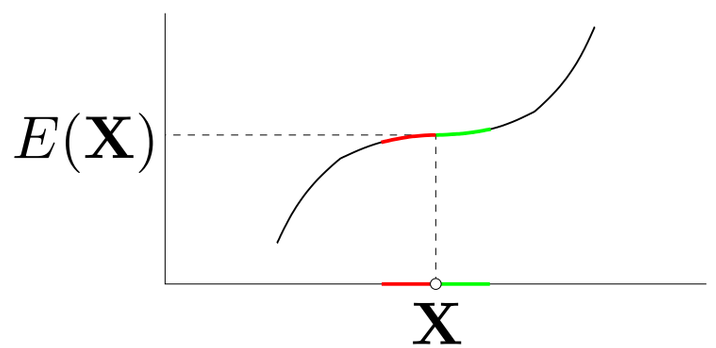
Negative curvature



Positive curvature



No curvature

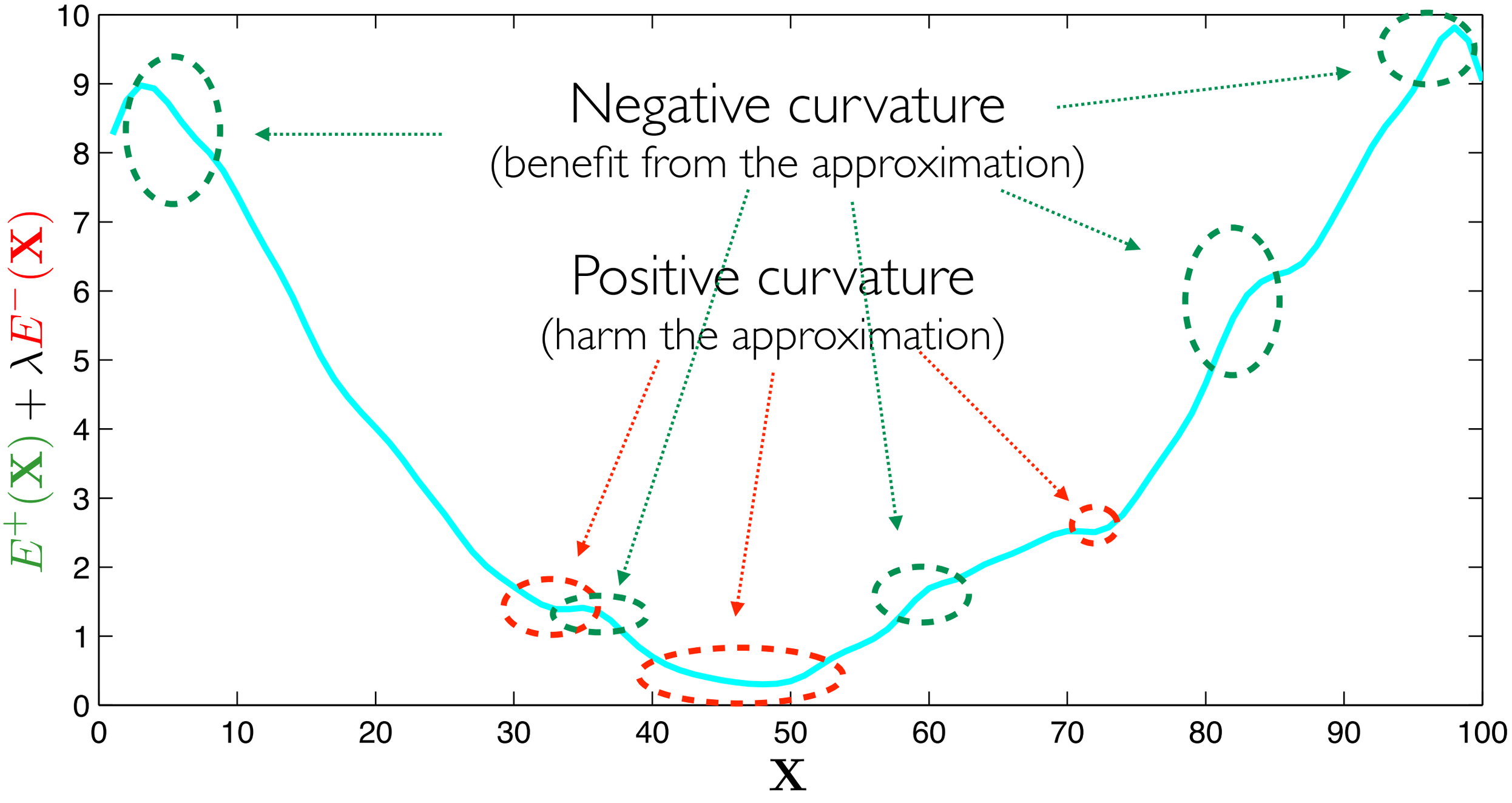


We have qualitative predictions:

1. Adding noise will be beneficial only where the mean curvature $\frac{1}{n}\operatorname{tr}\left(\nabla^2 E(\mathbf{X})\right)$ is negative
2. When the mean curvature is positive, the lower the accuracy the worse the optimization;
3. $\Delta E(\mathbf{X})$ will vary widely at the beginning of the optimization and become approximately constant and equal to $\frac{1}{2}\sigma^2 \operatorname{tr}\left(\nabla^2 E(\mathbf{X})\right)$.

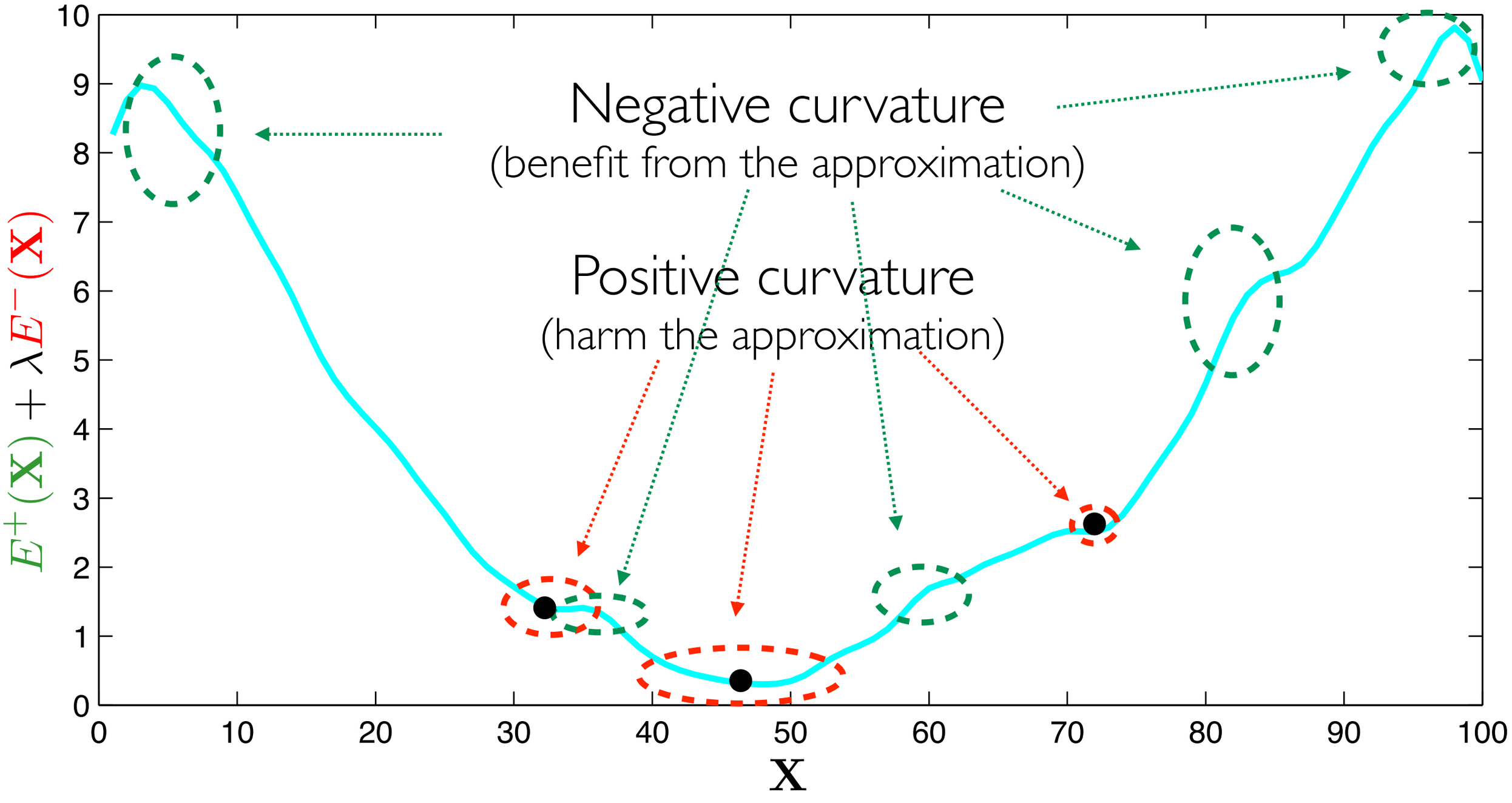
Model the effect of the approximate gradient

Under this model, we can suggest to increase the accuracy parameter as we proceed with iterations.



Model the effect of the approximate gradient

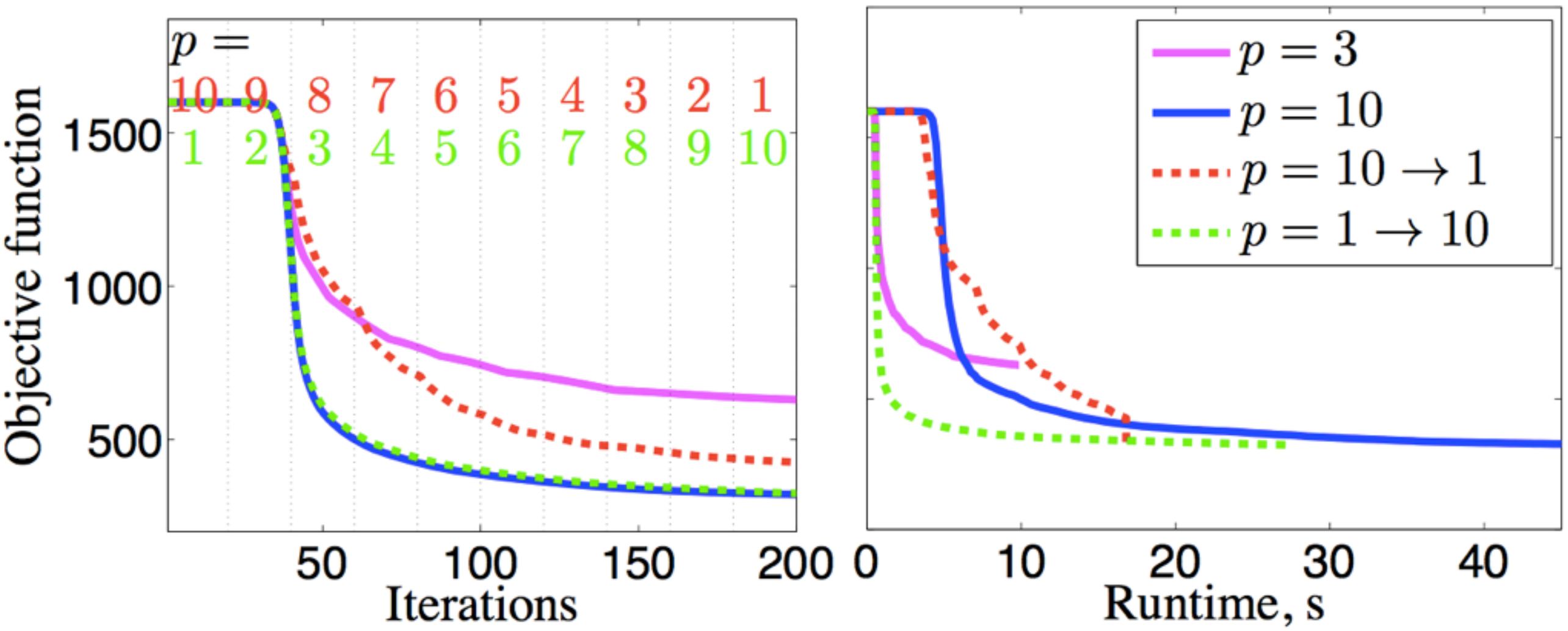
Under this model, we can suggest to increase the accuracy parameter as we proceed with iterations.

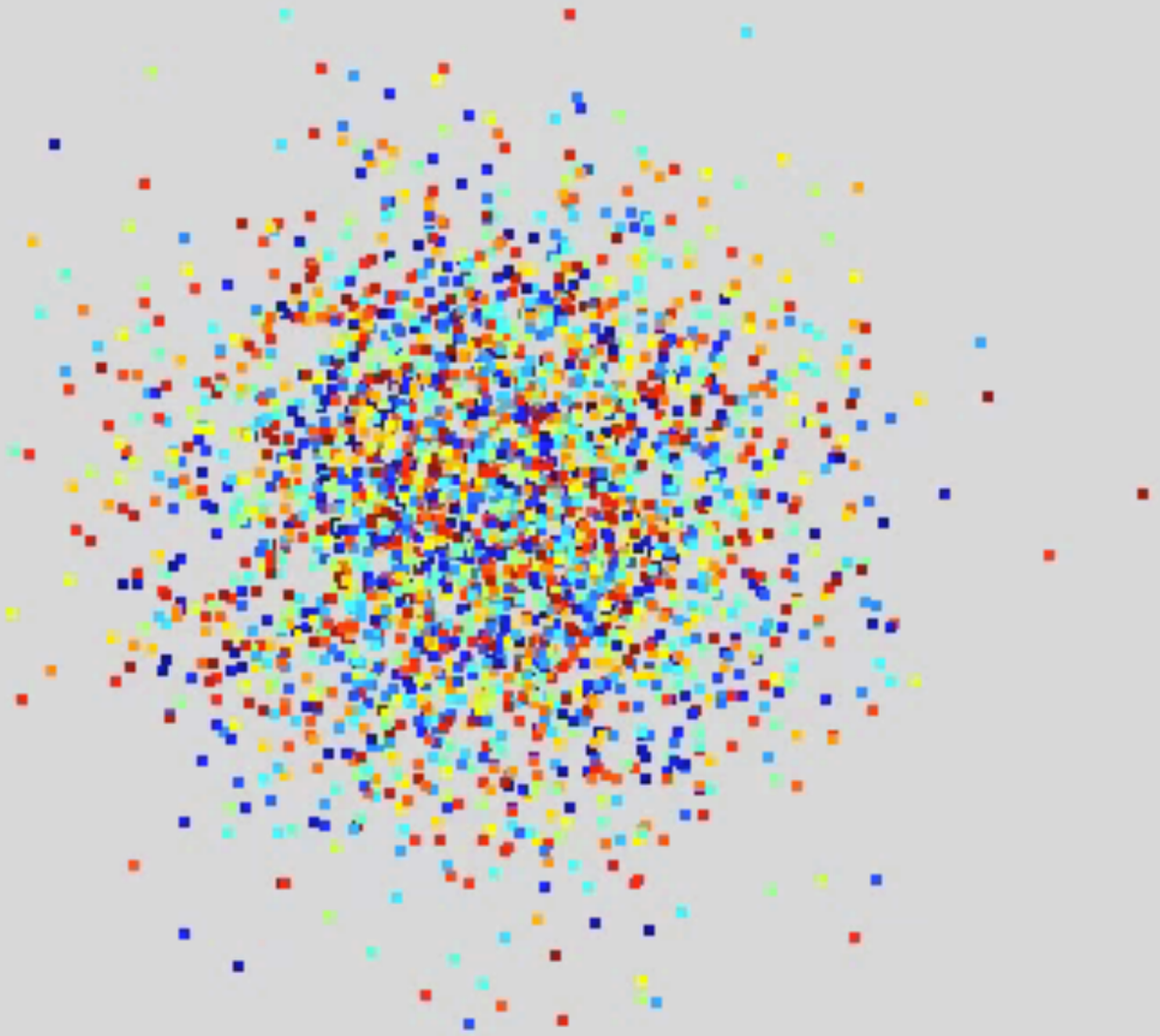


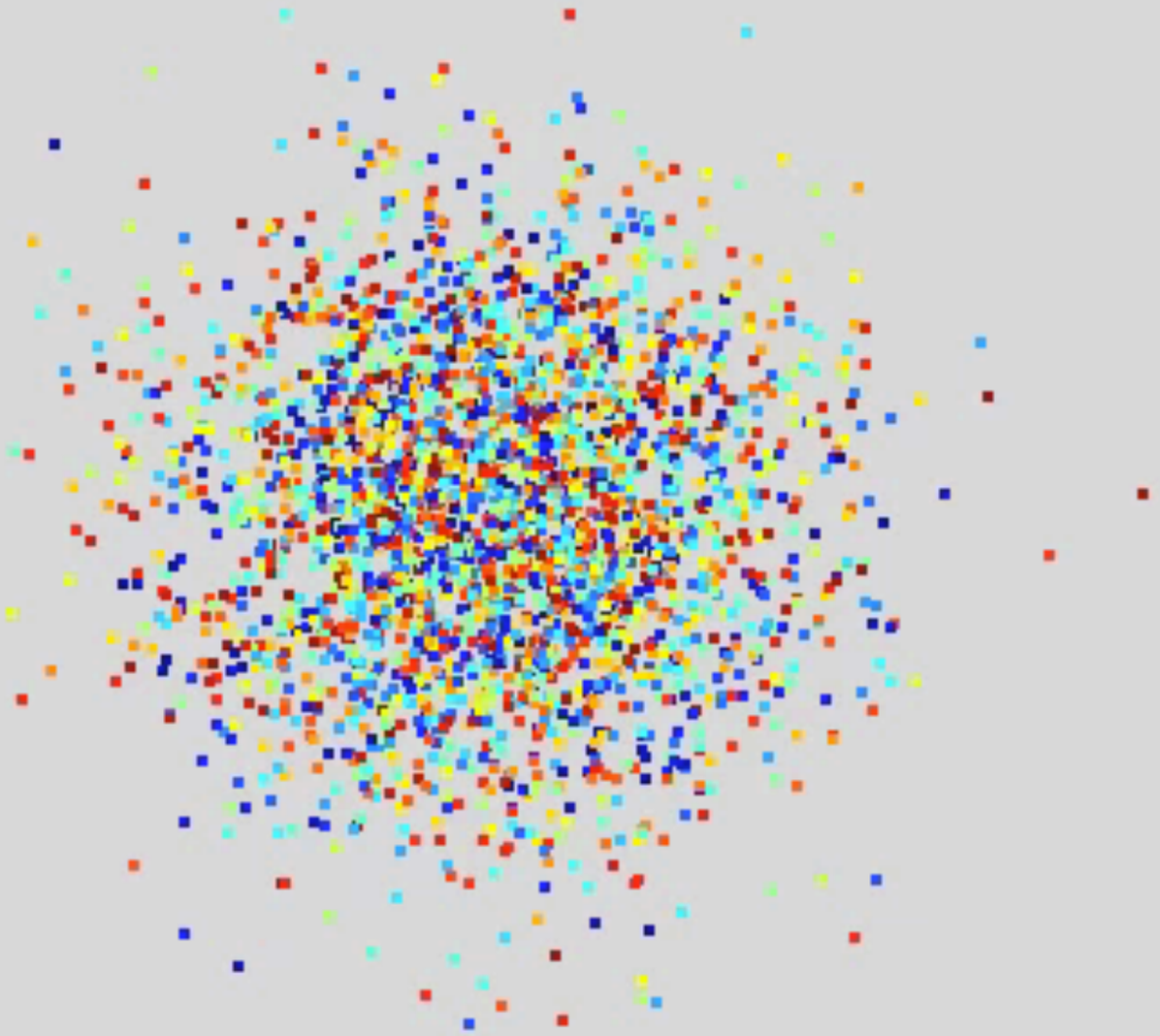
Accuracy of the approximation

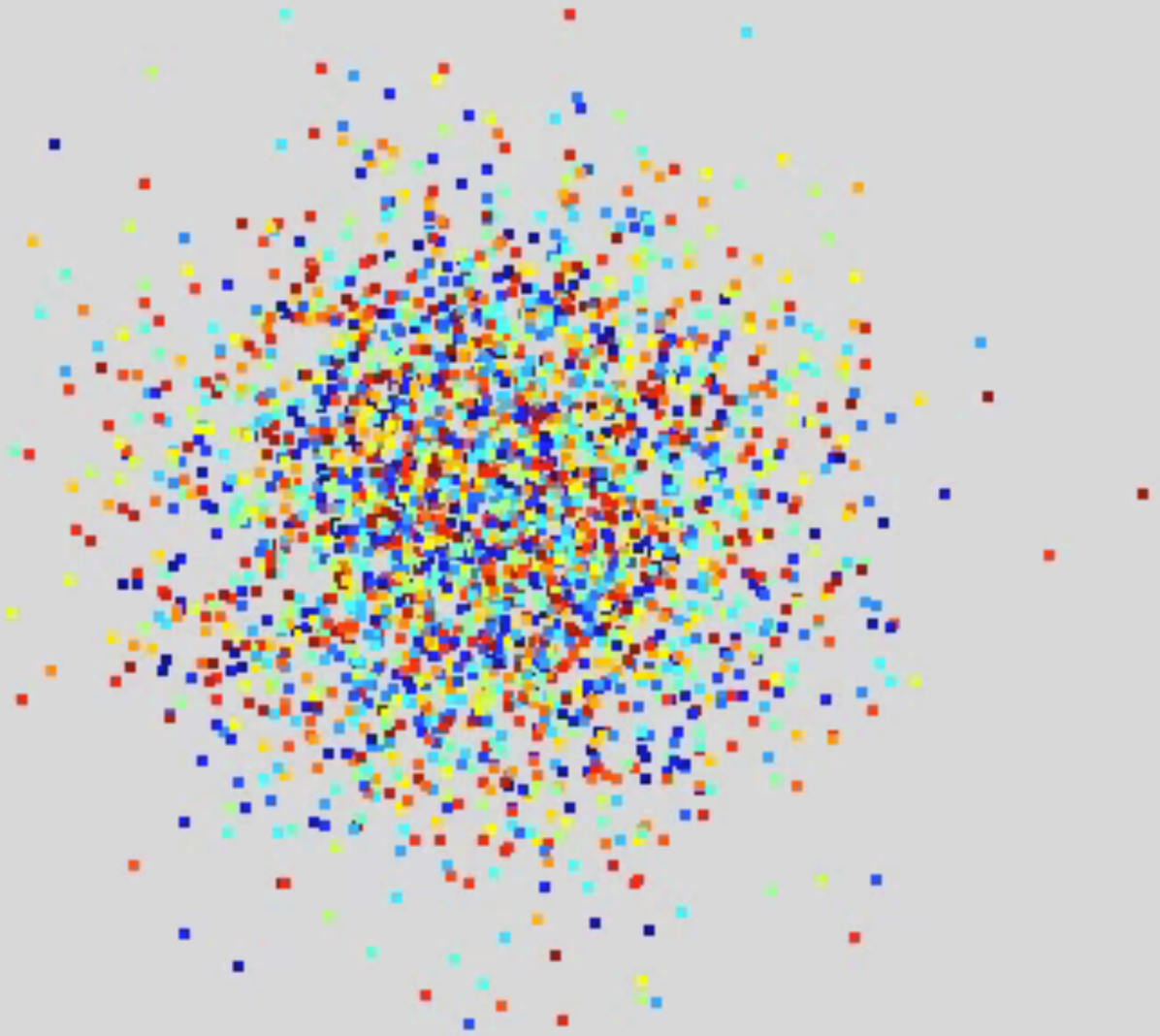
Compare different ways to change the accuracy of the approximation:

- fixed large,
- fixed small,
- changing from small to large,
- changing from large to small.



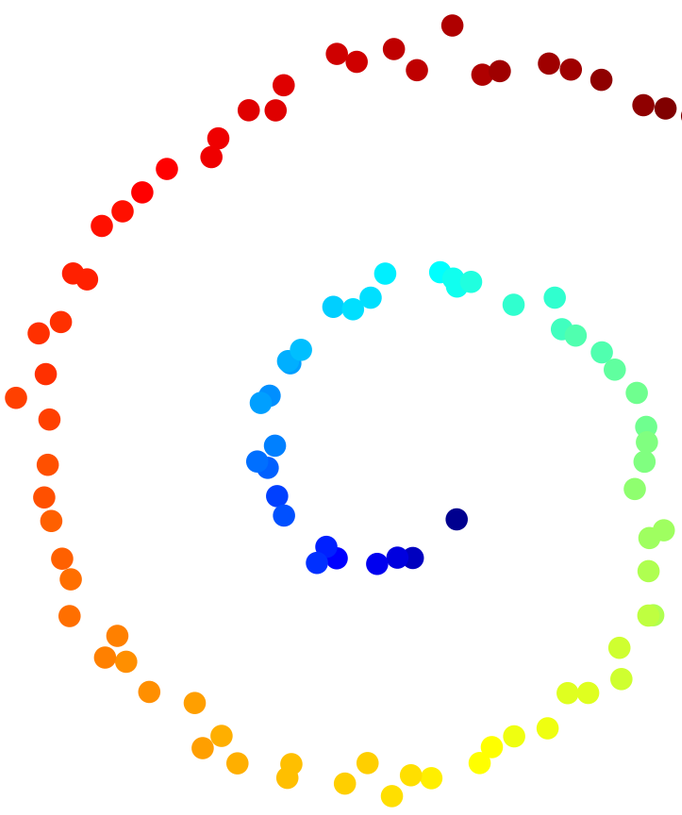






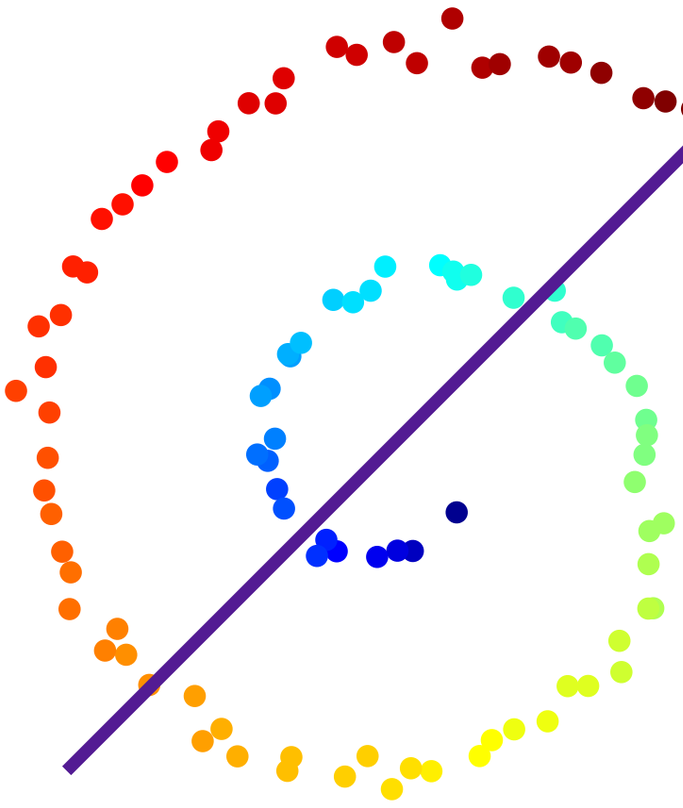
Principal Component Analysis (PCA)

Given a high-dimensional dataset, PCA find directions of *biggest variation* of the data and *projects* the data accordingly.



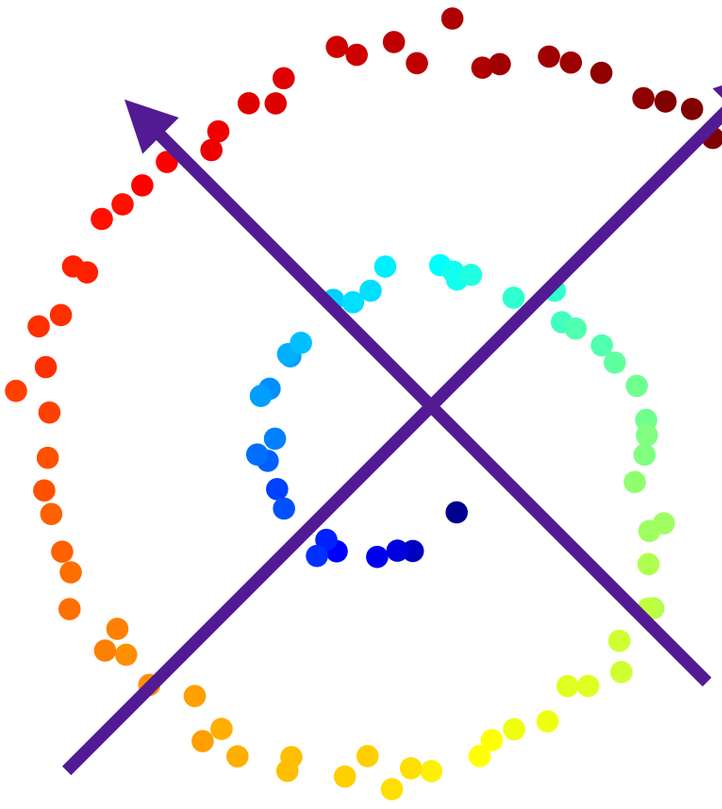
Principal Component Analysis (PCA)

Given a high-dimensional dataset, PCA find directions of *biggest variation* of the data and *projects* the data accordingly.



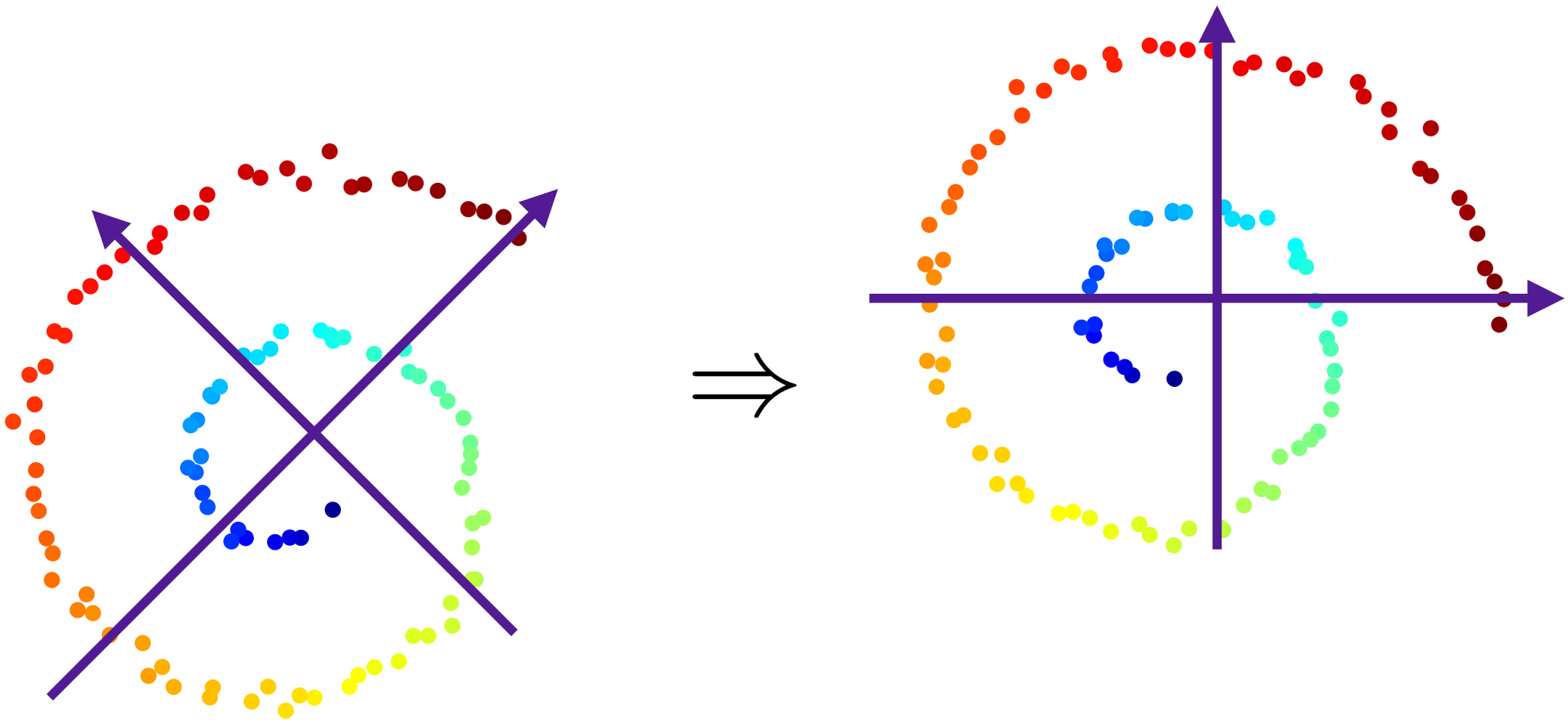
Principal Component Analysis (PCA)

Given a high-dimensional dataset, PCA find directions of *biggest variation* of the data and *projects* the data accordingly.



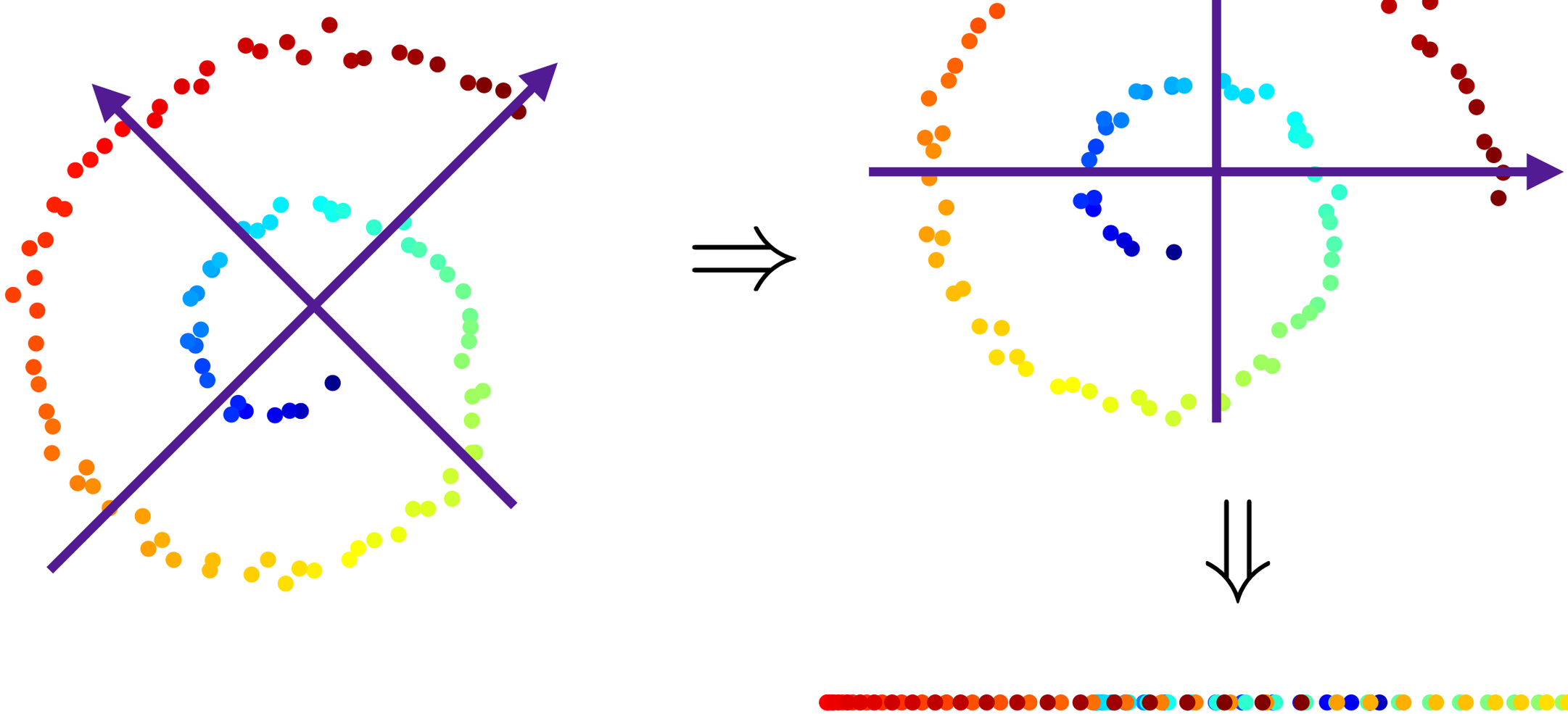
Principal Component Analysis (PCA)

Given a high-dimensional dataset, PCA find directions of *biggest variation* of the data and *projects* the data accordingly.



Principal Component Analysis (PCA)

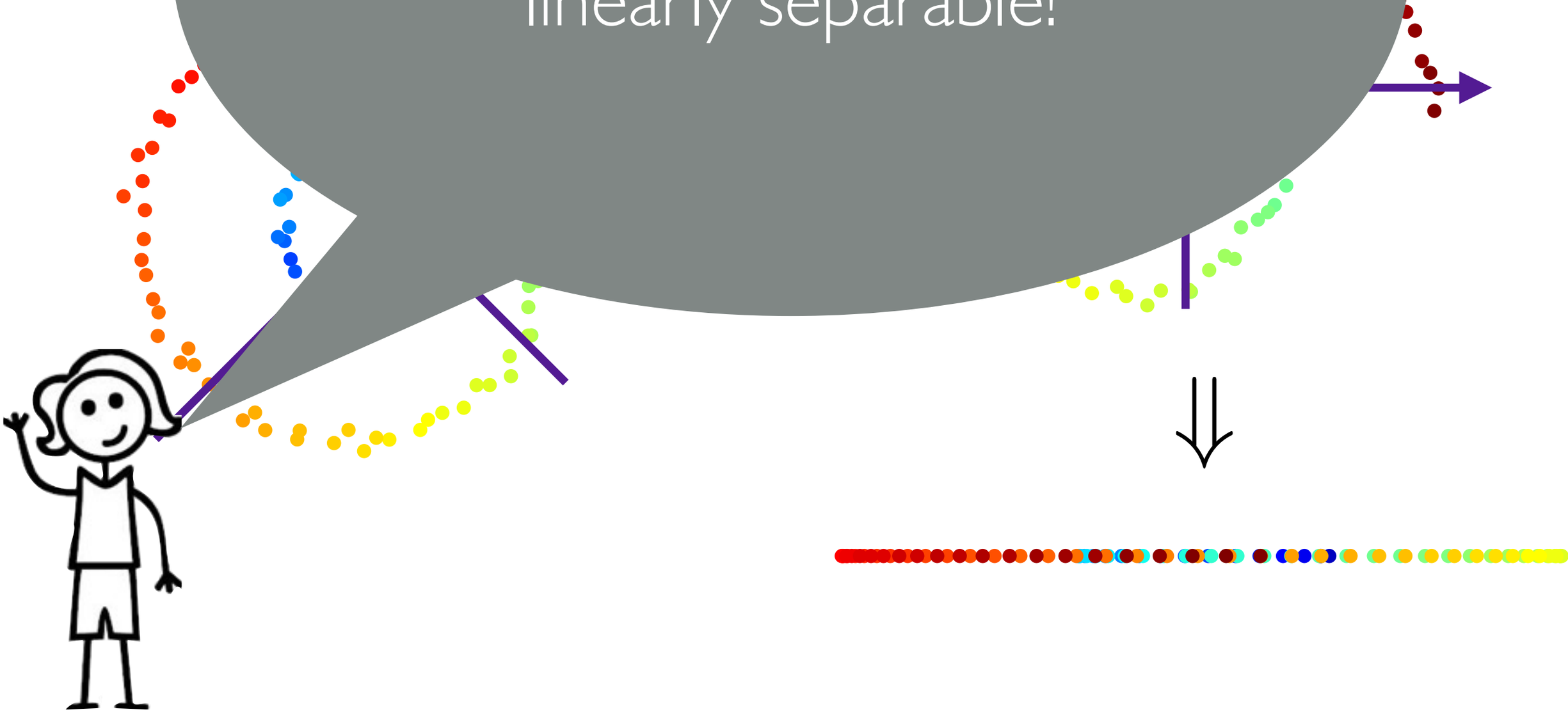
Given a high-dimensional dataset, PCA find directions of *biggest variation* of the data and *projects* the data accordingly.



Principal Component Analysis (PCA)

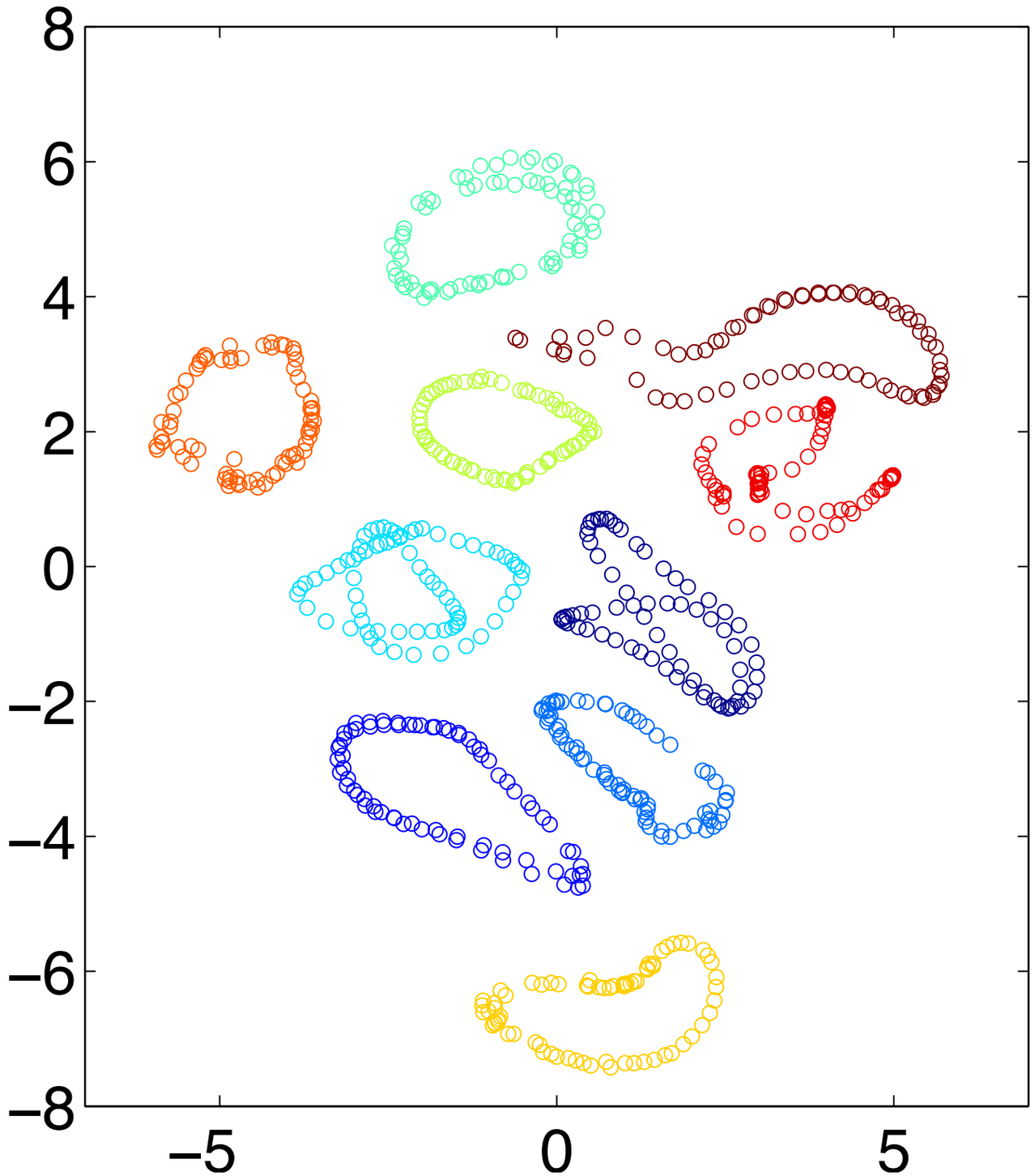
Given a high-dime ns of *biggest variation* of th

PCA works only if data is linearly separable!

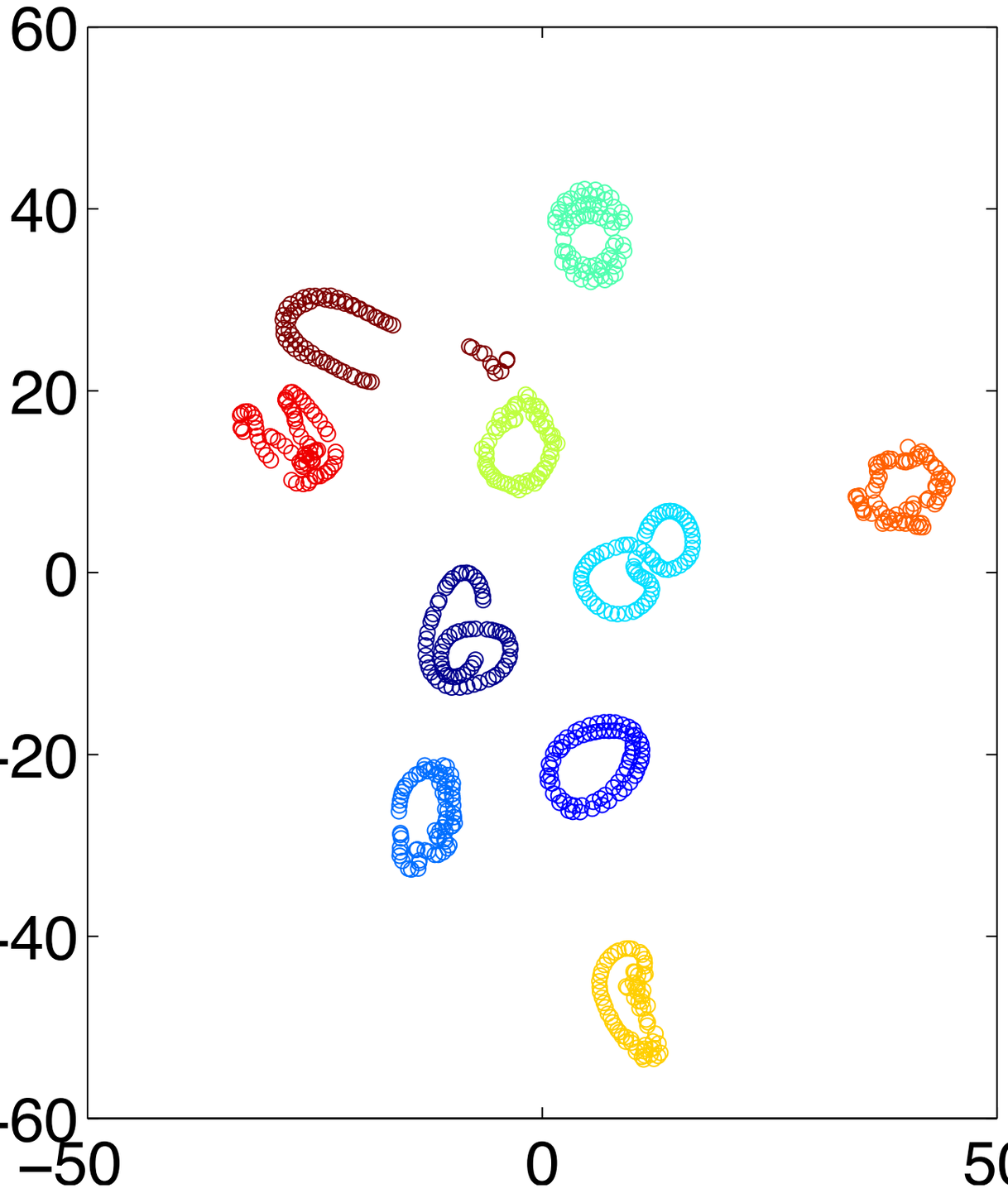


Rotational sequences

s-SNE

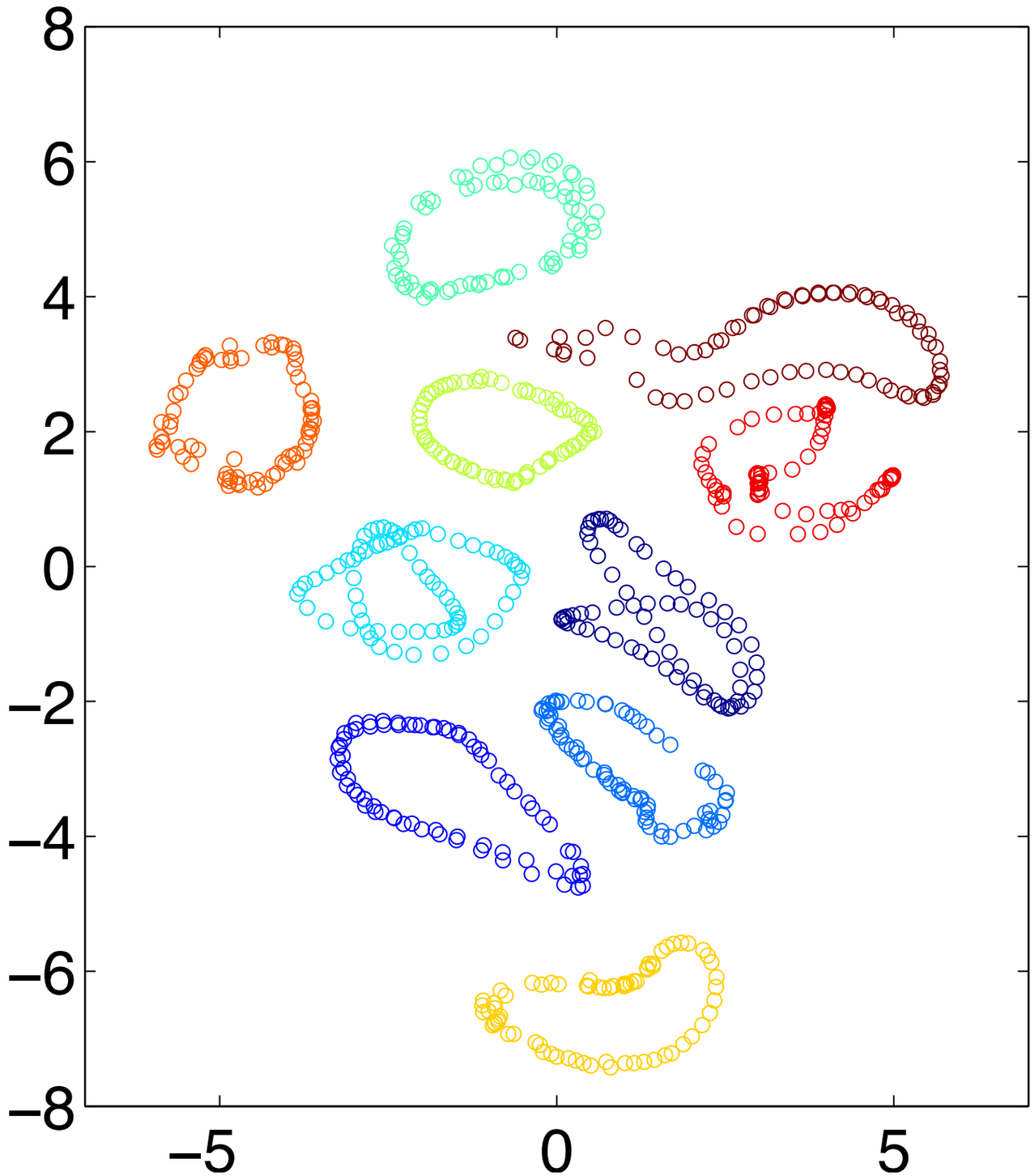


t-SNE

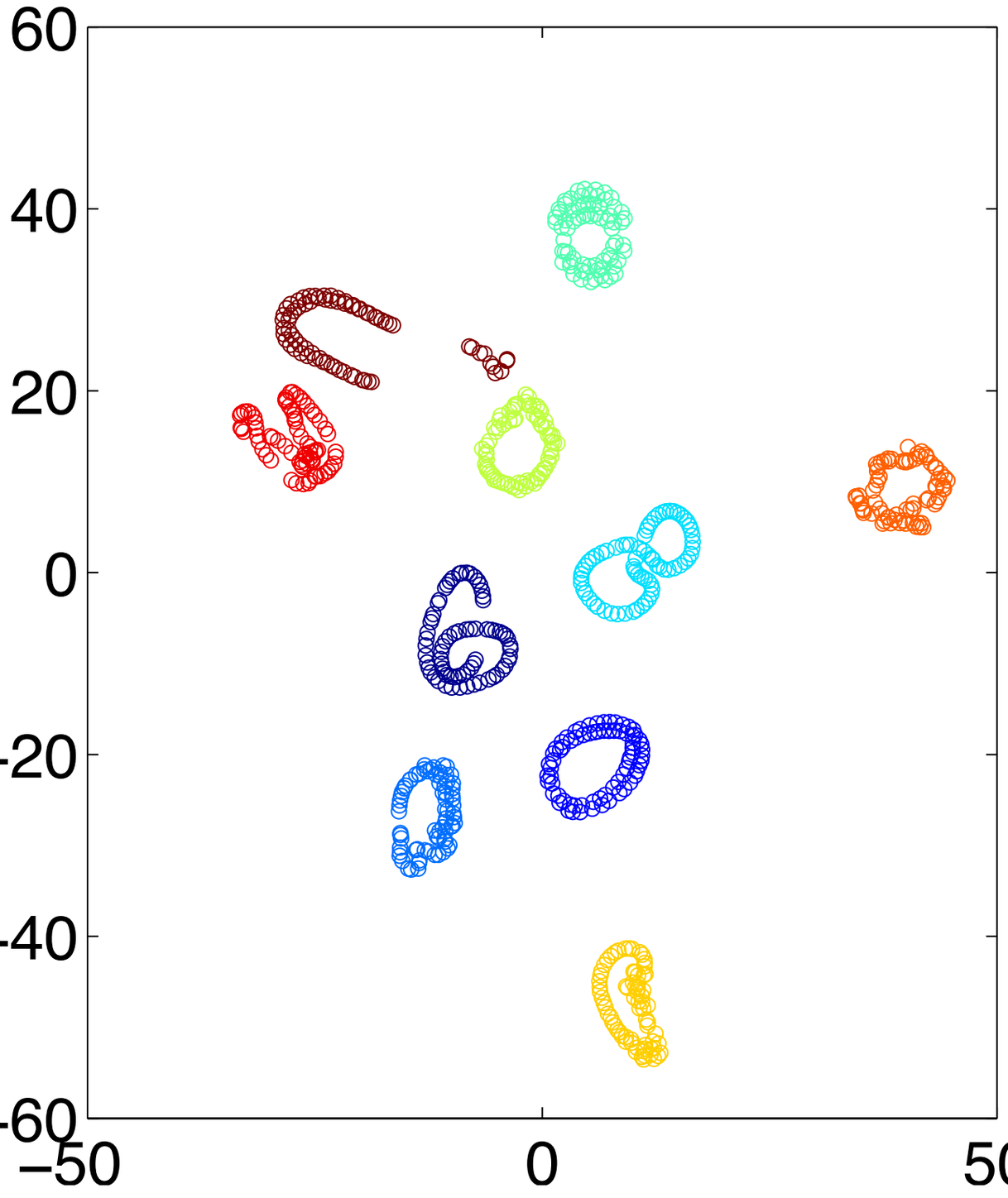


Rotational sequences

s-SNE

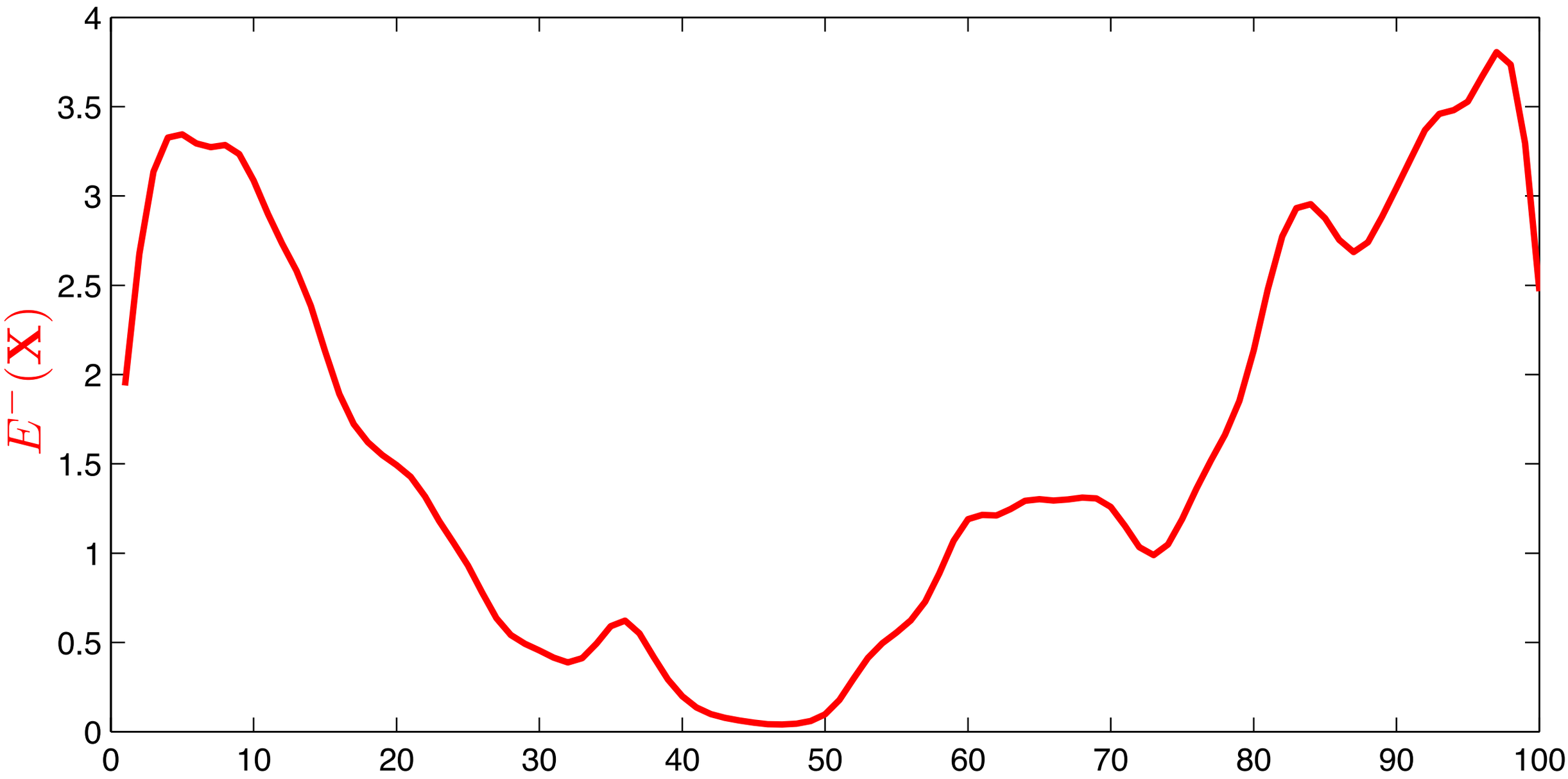


t-SNE



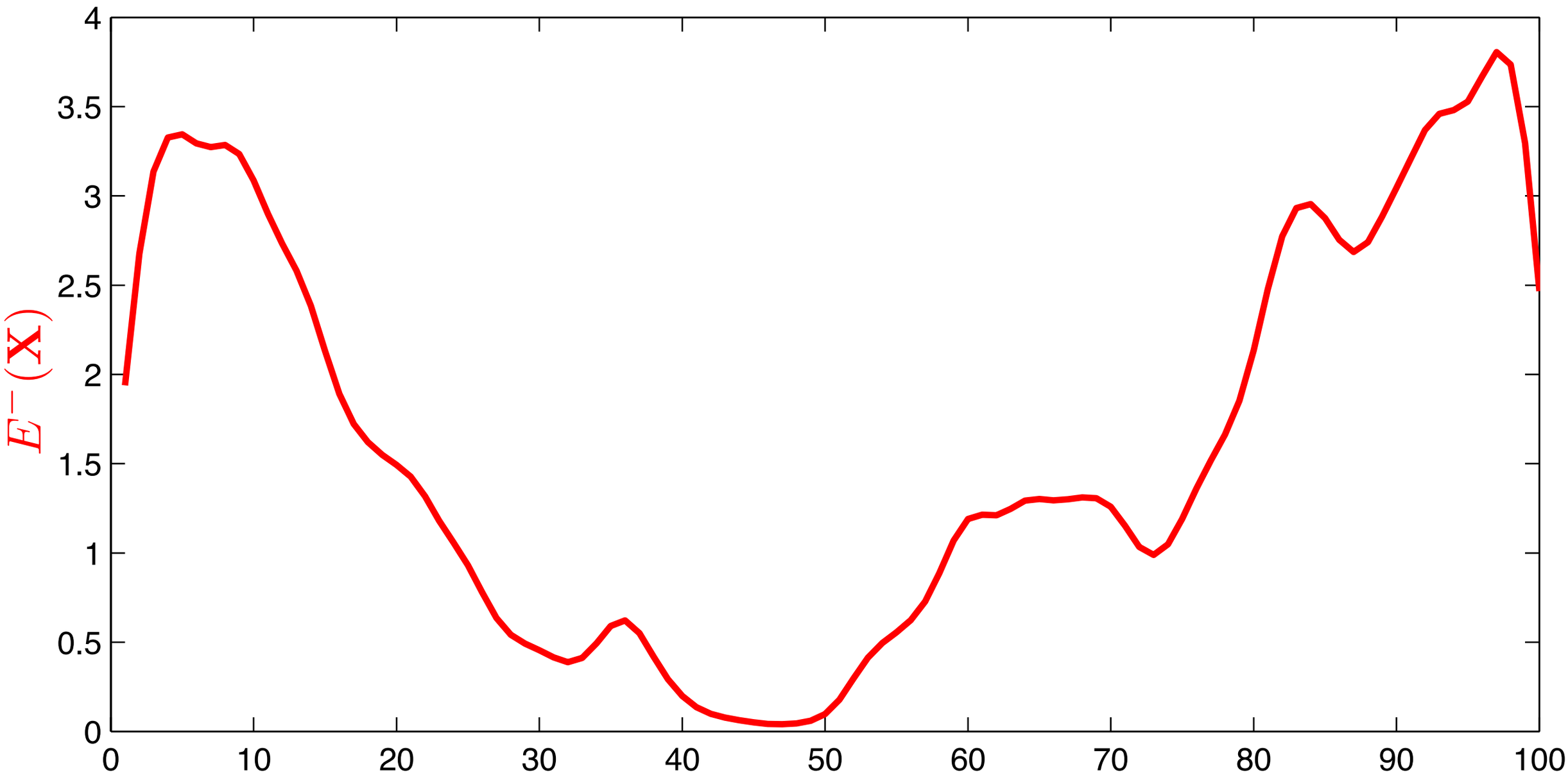
Optimization of Nonlinear Embedding

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$



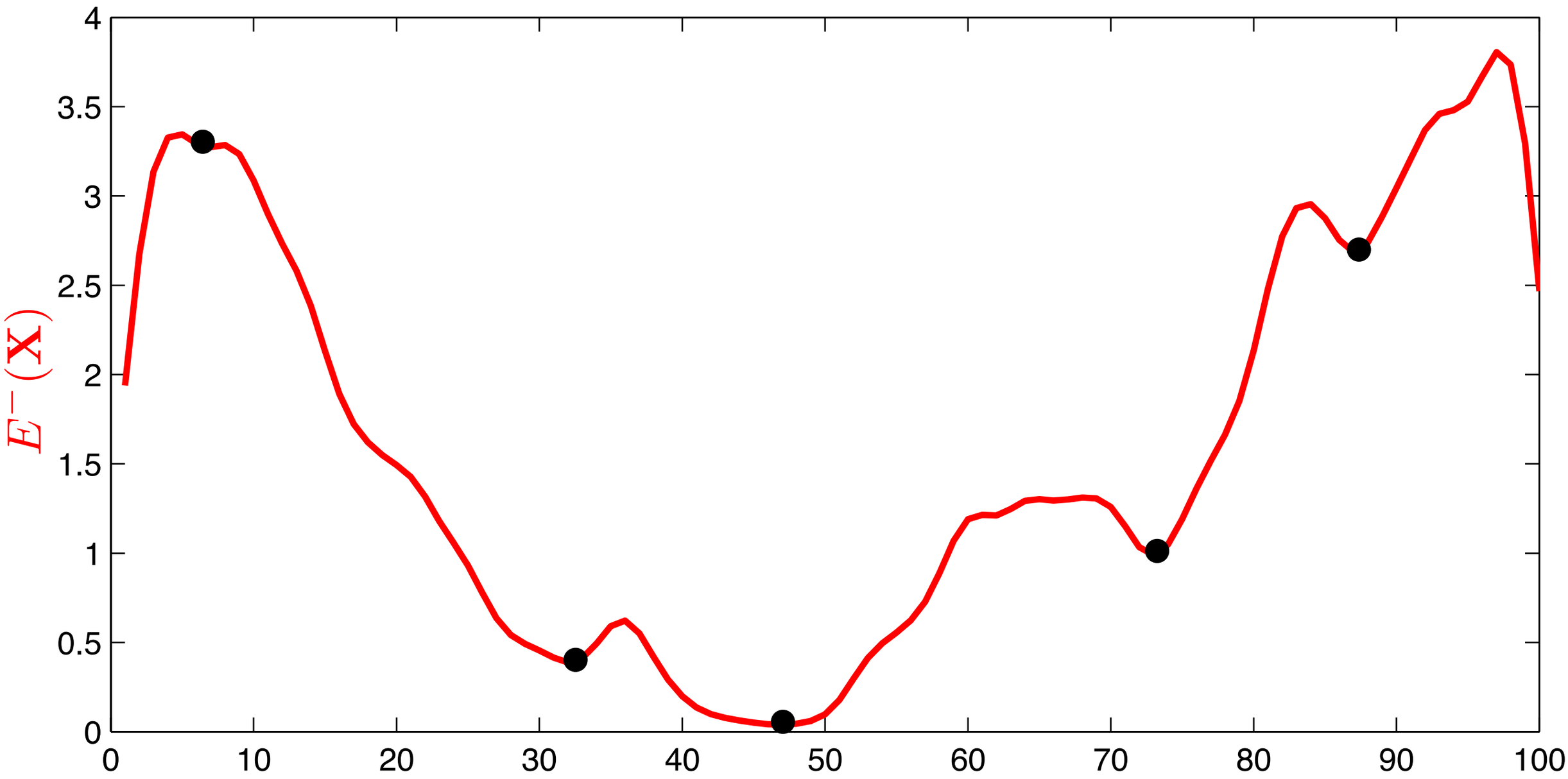
Optimization of Nonlinear Embedding

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$



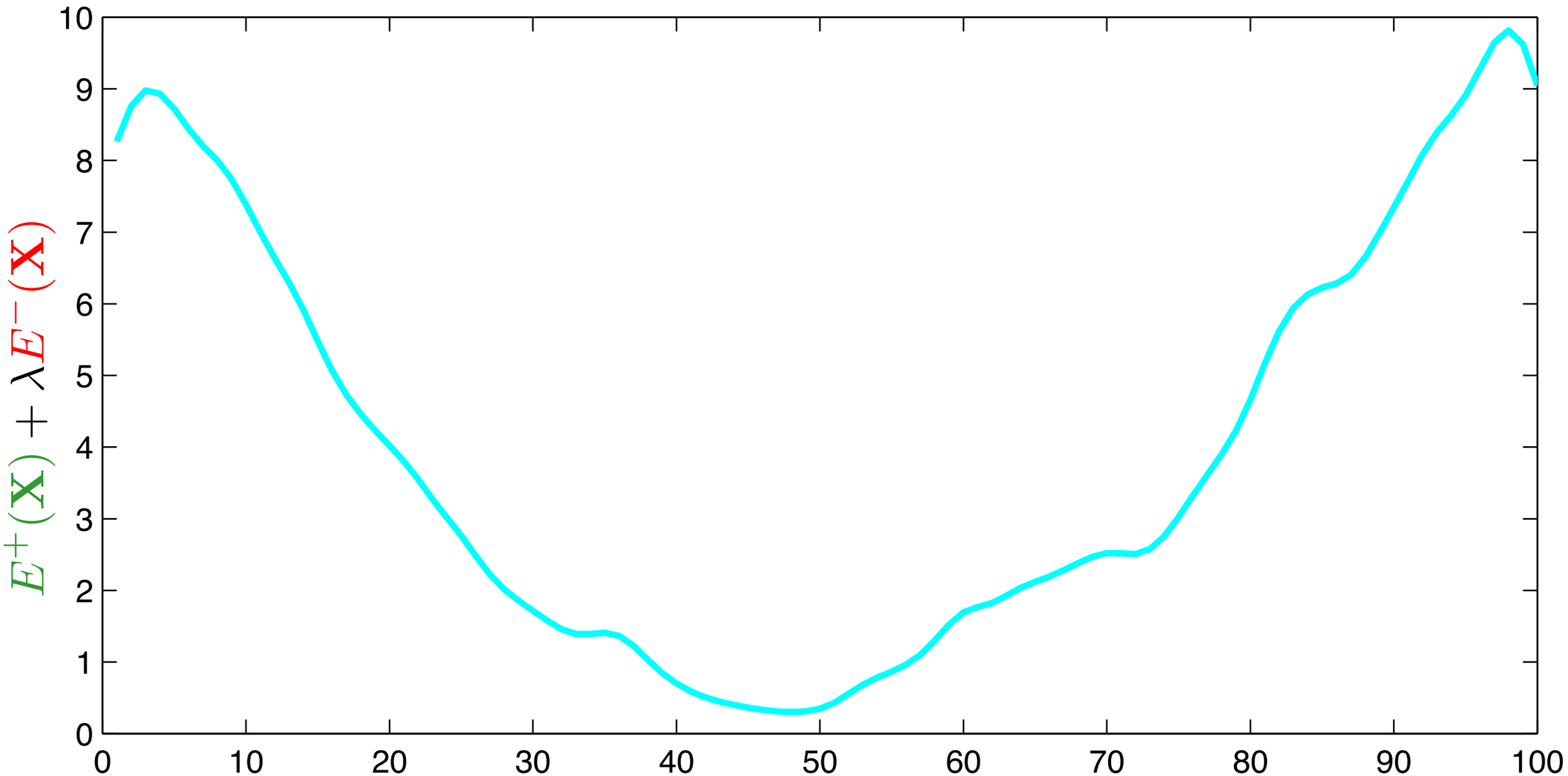
Optimization of Nonlinear Embedding

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$



Optimization of Nonlinear Embedding

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$



Optimization of Nonlinear Embedding

$$E(\mathbf{X}, \quad \lambda \geq 0$$

In **Nonlinear Embedding** methods, optimization should be done iteratively.