Large-Scale Methods for Nonlinear Manifold Learning

Max Vladymyrov
EECS, School of Engineering
University of California, Merced

PhD defense,
Merced, CA
November 28, 2014
Data is everywhere!
• 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

Arizona
Atlanta
Chicago
Cincinnati
Colorado
Florida
Houston
Los Angeles
Milwaukee
New York
Philadelphia
Pittsburgh
San Diego
San Francisco
St. Louis
Washington

ARI 0.475 1419 191 606 67
ATL 0.556 1453 184 534 92
CHI 0.488 1506 194 419 65
CIN 0.491 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 $X$

Three-dimensional dataset $Y$

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

Consider a dataset with 1000 handwritten digits:

![Handwritten digits image]

28x28
MNIST Handwritten digits

Consider a dataset with 1000 handwritten digits 2:

\[
\begin{pmatrix}
0 \\
0 \\
0 \\
\vdots \\
100 \\
104 \\
89 \\
90 \\
\vdots \\
0 \\
0 \\
0
\end{pmatrix} \in \mathbb{R}^{1 \times 784}
\]
MNIST Handwritten digits

Consider a dataset with 1000 handwritten digits:

High-dimensional dataset: \( Y \in \mathbb{R}^{1000 \times 784} \)
Number of points: \( N = 1000 \)
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

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

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)
Part I. Nonlinear dimensionality reduction

Nonlinear dimensionality reduction

- Spectral methods
- Stochastic Neighbor Embedding
- s-SNE
- t-SNE
- Elastic Embedding

Nonlinear Embedding Methods (NLE)

- Entropic affinities

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)

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

Part II
- Optimization using partial-Hessian
- Fast Multipole Methods
- Barnes-Hut method

Part III
- Locally Linear Landmarks
- Large-scale approx., using N-Body methods
Part I. Nonlinear dimensionality reduction

Nonlinear dimensionality reduction

- Spectral methods
- Stochastic Neighbor Embedding
- Nonlinear Embedding Methods (NLE)
- Entropic affinities

Part II

- Optimization using partial-Hessian
- Large-scale approx. using N-Body methods
  - Barnes-Hut method
  - Fast Multipole Methods

Part III

- Locally Linear Landmarks
Part I. Nonlinear dimensionality reduction

Nonlinear dimensionality reduction

Spectral methods

Stochastic Neighbor Embedding

s-SNE

t-SNE

Elastic Embedding

Entropic affinities

Nonlinear Embedding Methods (NLE)

Part II. Training of NLE

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

Spectral methods

Stochastic Neighbor Embedding

s-SNE

t-SNE

Elastic Embedding

Nonlinear Embedding Methods (NLE)

Entropic affinities

Part II. Training of NLE

Optimization using partial-Hessian

Part III. Scaling-up to large-scale datasets

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)

Part II. Training of NLE

- Optimization using partial-Hessian
- Locally Linear Landmarks

Part III. Scaling-up to large-scale datasets

- 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)

Part II. Training of NLE

- Optimization using partial-Hessian
- Locally Linear Landmarks

Part III. Scaling-up to large-scale datasets

- Large-scale approx. using N-Body methods
  - Barnes-Hut method
  - Fast Multipole Methods
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.
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 \( \mathbf{A} \).
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:

$$w_{nm} = \exp\left(-\frac{1}{2} \left\| (y_n - y_m)/\sigma \right\|^2 \right)$$
Gaussian affinity matrix
Gaussian affinity matrix
Gaussian affinity matrix: problem with $\sigma$

$$w_{nm} = \exp\left(-\frac{1}{2} \left\| (y_n - y_m) / \sigma \right\|^2 \right)$$

$\sigma = 1$
Gaussian affinity matrix: problem with $\sigma$

$$w_{nm} = \exp\left(-\frac{1}{2} \left\| \frac{y_n - y_m}{\sigma} \right\|^2 \right)$$

$\sigma = 1$
Gaussian affinity matrix

- Good $\sigma_n$ should be:
  - set *separately* for every data point,
  - take into account *whole distribution* of distances.
- $\sigma_n$ 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, $\sigma$ 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 $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^D$ 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 $\sigma$ given the perplexity $K$:

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

• We define entropic affinities as probabilities $p = (p_1, \ldots, p_N)$ for $\mathbf{y}$ with respect to $\sigma$. 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.
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.
Entropic affinities (computation)

\[ H(y_n, \sigma_n) = \log K \]
defines a root-finding problem for \( \sigma_n \).

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

\[ \downarrow \]

- We can solve for \( \sigma_n \) in just almost machine precision (\( tol = 10^{-15} \))
Entropic affinities (computation)

\[ H(y_n, \sigma_n) = \log K \] defines a root-finding problem for \( \sigma_n \).

• The problem is well defined for a Gaussian kernel for any \( \sigma_n \), and has a unique root for any \( K \in (0, N) \).

• There exists \( \sigma_n \) in constant time.

• We can use points.

• We can use \( \sigma_n \) with the bounds

• We can solve for \( \sigma_n \) in just almost machine precision (\( tol = 10^{-15} \))
Entropic affinities (computation)

\[ H(y_n, \sigma_n) = \log K \] defines a root-finding problem for \( \sigma_n \).

- The problem is well defined for a Gaussian kernel for any \( \sigma_n \), and has a unique root for any \( K \in (0, N) \).
- There exists tight bounds for the root \( \sigma_n \) that can be computed in constant time.
- We can use
  - with the bounds
  - We can use
  - points.
- We can solve for \( \sigma_n \) in just
  almost machine precision \((tol = 10^{-15})\)
Entropic affinities (computation)

\[ H(y_n, \sigma_n) = \log K \] defines a root-finding problem for \( \sigma_n \).

- The problem is well defined for a Gaussian kernel for any \( \sigma_n \) and has a unique root for any \( K \in (0, N) \).
- There exists tight bounds for the root \( \sigma_n \) 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 \( \sigma_n \) in just almost machine precision \((tol = 10^{-15})\)
Entropic affinities (computation)

$H(y_n, \sigma_n) = \log K$ defines a root-finding problem for $\sigma_n$.

- The problem is well defined for a Gaussian kernel for any $\sigma_n$ and has a unique root for any $K \in (0, N)$.
- There exists tight bounds for the root $\sigma_n$ 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 $\sigma_n$ 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 \( \sigma_n \).

- The problem is well defined for a Gaussian kernel for any \( \sigma_n \), and has a unique root for any \( K \in (0, N) \).
- There exists tight bounds for the root \( \sigma_n \) 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 \( \sigma_n \) 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)

solution to the previous problem
Entropic affinities (computation)

solution to the previous problem
Entropic affinities (computation)
Entropic affinities (computation)
Spectral methods

• Minimize

\[ \min_X \text{tr} \left( XAX^T \right) \quad \text{s.t.} \quad XBX^T = I \]

\begin{itemize}
  \item \( A_{N \times N} \): symmetric psd, contains information about the similarity between pairs of data points.
  \item \( B_{N \times N} \): symmetric pd (usually diagonal), set the scale of \( X \).
\end{itemize}

• Examples:
  \begin{itemize}
    \item Laplacian eigenmaps, \( A \) graph Laplacian,
    \item ISOMAP, \( A \) is given by a matrix of shortest distances,
    \item Kernel PCA, MDS, Locally Linear Embedding (LLE), etc.
  \end{itemize}

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

\[ X = U^T B^{-\frac{1}{2}}, \quad \text{where} \quad U = (u_1, \ldots, u_d) \quad \text{are the} \ d \ \text{trailing eigenvectors of the} \ N \times N \ \text{matrix} \]
\[ C = B^{-\frac{1}{2}} A B^{-\frac{1}{2}}. \]
Laplacian Eigenmaps (LE) (Belkin and Niyogi, '03)

- Minimize with respect to $X$

$$E_{LE}(X) = \frac{1}{2} \text{tr} (XLX^T) = \sum_{n,m=1}^{N} w_{nm} \|x_n - x_m\|^2$$

subject to translation and scale constraints

$$= w_{12} \|x_1 - x_2\|^2 + w_{13} \|x_1 - x_3\|^2 + \cdots + w_{nm} \|x_n - x_m\|^2 + \cdots$$

Intuition:

- if $w_{nm}$ is large (original points are located nearby to each other) $\Rightarrow$ place $x_n$ and $x_m$ 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 $L = \text{diag} \left( \sum_{n=1}^{N} w_{nm} \right) - W$. 
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.
Locality is preserved, scale is preserved!

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

Laplacian Eigenmaps
Stochastic neighbor embedding (SNE)  
(Hinton and Roweis, ’03)

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

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

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

• Minimize the KL-divergence between those probability distributions:

\[
E_{SNE}(\mathbf{X}) = \sum_{n=1}^{N} D(P_n \| Q_n) = \sum_{n,m=1}^{N} p_{n|m} \log \frac{p_{n|m}}{q_{n|m}}
\]
Stochastic neighbor embedding (SNE)
(Hinton and Roweis, ‘03)

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

\[ p_{m|n} = \frac{\exp(-\| (y_n - y_m)/\sigma^2 \|)}{\sum_{k \neq n} \exp(-\| (y_n - y_k)/\sigma^2 \|)} \]

\[ q_{m|n} = \frac{\exp(-\| x_n - x_m \|^2)}{\sum_{k \neq n} \exp(-\| x_n - x_k \|^2)} \]

• Minimize the KL-divergence between those probability distributions:

\[ E_{SNE}(X) = \sum_{n=1}^{N} D(P_n \| Q_n) = \sum_{n,m=1}^{N} p_{n|m} \log \frac{p_{n|m}}{q_{n|m}} \]
Variations of SNE

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

\[
p_{nm} = \frac{\exp(-\|y_n - y_m\|/\sigma^2)}{\sum_{k,l=1}^{N} \exp(-\|y_k - y_l\|/\sigma^2)} \quad q_{nm} = \frac{\exp(-\|x_n - x_m\|/\sigma^2)}{\sum_{k,l=1}^{N} \exp(-\|x_l - x_k\|/\sigma^2)}
\]

- 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:

\[
p_{nm} = \frac{\exp(-\|y_n - y_m\|/\sigma^2)}{\sum_{k,l=1}^{N} \exp(-\|y_k - y_l\|/\sigma^2)} \quad q_{nm} = \frac{(1 + \|x_n - x_m\|^2)^{-1}}{\sum_{k,l=1}^{N} (1 + \|x_l - x_k\|^2)^{-1}}
\]

- 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):

\[ E_{SNE}(X) = \sum_{n,m=1}^{N} p_{nm} \| x_n - x_m \|^2 + \sum_{n=1}^{N} \log \sum_{m \neq n}^{N} \exp(-\| x_n - x_m \|^2) \]

- 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:
  \[ w_{nm}^+ = \exp\left(-\frac{1}{2} ||(y_n - y_m)/\sigma||^2\right) \quad w_{nm}^- = ||y_n - y_m||^2 \]

- Minimize with respect to \( X \)
  \[ E_{EE}(X, \lambda) = \sum_{n,m=1}^{N} w_{nm}^+ ||x_n - x_m||^2 + \lambda \sum_{n,m=1}^{N} w_{nm}^- \exp(||-x_n - x_m||^2) \]

Intuition:
- if \( y_n \) and \( y_m \) are similar, first term will pull \( x_n \) and \( x_m \) together,
- if \( y_n \) and \( y_m \) are different, second term will push \( x_n \) and \( x_m \) 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

\[ E_{LE}(X) = \sum_{n,m=1}^{N} w_{nm} \| x_n - x_m \|^2 \quad \text{s.t. translation and scale constraints} \]

\[ E_{SNE}(X) = \sum_{n,m=1}^{N} p_{nm} \| x_n - x_m \|^2 + \sum_{n=1}^{N} \log \sum_{m \neq n}^{N} \exp(-\| x_n - x_m \|^2) \]

\[ E_{s-SNE}(X) = \sum_{n,m=1}^{N} p_{nm} \| x_n - x_m \|^2 + \log \sum_{n,m=1}^{N} \exp(-\| x_n - x_m \|^2) \]

\[ E_{t-SNE}(X) = \sum_{n,m=1}^{N} p_{nm} \log(1 + \| x_n - x_m \|^2) + \sum_{n,m=1}^{N} (1 + \| -x_n - x_m \|^2)^{-1} \]

\[ E_{EE}(X) = \sum_{n,m=1}^{N} w_{nm}^+ \| x_n - x_m \|^2 + \lambda \sum_{n,m=1}^{N} w_{nm}^- \exp(\| -x_n - x_m \|^2) \]
Nonlinear Embedding (NLE) methods

General embedding formulation:

\[ E(X, \lambda) = E^+(X) + \lambda E^-(X) \quad \lambda \geq 0 \]

\( E^+(X) \) is an attractive term:
- often quadratic,
- minimal with coincident points,
- defined usually on the sparse affinity (not all interactions are computed).

\( E^-(X) \) 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

Nonlinear dimensionality reduction

- Spectral methods
- Stochastic Neighbor Embedding
- s-SNE
- t-SNE
- Elastic Embedding

Nonlinear Embedding Methods (NLE)

- Entropic Affinities

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
  - Spectral methods
  - Stochastic Neighbor Embedding
  - s-SNE
  - t-SNE
  - Elastic Embedding

- Entropic Affinities

Nonlinear Embedding Methods (NLE)

Part II. Training of NLE

- Optimization using partial-Hessian

Part III

- Locally Linear Landmarks
- Large-scale approx. using N-Body methods
- Barnes-Hut method
- Fast Multipole Methods
NLE: Simple optimization algorithm

• Minimize objective function:
\[ E(X, \lambda) = E^+(X) + \lambda E^-(X) \quad \lambda \geq 0 \]

• Gradient Descent:
  ‣ compute the gradient
    \[ g \equiv \nabla E = 4X(L^+ - \lambda L^-) \]
  ‣ compute the direction
    \[ p_k = -g_k \]
  ‣ compute new iteration \( x_{k+1} \) with a line search:
    \[ x_{k+1} = x_k + \eta p_k \]
  ‣ 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:

\[ E(X, \lambda) = E^+(X) + \lambda E^-(X) \quad \lambda \geq 0 \]

• Gradient Descent:
  ‣ compute the gradient
    \[ g \equiv \nabla E = 4X(L^+ - \lambda L^-) \]
  ‣ compute the direction
    \[ p_k = -g_k \]
  ‣ compute new iteration \( x_{k+1} \)
    with a line search:
    \[ x_{k+1} = x_k + \eta p_k \]
  ‣ repeat till convergence.

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

• Minimize objective function:
  \[ E(X, \lambda) = E^+(X) + \lambda E^-(X) \quad \lambda \geq 0 \]

• Gradient Descent:
  ‣ compute the gradient
    \[ g \equiv \nabla E = 4X(L^+ - \lambda L^-) \]
  ‣ compute the direction
    \[ p_k = -g_k \]
  ‣ compute new iteration \( x_{k+1} \)
    with a line search:
    \[ x_{k+1} = x_k + \eta p_k \]
  ‣ repeat till convergence.

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

• Minimize objective function:
\[ E(X, \lambda) = E^+(X) + \lambda E^-(X) \quad \lambda \geq 0 \]

• Gradient Descent:
  ‣ compute the gradient 
    \[ g \equiv \nabla E = 4X(L^+ - \lambda L^-) \]
  ‣ compute the direction 
    \[ p_k = -g_k \]
  ‣ compute new iteration \( x_{k+1} \) with a line search:
    \[ x_{k+1} = x_k + \eta p_k \]
  ‣ 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:
\[ E(X, \lambda) = E^+(X) + \lambda E^-(X) \quad \lambda \geq 0 \]

• Gradient Descent:
  ‣ compute the gradient
    \[ g \equiv \nabla E = 4X(L^+ - \lambda L^-) \]
  ‣ compute the direction
    \[ p_k = -g_k \]
  ‣ compute new iteration \( x_{k+1} \) with a line search:
    \[ x_{k+1} = x_k + \eta p_k \]
  ‣ 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:
  \[ E(X, \lambda) = E^+(X) + \lambda E^-(X) \quad \lambda \geq 0 \]

• Gradient Descent:
  ‣ compute the gradient
    \[ g = \nabla E = 4X(L^+ - \lambda L^-) \]
  ‣ compute the direction
    \[ p_k = -g_k \]
  ‣ compute new iteration \( x_{k+1} \)
    with a line search:
    \[ x_{k+1} = x_k + \eta p_k \]
  ‣ repeat till convergence.

• Other gradient-based optimization methods are applicable:
  L-BFGS, Conjugate Gradient, etc.
Including second-order information

Consider the following method. For every iteration $k$:

- choose any positive definite $B_k$,
- solve a linear system $B_k p_k = -g_k$ for a search direction $p_k$,
- use line search to find a step size $\eta$ for the next iteration

\[ x_{k+1} = x_k + \eta p_k \] (e.g. with backtracking line search).

Convergence is guaranteed!
How to choose good $B_k$?

$B_k = I$ (gradient descent) vs. $B_k = \nabla^2 E$ (Newton’s method) 

- $B_k$ should be positive definite (pd),
- fast to compute and solve the linear system $B_k p_k = -g_k$,
- contain as much Hessian information as possible,
- more Hessian information results in a faster convergence rate.

We want $B_k$:
- positive definite (pd),
- fast to compute and solve the linear system $B_k p_k = -g_k$, 
- contain as much Hessian information as possible,
The Hessian of Nonlinear Embedding

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

$$\nabla^2 E = 4 L^+ \otimes I_{d \times d}$$

$$- 4\lambda L^- \otimes I_{d \times d}$$

$$+ 8L^{xx}$$

$$- 16\lambda \text{vec}(XL^q)\text{vec}(XL^q)^T$$

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

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

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

$$\nabla^2 E = 4L^+ \otimes I_{d \times d}$$

$$- 4 \lambda L^- \otimes I_{d \times d}$$

$$+ 8L^{xx}$$

$$- 16 \lambda \text{vec}(XL^q) \text{vec}(XL^q)^T$$

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

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

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

$$\nabla^2 E = 4L^+ \otimes I_{d \times d}$$

$$- 4\lambda L^- \otimes I_{d \times d}$$

$$+ 8L^{xx}$$

$$- 16\lambda \text{vec}(XL^q)\text{vec}(XL^q)^T$$

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

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

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

$$
\nabla^2 E = 4L^+ \otimes I_{d \times d}
- 4\lambda L^- \otimes I_{d \times d}
+ 8L^{xx}
- 16\lambda \text{vec}(XL^q)\text{vec}(XL^q)^T
$$

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

- $L^+, L^-, L^q$: constant for Gaussian kernel \(\Rightarrow\) psd.
- $L^{xx}$: depends on the embedding $X$. Some parts are psd.
The Hessian of Nonlinear Embedding

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

$$\nabla^2 E = 4L^+ \otimes I_{d \times d} - 4\lambda L^- \otimes I_{d \times d} + 8L^{xx} - 16\lambda vec(XL^q)vec(XL^q)^T$$

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

• $L^+, L^-, L^q$: constant for Gaussian kernel $\Rightarrow$ psd.
• $L^{xx}$: depends on the embedding $X$. Some parts are psd.
The Hessian of Nonlinear Embedding

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

$$\nabla^2 E = 4L^+ \otimes I_{d \times d} - 4\lambda L^- \otimes I_{d \times d} + 8L^{xx} - 16\lambda \text{vec}(XL^q)\text{vec}(XL^q)^T$$

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

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

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

$$\nabla^2 E = 4L^+ \otimes I_{d \times d} - 4\lambda L^- \otimes I_{d \times d} + 8L^{xx} - 16\lambda\text{vec}(XL^q)^T \text{vec}(XL^q)^T$$

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

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

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

$$\nabla^2 E = 4L^+ \otimes I_{d \times d}$$
$$- 4\lambda L^- \otimes I_{d \times d}$$
$$+ 8L^{xx}$$
$$- 16\lambda \text{vec}(XL^q)\text{vec}(XL^q)^T$$

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

- $L^+, L^-, L^q$: constant for Gaussian kernel $\Rightarrow$ psd.
- $L^{xx}$: depends on the embedding $X$. Some parts are psd.
The spectral direction \cite{ 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 \( (\text{Vladymyrov and Carreira-Perpiñán, ’12}) \)

\[ B_k = 4L^+ \otimes I_{d \times d} \] is a convenient Hessian approximation.

We wanted \( B_k \):

- positive definite (pd),
- fast to compute and solve the linear system \( B_k p_k = -g_k \)
  - block-diagonal and has \( d \) blocks of \( N \times N \) graph Laplacian \( 4L^+ \)
  - constant for Gaussian kernel \( B = B_k \). For other kernels we can fix it as some intermediate \( x \),
  - linear system can be solved efficiently using Cholesky factorization of \( B \),
  - (further) sparsify the weights \( W^+ \) with a \( \kappa \)-NN graph.
- contain as much Hessian information as possible,
The spectral direction (Vladymyrov and Carreira-Perpiñán, ’12)

$B_k = 4L^+ \otimes I_{d \times d}$ is a convenient Hessian approximation.

We wanted $B_k$:

✓ positive definite (pd),
✓ fast to compute and solve the linear system $B_k p_k = -g_k$
  ‣ block-diagonal and has $d$ blocks of $N \times N$ graph Laplacian $4L^+$
  ‣ constant for Gaussian kernel $B = B_k$. For other kernels we can fix it as some intermediate $\mathbf{x}$,
  ‣ linear system can be solved efficiently using Cholesky factorization of $B$,
  ‣ (further) sparsify the weights $W^+$ with a $\kappa$-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 \( \kappa \)-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 \( \mathbf{E} \) 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 \( \kappa \)-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)

<table>
<thead>
<tr>
<th></th>
<th>Cost per iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective function</td>
<td>$O(N^2d)$</td>
</tr>
<tr>
<td>Gradient</td>
<td>$O(N^2d)$</td>
</tr>
<tr>
<td>Spectral direction</td>
<td>$O(N\kappa d)$</td>
</tr>
</tbody>
</table>

- 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, ...).
Optimization methods compared:
- Gradient descent
- Fixed-point iterations
- The spectral direction
- L-BFGS

\[ B_k = I \]
\[ B_k = 4D^+ \otimes I_{d \times d} \]
\[ B_k = 4L^+ \otimes I_{d \times d} \]
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.
Comparing **fixed-point iteration** to the spectral diction for 20,000 MNIST digits in one hour of optimization.
Comparing fixed-point iteration to the spectral direction 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

Nonlinear dimensionality reduction

Spectral methods

Stochastic Neighbor Embedding

s-SNE

t-SNE

Elastic Embedding

Nonlinear Embedding Methods (NLE)

Entropic Affinities

Part II. Training of NLE

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

Spectral methods

Stochastic Neighbor Embedding

s-SNE

t-SNE

Elastic Embedding

Nonlinear Embedding Methods (NLE)

Part II. Training of NLE

Optimization using partial-Hessian

Part III. Scaling-up to large-scale datasets

Locally Linear Landmarks

Large-scale approx. using N-Body methods

Barnes-Hut method

Fast Multipole Methods
Part I. Nonlinear dimensionality reduction

- Nonlinear dimensionality reduction
- Spectral methods
- Stochastic Neighbor Embedding
- s-SNE
- t-SNE
- Elastic Embedding
- Entropic Affinities

Nonlinear Embedding Methods (NLE)

Part II. Training of NLE

- Optimization using partial-Hessian

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:
\[
\min_X \text{tr} \left( XAX^T \right) \quad \text{s.t.} \quad XBX^T = I
\]

• Solution is unique and can be found in closed form for by the eigenvectors of \( N \times N \) matrix constructed from \( A \) and \( B \).

With large \( N \), solving this eigenproblem is infeasible even if \( A \) and \( B \) are sparse.

Original dataset \( Y \) \hspace{5cm} Affinity matrix \hspace{5cm} Learned \( X \)
Learning with landmarks

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 → Learned $\mathbf{X}$

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 et al, 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 $\tilde{\mathbf{X}} = \tilde{\mathbf{x}}_1 \ldots, \tilde{\mathbf{x}}_L$.
  ‣ …and bad prediction for the non-landmarks.
Locally Linear Landmarks (LLL)  
(Vladymyrov and Carreira-Perpiñán, ’13)

• Assume each projection is a locally function of the landmarks:
  \[ x_n = \sum_{l=1}^{L} z_{ln} \tilde{x}, \quad n = 1, \ldots, N \quad \Rightarrow \quad X = \tilde{X}Z \]

• 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 \( \tilde{X} \):
  \[
  \min_{\tilde{X}} \text{tr} \left( \tilde{X} \tilde{A} \tilde{X}^T \right) \quad \text{s.t.} \quad \tilde{X} \tilde{B} \tilde{X}^T = I
  \]
  with reduced affinities \( \tilde{A} = ZAZ^T, \tilde{B} = ZBZ^T \).

• After \( \tilde{X} \) is found, the non-landmarks are predicted as \( X = \tilde{X}Z \) (out-of-sample mapping).

• Advantages over Nyström method:
  ‣ The reduced affinities \( \tilde{A} = ZAZ^T \) 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):
  \[ A \Rightarrow a_{ij} \Rightarrow \text{path } i \rightarrow j \]
- LLL (reduced affinities):
  \[ \tilde{A} = ZAZ^T \Rightarrow \tilde{a}_{ij} = \sum_{n,m=1}^{N} z_{in}a_{nm}z_{jm} \Rightarrow \text{path } i \rightarrow n \rightarrow m \rightarrow j \ \forall n, m \]

So landmarks \( i \) and \( j \) can be farther apart and still be connected along the manifold.
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.
Part I. Nonlinear dimensionality reduction

Nonlinear dimensionality reduction

Spectral methods
Stochastic Neighbor Embedding
s-SNE

t-SNE
Elastic Embedding

Part II. Training of NLE

Optimization using partial-Hessian

Part III. Scaling-up to large-scale datasets

Locally Linear Landmarks

Large-scale approx. using N-Body methods
    Barnes-Hut method
    Fast Multipole Methods
Part I. Nonlinear dimensionality reduction

Nonlinear dimensionality reduction

Spectral methods

Stochastic Neighbor Embedding

s-SNE

t-SNE

Elastic Embedding

Nonlinear Embedding Methods (NLE)

Entropic Affinities

Part II. Training of NLE

Optimization using partial-Hessian

Part III. Scaling-up to large-scale datasets

Locally Linear Landmarks

Large-scale approx. using N-Body methods

Barnes-Hut method

Fast Multipole Methods
Optimization of NLE

For every iteration $k$:

- compute the gradient $G_k$,
- find search direction $P_k$,
- use line search to find a step size $\eta$ for the next iteration:

$$X_{k+1} = X_k + \eta P_k$$

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:

\[ EE(X) = \sum_{n,m=1}^{N} w_{nm} \| x_n - x_m \|^2 + \lambda \sum_{n=1}^{N} S(x_n) \]
\[ G_{EE}(X) = 4XL - 4\lambda X \text{diag} (S(X)) + 4\lambda S^{xx}(X) \]

with

\[ S(x_n) = \sum_{m=1}^{N} e^{-\|x_n - x_m\|^2} \]
\[ S^{xx}(x_n) = \sum_{m=1}^{N} x_m e^{-\|x_n - x_m\|^2} \]

Computing \( S^{xx}(x_n) \) and \( S(x_n) \) for every \( n = 1, \ldots, N \) is \( O(N^2) \).

No matter how fast is the optimization, it just decreases the number of iterations required for convergence. Each iteration is still \( O(N^2) \) 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).

\[ S(x_n) = \sum_{m=1}^{N} e^{-\|x_n - x_m\|^2} \]

\[ S^x(x_n) = \sum_{m=1}^{N} x_m e^{-\|x_n - x_m\|^2} \]

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).

\[ S(x_n) = \sum_{m=1}^{N} e^{-\|x_n - x_m\|^2} \quad S^x(x_n) = \sum_{m=1}^{N} x_m e^{-\|x_n - x_m\|^2} \]

- 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 $x_n$ and others points:

• Build a tree around $X$
• 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 $O(N \log N)$

• 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 $x_n$ and others points:

- Build a tree around $X$.

- 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 $O(N \log N)$

- 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 $x_n$ and others points:

- Build a tree around $X$.
- 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 $O(N \log N)$

- 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 $x_n$ and others points:

- Build a tree around $X$.
- 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 $O(N \log N)$

- 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 $x_n$ and others points:
- Build a tree around $X$.
- 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 $O(N \log N)$
- 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 $x_n$ and others points:

• Build a tree around $X$.

• 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 $O(N \log N)$.

• 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 $O(N \log N)$.
- Problems:
  ‣ do not scale well with dimensions of latent space,
  ‣ error bounds are usually hard to derive.
Fast multipole methods  (Greengard and Rokhlin '87)

Properties:

😊 Time complexity $O(N)$.

😊 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:

\[ Q(x_n) = \sum_{m=1}^{N} q_m K(\|(x_n - x_m)/\sigma\|^2) \]

The idea is to do a series expansion of the kernel \( K \), such that the sum decouples over \( x_n \) and \( x_m \):

\[ K(\|(x_n - x_m)/\sigma\|^2) = \sum_{\alpha \geq 0} f_{\alpha}(x_n) g_{\alpha}(x_m) \]

using multi-index notation \( \alpha \geq 0 \Rightarrow \alpha_1, \ldots, \alpha_d \geq 0 \).
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 ⇒ 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 ⇒ exactly,
   • a lot of points ⇒ 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 ⇒ 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 ⇒ exactly,
   • a lot of points ⇒ 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 ⇒ 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 ⇒ exactly,
   • a lot of points ⇒ 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 ⇒ 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 ⇒ exactly,
   • a lot of points ⇒ 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 ⇒ 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 ⇒ exactly,
   - a lot of points ⇒ 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 ⇒ 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 ⇒ exactly,
   - a lot of points ⇒ 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 \cite{Greengard, 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 \hspace{1em} (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 ⇒ 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 ⇒ exactly,
   • a lot of points ⇒ 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 ⇒ 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 ⇒ exactly,
   • a lot of points ⇒ use center of mass.
Application of N-Body to NLE

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

\[ S(x_n) = \sum_{m=1}^{N} K(||x_n - x_m||^2) \quad S^x(x_n) = \sum_{m=1}^{N} x_m K(||x_n - x_m||^2) \]

• The objective function and the gradient of EE:

\[ E_{EE}(X) = \sum_{n,m=1}^{N} w_{nm} ||x_n - x_m||^2 + \lambda \sum_{n=1}^{N} S(x_n) \]
\[ G_{EE}(X) = 4XL - 4\lambda X \text{ diag } (S(X)) + 4\lambda S^x(X) \]

• Given \( S(x_n) \) and \( S^x(x_n) \), 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 \textit{per minute of runtime}.
Experiments: 1 000 000 handwritten digits

- Objective function vs. iterations
- Objective function vs. runtime

- Algorithms used:
  - GD; FMM
  - FP; FMM
  - LBFGS; FMM
  - GD; BH
  - FP; BH
  - LBFGS; BH

- Graphs show performance metrics for different algorithms and data sets.
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.

• Entropic affinities produce high-quality affinities with almost a closed-form solution based on a single global intuitive parameter $K$.

• 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


• 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.


Software

• Code for all the methods presented is available online:
  [https://eng.ucmerced.edu/people/vladymyrov](https://eng.ucmerced.edu/people/vladymyrov)
Papers


• 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.


Software

• Code for all the methods presented is available online: [https://eng.ucmerced.edu/people/vladymyrov](https://eng.ucmerced.edu/people/vladymyrov)

Thank you!
Questions?
Model the effect of the approximate gradient

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

Mean of the absolute error:
\[
\langle E(X + \epsilon) - E(X) \rangle = \frac{1}{2}\sigma^2 \text{tr}\left(\nabla^2 E(X)\right) + \mathcal{O}(\sigma^4)
\]

We have qualitative predictions:

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

Positive curvature (harm the approximation)

Negative curvature (benefit from the approximation)
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 finds directions of **biggest variation** of the data and **projects** the data accordingly.
Given a high-dimensional dataset, PCA finds directions of \textit{biggest variation} of the data and \textit{projects} the data accordingly.
Given a high-dimensional dataset, PCA finds directions of **biggest variation** of the data and **projects** the data accordingly.
Principal Component Analysis (PCA)

Given a high-dimensional dataset, PCA finds directions of \textit{biggest variation} of the data and \textit{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.
Given a high-dimensional dataset, PCA finds directions of biggest variation of the data and projects the data accordingly.

**PCA** works only if data is linearly separable!
Rotational sequences
Rotational sequences

s-SNE

t-SNE
Optimization of Nonlinear Embedding

\[ E(X, \lambda) = E^+(X) + \lambda E^-(X) \quad \lambda \geq 0 \]
Optimization of Nonlinear Embedding

\[ E(X, \lambda) = E^+(X) + \lambda E^-(X) \quad \lambda \geq 0 \]
Optimization of Nonlinear Embedding

\[ E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0 \]
Optimization of Nonlinear Embedding

\[ E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0 \]
In Nonlinear Embedding methods, optimization should be done iteratively.