Constructing better and scalable affinity matrices for machine leaning algorithms

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

EECS, UC Merced
January 24, 2014

In collaboration with Miguel Á. Carreira-Perpiñán
Constructing better and scalable affinity matrices for machine learning algorithms

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

EECS, UC Merced
January 24, 2014

In collaboration with Miguel Á. Carreira-Perpiñán
Constructing better and scalable affinity matrices for machine leaning algorithms

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

EECS, UC Merced
January 24, 2014

In collaboration with Miguel Á. Carreira-Perpiñán
Affinity matrices

Given high-dimensional data points \( \mathbf{Y}_{D \times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N) \). Affinity matrix \( \mathbf{A} \in \mathbb{R}^{N \times N} \) provides a similarity measure between points in \( \mathbf{Y} \).

Used in:

- Dimensionality reduction algorithms:
  - Laplacian Eigenmaps, Stochastic Neighbor Embedding, t-SNE.
- Clustering:
  - Mean-Shift, Spectral clustering.
- Semi-supervised learning
- etc.
Use in dimensionality reduction

Given high-dimensional data points $\mathbf{Y}_{D \times N} = (y_1, \ldots, y_N)$. 

1. Convert data points to a $N \times N$ affinity matrix $\mathbf{A}$.
2. Find low-dimensional coordinates $\mathbf{X}_{d \times N} = (x_1, \ldots, x_N)$, so that their similarity is as close as possible to $\mathbf{A}$.
COIL-20 Rotational sequences

10 objects:

72 images per object:

High-dimensional dataset: \( Y \in \mathbb{R}^{720 \times 16384} \)
Number of points: \( N = 720 \)
Number of dimensions: \( D = 16384 \)
Reduction space: \( d = 2 \)
COIL-20 Rotational sequences

10 objects:

72 images per object:

128 images per object:

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

Number of points: \( N = 720 \)

Number of dimensions: \( D = 16\,384 \)

Reduction space: \( d = 2 \)
Gaussian affinity matrix (subset)
Gaussian affinity matrix (subset)
Low-dimensional embedding

visualized by Elastic Embedding
(Carreira-Perpiñán, 2010)
Part 1.

Constructing better affinities
Gaussian affinity matrix

- Affinity matrix $W \in \mathbb{R}^{N \times N}$ is a Gaussian, with a scale parameter $\sigma$:
  \[ w_{nm} = \exp\left(-\frac{1}{2} \left\| (y_n - y_m) / \sigma \right\|^2 \right) \]

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

- Property:
  - this affinity matrix enforces locality.
Gaussian affinity matrix
Gaussian affinity matrix
Gaussian affinity matrix: problem with $\sigma$

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

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

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

$\sigma = 1$
Problems with Gaussian affinities

• 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).
Rule of thumb: $\sigma = \text{distance to 7th neighbor}$
Rule of thumb: $\sigma = \text{distance to 7th neighbor}$
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 \( \sigma \) (or precision \( \beta = \frac{1}{2\sigma^2} \)) given the perplexity \( K \):

\[
H(\mathbf{x}, \beta) = \log K
\]

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

Radius of the circle corresponds to $\sigma$
Entropic affinities: properties

\[ H(y_n, \beta_n) \equiv - \sum_{n=1}^{N} p_n(y_n, \beta_n) \log(p_n(y_n, \beta_n)) = \log K \]

• This is a 1D root-finding problem or an inversion problem \( \beta_n = H_{y_n}^{-1}(\log K) \).
• We can prove that:
  ‣ The root-finding problem is well defined for a Gaussian kernel for any \( \beta_n > 0 \), and has a unique root for any \( K \in (0, N) \).
  ‣ The inverse is a uniquely defined continuously differentiable function for all \( y_n \in \mathbb{R}^N \) and \( K \in (0, N) \).
Entropic affinities: bounds

The bounds $[\beta_L, \beta_U]$ for every $K \in (0, N)$ and $x_n \in \mathbb{R}^N$:

$$\beta_L = \max \left( \frac{N \log \frac{N}{K}}{(N - 1) \Delta_2^2}, \sqrt{\log \frac{N}{K}} \right),$$

$$\beta_U = \frac{1}{\Delta_2^2} \log \left( \frac{p_1}{1 - p_1} (N - 1) \right),$$

where $\Delta_2^2 = d_2^2 - d_1^2$, $\Delta_N^2 = d_N^2 - d_1^2$, and $p_1$ is a unique solution of the equation

$$2(1 - p_1) \log \frac{N}{2(1 - p_1)} = \log \left( \min(\sqrt{2N}, K) \right)$$

The bounds are computed in $O(1)$ for each point.
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 a previous problem
Entropic affinities (computation)

Solution to a previous problem
Entropic affinities (computation)
Entropic affinities (computation)
Entropic affinities (computation)

Missing parts:
1. Robust and fast root-finding algorithm.
2. Good initialization.
1. Computation of $\beta_n$ (root-finding)

<table>
<thead>
<tr>
<th>Methods</th>
<th>Convergence order</th>
<th>Derivatives needed</th>
<th>Number of $O(N)$ evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Derivative-free</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bisection</td>
<td>linear</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brent</td>
<td>linear</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Ridder</td>
<td>quadratic</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>Derivative-based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Newton</td>
<td>quadratic</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Halley</td>
<td>cubic</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Euler</td>
<td>cubic</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- The cost of the objective function evaluation and each of derivative is $O(N)$.
- Derivative-free methods above generally converge globally. They work by iteratively shrinking an interval bracketing the root.
- Derivative-free methods have higher convergence order, but may diverge.
Robustified root-finding algorithm

- We embed the derivative-based algorithm into bisection loop for global convergence.
- We run the following algorithm for each $y_n \in \{y_1, \ldots, y_N\}$

**Input:** initial $\beta$, perplexity $K$, distances $d_1^2, \ldots, d_N^2$, bounds $B$.

```plaintext
while true do
  for $k = 1$ to maxit do
    compute $\beta$ using a derivative-based method
    if tolerance achieved return
    if $\beta \notin B$ exit for loop
    update $B$
  end for
  compute $\beta$ using bisection
  update $B$
end while
```

![Graph showing the function $H(\beta)$ and $\log(K)$ against $\log(\beta)$ with iterations marked.](image)
Robustified root-finding algorithm

• We embed the derivative-based algorithm into bisection loop for global convergence
• We run the following algorithm for each \( y_n \in \{y_1, \ldots, y_N\} \)

Input: initial \( \beta \), perplexity \( K \), distances \( d_1^2, \ldots, d_N^2 \), bounds \( B \).

while true do
  for \( k = 1 \) to maxit do
    compute \( \beta \) using a derivative-based method
    if tolerance achieved return
    if \( \beta \notin B \) exit for loop
    update \( B \)
  end for
  compute \( \beta \) using bisection
  update \( B \)
end while
Robustified root-finding algorithm

- We embed the derivative-based algorithm into bisection loop for global convergence.
- We run the following algorithm for each $y_n \in \{y_1, \ldots, y_N\}$

**Input:** initial $\beta$, perplexity $K$, distances $d_1^2, \ldots, d_N^2$, bounds $\mathcal{B}$. 

```plaintext
while true do
    for $k = 1$ to $\text{maxit}$ do
        compute $\beta$ using a derivative-based method
        if tolerance achieved return
        if $\beta \notin \mathcal{B}$ exit for loop
        update $\mathcal{B}$
    end for
    compute $\beta$ using bisection
    update $\mathcal{B}$
end while
```

- We embed the derivative-based algorithm into bisection loop for global convergence.
- We run the following algorithm for each $y_n \in \{y_1, \ldots, y_N\}$

**Input:** initial $\beta$, perplexity $K$, distances $d_1^2, \ldots, d_N^2$, bounds $\mathcal{B}$. 

```plaintext
while true do
    for $k = 1$ to $\text{maxit}$ do
        compute $\beta$ using a derivative-based method
        if tolerance achieved return
        if $\beta \notin \mathcal{B}$ exit for loop
        update $\mathcal{B}$
    end for
    compute $\beta$ using bisection
    update $\mathcal{B}$
end while
```
Robustified root-finding algorithm

- We embed the derivative-based algorithm into bisection loop for global convergence
- We run the following algorithm for each $y_n \in \{y_1, \ldots, y_N\}$

**Input:** initial $\beta$, perplexity $K$, distances $d_1^2, \ldots, d_N^2$, bounds $B$.

while true do
  for $k = 1$ to maxit do
    compute $\beta$ using a derivative-based method
    if tolerance achieved return
    if $\beta \notin B$ exit for loop
    update $B$
  end for
  compute $\beta$ using bisection
  update $B$
end while
Robustified root-finding algorithm

- We embed the derivative-based algorithm into bisection loop for global convergence
- We run the following algorithm for each $\mathbf{y}_n \in \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$

**Input:** initial $\beta$, perplexity $K$, distances $d_1^2, \ldots, d_N^2$, bounds $\mathcal{B}$.

while true do
  for $k = 1$ to maxit do
    compute $\beta$ using a derivative-based method
    if tolerance achieved return
    if $\beta \notin \mathcal{B}$ exit for loop
    update $\mathcal{B}$
  end for
  compute $\beta$ using bisection
  update $\mathcal{B}$
end while
2. Initialization of $\beta_n$

1. Simple initialization:
   • middle of the bounds,
   • distance to $k$th nearest neighbor.
   
   Typically far from root and require more iterations.

2. Each new $\beta_n$ is initialized from the solution to its predecessor:
   • sequential order;
   • tree order.

   We need to find an order that is correlated with the behavior of $\beta_n$. 

2. Initialization of $\beta_n$

1. Simple initialization:
   • middle of the bounds,
   • distance to $k$th nearest neighbor.
   Typically far from root and require more iterations.

2. Each new $\beta_n$ is initialized from the solution to its predecessor:
   • sequential order;
   • tree order.
   We need to find an order that is correlated with the behavior of $\beta_n$. 
2. Initialization of $\beta_n$

1. Simple initialization:
   - middle of the bounds,
   - distance to $k$th nearest neighbor.
   Typically far from root and require more iterations.

2. Each new $\beta_n$ is initialized from the solution to its predecessor:
   - sequential order;
   - tree order.
   We need to find an order that is correlated with the behavior of $\beta_n$. 
2. Initialization of $\beta_n$

1. Simple initialization:
   • middle of the bounds,
   • distance to $k$th nearest neighbor.
   Typically far from root and require more iterations.

2. Each new $\beta_n$ is initialized from the solution to its predecessor:
   • sequential order;
   • tree order.
   We need to find an order that is correlated with the behavior of $\beta_n$. 
2. Initialization of $\beta_n$

1. Simple initialization:
   - middle of the bounds,
   - distance to $k$th nearest neighbor.
   Typically far from root and require more iterations.

2. Each new $\beta_n$ is initialized from the solution to its predecessor:
   - sequential order;
   - tree order.
   We need to find an order that is correlated with the behavior of $\beta_n$. 
Sequential or tree order

• $D_k$ density strategy: for the fixed entropy value, $\beta$ is larger in dense regions and smaller in sparser ones.
  ‣ Use nearest neighbor density density estimate.
  ‣ $\beta_n$ is proportional to the distance to $k$th nearest neighbor of $x_n$.

• MST, local strategy: nearby points have similar $\beta$ values.
  ‣ Build a MST around the data.
  ‣ Process the points in level-order, so parents are solved for before children.
Experimental evaluation: setup

We set the perplexity to $K = 30$ and the tolerance to $10^{-10}$.

Initializations:
- "oracle": processes the points in the order of their true $\beta$ values,
- MST: local-based order;
- $D_K$: density-based order;
- bounds: initialize from the midpoint of the brackets,
- random: initialize from $x_n$ at random.

Root-finding methods:
- Derivative-free: Bisection, Brent, Ridder.
- Derivative-based: Newton, Euler, Halley.
Experimental evaluation: Lena

Bisection: $> 10$ min.
Our method: $1$ min.
Computing just the affinities given $\beta$s: $20$ s.
Experimental evaluation: image

512 × 512 Lena image. Each data point is a pixel represented by spatial and range features \((i, j, L, u, v) \in \mathbb{R}^5\):

- \((i, j)\) is the pixel location;
- \((L, u, v)\) the pixel value.

\(N = 262,144\) points, \(D = 5\) dimensions

---

**Number of iterations**

- Oracle
- MST
- \(D_K\)
- Raster
- Bounds
- Random

**Runtime**

- Oracle
- MST
- \(D_K\)
- Raster
- Bounds
- Random

**Number of points converged after \(i\) iterations**

- Euler
- Newton
- Halley
- Ridder
- Brent
- Bisection
Experimental evaluation: digits

60,000 handwritten digits from the MNIST dataset. Each datapoint is a $28 \times 28$ grayscale image.

$N = 60,000$ points, $D = 784$ dimensions
Articles from Grolier's encyclopedia. Each point is a word count of the most popular 15,275 words from 30,991 articles.

\[ N = 30,991 \text{ points}, D = 15,275 \text{ dimensions} \]
Conclusions

• We studied the behavior of entropic affinities and their properties.

• Search for the affinities involves finding the root of non-linear equation.

• We can find the root almost to machine precision in just over one iteration per point on average using:
  
  ‣ root-finding methods with high-order convergence,
  ‣ warm-start initialization based on local or density orders,
  ‣ bounds for the root.

• In applications such as spectral clustering and embeddings, semi-supervised learning using entropic affinities should give better results than fixing the bandwidth to a single value or using a rule-of-thumb.

• The only user parameter is the global perplexity value $K$.

• MATLAB code online at http://eecs.ucmerced.edu. Run it simply like $[W,s] = ea(X,K)$. 
Part 2.

Constructing **scalable affinities**
Spectral methods

• Consider a spectral problem:
  \[
  \min_X \text{tr} \left( XAX^T \right) \quad \text{s.t.} \quad XBX^T = I
  \]
  
  \(A_{N \times N}\) : symmetric psd affinity matrix.
  \(B_{N \times N}\) : symmetric pd (usually diagonal), set the scale of \(X\).

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

• Solution is unique and can be found in closed form from the eigenvectors of \(N \times N\) matrix \(C = B^{-\frac{1}{2}} AB^{-\frac{1}{2}}\).

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

Goal is find a fast, approximate solution for the embedding $X$ using only the subset of the original points from $Y$.

Applications:
- When $N$ is so large that the direct solution is infeasible.
- To select hyperparameters (e.g., for Gaussian kernel: $k, \sigma$) efficiently even if $N$ is not large (since a grid search over these requires solving the eigenproblem many times).
- As an out-of-sample extension to spectral methods.
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 $Y$

Affinity matrix

Learned $X$

Landmarks $\tilde{Y}$

Reduced affinity matrix

Learned $\tilde{X}$
The **Nyström method** is the standard way to approximate large-scale eigenproblems. Essentially, an out-of-sample formula:

1. Solve the eigenproblem for a subset of points (landmarks)

   \[ \tilde{Y} = \tilde{y}_1, \ldots, \tilde{y}_L, \] where \( L \ll N \).

2. Predict \( \mathbf{x} \) for any other point \( \mathbf{y} \) through an interpolation formula:

   \[
x_k = \frac{\sqrt{L}}{\lambda_k} \sum_{l=1}^{L} K(\mathbf{y}, \tilde{y}_l) u_{lk} \quad k = 1, \ldots, d
   \]

**Problems:**

- Needs to know the interpolation kernel \( K(\mathbf{y}, \mathbf{y}') \) (sometimes tricky).
- 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)

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

- 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):
  \[ \mathbf{A} \Rightarrow a_{ij} \Rightarrow \text{path } i \rightarrow j \]

- LLL (reduced affinities):
  \[ \tilde{\mathbf{A}} = \mathbf{ZAZ}^T \Rightarrow \tilde{a}_{ij} = \sum_{n,m=1}^{N} z_i a_{nm} z_j m \Rightarrow \text{path } i \rightarrow n \rightarrow m \rightarrow j \quad \forall n, m \]

So landmarks \( i \) and \( j \) can be farther apart and still be connected along the manifold.
LLL: construction of the weight matrix $Z$

- Most embedding methods seek to preserve local neighborhoods between the high- and low-dimensional spaces.
- Hence, if we assume that a point may be approximately linearly reconstructed from its nearest landmarks in high-dim space:

$$y_n \approx \sum_{l=1}^{L} zln \tilde{y}_l, \quad n = 1, \ldots, N \quad \Rightarrow \quad Y \approx \tilde{Y}Z$$

the same will happen in low-dim space: $X \approx \tilde{X}Z$.

- We consider only the $K_Z$ nearest landmarks, $d + 1 \leq K_Z \leq L$
  1. Find the $K_Z$ nearest landmarks of each data point.
  2. Find their weights as $Z = \arg \min Z \|Y - \tilde{Y}Z\|^2$ s.t. $Z^T 1 = 1$.

These are the same weights used by Locally Linear Embedding (LLE) (Roweis & Saul 2000).

- This implies the out-of-sample mapping (projection for a test point) is globally nonlinear but locally linear: $x = M(y)y$, where matrix $M(y)$ of $d \times D$ depends only on the set of nearest landmarks of $y$. 
• We assume the affinity matrix is given. If not, use approximate nearest neighbors to compute it.

• Time: the exact runtimes depend on the sparsity structure of the affinity matrix $A$ and the weight matrix $Z$, but in general the time is dominated by:
  › LLL: finding the nearest landmarks for each data point.
  › Nyström: computing the out-of-sample mapping for each data point.
and this is $O(Nld)$ in both cases.
Note LLL uses fewer landmarks to achieve the same error.

• Memory: LLL and Nyström are both $O(Ld)$.  

LLL: user parameters

- **Location of landmarks**: a random subset of the data works well. Refinements such as $k$-means improve a little with small $L$ but add runtime.

- **Total number of landmarks $L$**: as large as possible. The more landmarks, the better the result.

- **Number of neighboring landmarks $K_Z$ for the projection matrix $Z$**: $K_Z \geq d + 1$, where $d$ is the dimension of the latent space.
  - Each point should be a locally linear reconstruction of its $K_Z$ nearest landmarks:
    - $K_Z$ landmarks span a space of dimension $K_Z - 1 \Rightarrow K_Z \geq d + 1$.
    - Having more landmarks protects against occasional collinearities, but decreases the locality.
LLL:algorithm

Given spectral problem $\min_X \text{tr} \left( XAX^T \right)$ s.t. $XBX^T = I$ for dataset $Y$:

1. Choose the number of landmarks $L$, as high as your computer can support, and $K_Z \geq d + 1$.

2. Pick $L$ landmarks $\tilde{y}_1, \ldots, \tilde{y}_L$ at random from the dataset.

3. Compute local reconstruction weights $Z_{L \times N}$ for each data point wrt its nearest $K_Z$ landmarks:

   $$Z = \arg \min_Z \| Y - \tilde{Y}Z \|^2 \quad \text{s.t.} \quad Z^T 1 = 1$$

4. Solve reduced eigenproblem

   $$\min_{\tilde{X}} \text{tr} \left( \tilde{X} \tilde{A} \tilde{X}^T \right) \quad \text{s.t.} \quad \tilde{X} \tilde{B} \tilde{X}^T = I \quad \text{with} \quad \tilde{A} = ZAZ^T, \quad \tilde{B} = ZBZ^T$$

   for the landmark projections $\tilde{X}$.

5. Predict non-landmarks with out-of-sample mapping $X = \tilde{X}Z$. 
Experiments: Laplacian eigenmaps

We apply LLL to Laplacian eigenmaps (LE) (Belkin & Niyogi, 2003):

• **A:** graph Laplacian matrix $L = D - W$ for a Gaussian affinity matrix $W = (\exp(-\|y_n - y_m\|^2))_{nm}$ on $k$-nearest-neighbor graph.

• **B:** degree matrix $D = \text{diag} \left( \sum_{m=1}^{N} w_{nm} \right)$.

$$\min_X \text{tr} \left( XLX^T \right) \text{ s.t. } XDX^T = I, \ XD1 = 0.$$

LLL's reduced eigenproblem has $\tilde{A} = ZLZ^T$, $\tilde{B} = ZDZ^T$.

We compare LLL with 3 baselines:

1. **Exact LE** runs LE on the full dataset.
   - Ground-truth embedding, but the runtime is large.

Landmark LE runs LE only on a set of landmark points. Once their projection is found, the rest of the points are embedded using:

2. **LE(Nys.):** out-of-sample mapping using Nyström's method.

3. **LE(Z):** out-of-sample mapping using reconstruction weights.
Experiments: effect of the number of landmarks

- $N = 60,000$ MNIST digits, project to $d = 50$, $K_Z = 50$ landmarks.
- Choose landmarks randomly, from $L = 50$ to $L = N$.

LLL produces an embedding with quite lower error than Nyström's method for the same number of landmarks $L$. 

![Graph showing runtime and error with respect to Exact LE for different numbers of landmarks.](image)
Experiments: effect of the number of landmarks

- **Exact LE**, 80 s.
- **LE (Z)**, 5 s.
- **LE (Nys.)**, 5 s.

Legend:
- 0: Red
- 1: Blue
- 2: Green
- 3: Cyan
- 4: Orange
- 5: Purple
- 6: Black
- 7: Gray
- 8: Yellow
- 9: Pink
Experiments: model selection in Swiss roll dataset

Vary the hyperparameters of Laplacian eigenmaps (affinity bandwidth $\sigma$, $k$-nearest-neighbor graph) and compute for each combination the relative error of the embedding $\mathbf{X}$ wrt the ground truth on $N = 4000$ points using $L = 300$ landmarks. Matrix $\mathbf{Z}$ need only be computed once. The minima of the model selection error curves for LLL and Exact LE align well.

The graphs show the runtime and error for different values of bandwidth $\sigma$ and number of neighbors $k$. The runtime increases with increasing $\sigma$ and $k$, while the error decreases with increasing $\sigma$ and increases with increasing $k$. The minima of the error curves align well for both LLL and Exact LE.
Experiments: model selection in classification task

Find hyperparameters to achieve low 1-nn classification error in MNIST.
- 50,000 points as training, 10,000 as test, 10,000 as out-of-sample.
- Project to $d = 500$ using LLL ($K_Z = 50$, $L = 1,000$).

In runtime, LLL is $15-40\times$ faster than Exact LE. The model selection curves align well, except eigs in Exact LE fails to converge for small $k$.

![Exact LE test error (%)](image1)

![LLL test error (%)](image2)
Experiments: large-scale dataset

- \( N = 1\,020\,000 \) points from infiniteMNIST.
- \( L = 10^4 \) random landmarks (1\%), \( K_Z = 5 \).

LLL (18 min runtime)  
LE (Z)
Experiments: large-scale dataset

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

• The basic reason why LLL improves over Nyström's method is that, by using the entire dataset, it constructs affinities that better represent the manifold for the same number of landmarks.
• Hence, it requires fewer landmarks, and is faster at training and test time.
• It applies to any spectral method. No need to work out a special kernel as in Nyström's method.
• LLL can be used:
  ‣ to find a fast, approximate embedding of large dataset,
  ‣ to do fast model selection,
  ‣ as an out-of-sample extension to spectral methods.
Summary

**Affinity matrix** serves an important role in the behavior of many machine learning methods.

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

- 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.

Partially supported by NSF CAREER award IIS–0754089.
Summary

Affinity matrix serves an important role in the behavior of many machine learning methods.

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

- 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.

Thank you! Questions?

Partially supported by NSF CAREER award IIS–0754089.