UNIVERSITY OF CALIFORNIA, MERCED

Large-Scale Methods for Nonlinear Manifold Learning

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering & Computer Science

by

Maksym Vladymyrov

Committee in charge:

Professor Miguel Á. Carreira-Perpiñán, Chair
Professor Ming-Hsuan Yang
Professor Florin Rusu
Professor Jaakko Peltonen

2014

The dissertation of Maksym Vladymyrov is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

Chair

University of California, Merced

2014

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Many people directly or indirectly contributed to this dissertation, and I am extremely grateful for their support. My parents and grandparents always fostered an intellectual environment in the house, where books and knowledge were always the ultimate source of wisdom. In high school, I was lucky enough to be educated by people of great intellect and sagacity that taught me persistence and humility. My undergraduate professors taught me the ability to be adaptive and quickly acquire new knowledge. I thank all those people who have contributed to my education. I would like to specifically mention my friends and mentors from Ukraine: Alexandra Jigulina, Nikolai Bahmetiev, Anastasia Nosich, Vladimir Elovsky, Kseniia Verbinina, Denis Oleynik, Irina Zaretskaya, Igor Illin, Nikolai Isaev and Maina Levina. Each of those people made a profound impact on my life and without them I would not be where I am today.

Next, I want to thank my advisor, Miguel Á. Carreira-Perpiñán, whose passion for science and knowledge in something that has always inspired me. I'm also thankful to my committee members Ming-Hsuan Yang, Florin Rusu and especially to Jaakko Peltonen for finding time and making important comments to the draft of this dissertation. I would like to thank Jeff Shrager, David Noelle, Shawn Newsam, Sergey Kirshner and Chris Kello for their encouragement and support.

Special thanks to my friends Marco Valesi, Alicia Ramos Jordan, Carlo Camporesi, Aki Ohdera, Ankur Kamthe, Varick Erickson, Erin Gaab, Megan Schill, Lisa Neubauer, Joe Torres, Paola Di Giuseppantonio, Fabrizio Galeazzi, Martyna Citkowicz, Mapi Asta, Karen Sachs, Clement Otu and many others. You all know that I love you and this dissertation would not have been possible without you!

I would also like to acknowledge the following funding sources that have financially supported my research for this dissertation: NSF CAREER award IIS-0754089.

Last but not least, I would like to thank my beautiful family Nina, Marina, Oleksii and Lubov' Vladymyrov. This dissertation is dedicated to them.

San Francisco, 2014

VITA

2007	Bachelor of Science in Applied Mathematics, Kharkiv National University, Ukraine
2008	Bachelor of Science in International Relations, Kharkiv National University, Ukraine
2008	Master of Science in Computer Science, Kharkiv National University, Ukraine
2009	Master of Science in International Economic Relations, Kharkiv National University, Ukraine
2014	Ph. D. in Electrical Engineering and Computer Science, University of California, Merced

PUBLICATIONS

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.

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.

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.

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.

ABSTRACT OF THE DISSERTATION

Large-Scale Methods for Nonlinear Manifold Learning

by

Maksym Vladymyrov

Doctor of Philosophy in Electrical Engineering & Computer Science

University of California, Merced, 2014

Professor Miguel Á. Carreira-Perpiñán, Chair

High-dimensional data representation is an important problem in many different areas of science. Nowadays, it is becoming crucial to interpret the data of varying dimensionality correctly. Dimensionality reduction methods process the data in order to help visualize the data, reduce its complexity, or find latent representation of the original problem. The algorithms of nonlinear dimensionality reduction (also known as manifold learning) are used to decrease the dimensionality of the problem while preserving the general structure of the data. Both spectral methods (such as Laplacian Eigenmaps or ISOMAP) and nonlinear embedding algorithms (NLE, such as t-SNE or Elastic Embedding) have shown to provide very good nonlinear embedding of high-dimensional data sets. However, those methods are notorious

for very slow optimization, practically preventing them from being used when a data set is bigger than few thousand points.

In my thesis we investigate several techniques to improve different stages of nonlinear dimensionally algorithms. First, we analyze the entropic affinities as a better way to build a similarity matrix. We explore its properties and propose a nearly-optimal algorithm to construct them. Second, we present a novel faster method to optimize NLE by using second-order information during the optimization. Third, for spectral methods, we investigate landmark-based optimization that cleverly substitutes original large-scale problem with a much smaller easy-to-solve subproblem. Finally, we apply Fast Multipole Methods approximation that allows fast computation of the gradient and the objective function of NLE and reduces their computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.

Each of the proposed methods accelerate the optimization dramatically by one or two orders of magnitude compared to the existing techniques, effectively allowing corresponding methods to run on a dataset with millions of points.

Chapter 1

Introduction

Dimensionality reduction is an important problem in machine learning. It is often used to explore the structure of high-dimensional datasets, to identify useful information such as clustering, or to extract low-dimensional features that are useful for classification, search or other tasks. More generally, given a high-dimensional data $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ consisting of N points in D-dimensional space, dimensionality reduction algorithms try to obtain a projection of that data $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ onto some low-dimensional space d (with $d < D$, ofter $d \ll D$) that somehow preserves the structure of that data. In addition, different methods can also return the following:

- a *reduction mapping* $F : \mathbb{R}^D \to \mathbb{R}^d$ that maps any point (not just the ones from the training set $\mathbf{Y}$) to a low-dimensional space,
- a *reconstruction mapping* $f : \mathbb{R}^d \to \mathbb{R}^D$ that returns a point in the original space given the point in the projection space,
- *joint probability density* $p(\mathbf{x}, \mathbf{y})$,
- an estimate of the *intrinsic dimensionality* d.

In case when the mappings F and f are defined explicitly in a parametric form (e.g. by using real-basis functions or a neural network), the optimization is done over the parameters of those mappings and the method is called *parametric*. Otherwise, the optimization is performed over the low-dimensional projection points $\mathbf{X}$ and

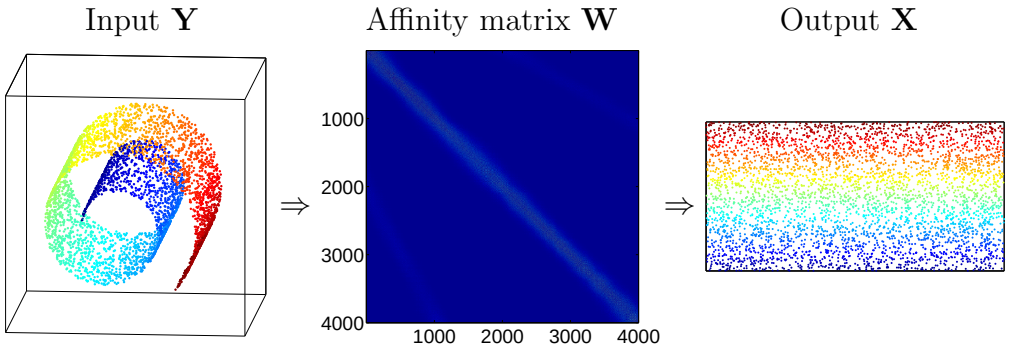

Figure 1.1: Typical progression of embedding algorithms based on pairwise affinities. First the dataset $\mathbf{Y}$ is encoded as $N \times N$ affinity matrix $\mathbf{W}$. Then, the output $\mathbf{X}$ is found such that affinities $\mathbf{W}$ are best preserved.

the method is called *non-parametric*. In the latter case, the method returns only the projection of the training set $\mathbf{X}$ and it becomes an important problem to find a projection of a new test point $\mathbf{y}$. This problem is called *out-of-sample projection* problem.

In this thesis we are going to focus on a class of *nonparametric dimensionality reduction methods*, where the goal is to find the projection $\mathbf{X}$ of the training set $\mathbf{Y}$. However, for some of the problems, we are going to mention out-of-sample projection as well.

More specifically, we focus on the class of *embedding algorithms based on pairwise affinities*. Here, we first encode $\mathbf{Y}$ as a weighted *affinity graph* $\mathbf{W}$, where each data point is a vertex and weighted edges indicate similarity or distance between objects. Then, the goal is to find a low-dimensional projection $\mathbf{X}$ whose Euclidean distances optimally preserve these similarities (see fig. 1.1).

There are generally three big group of methods that deal with this problem: linear methods, spectral methods and nonlinear embedding methods. Below we are going to describe each of those groups in detail.

Linear methods such as Principal Component Analysis (PCA), factor analysis, classical Multidimensional Scaling (cMDS, Cox and Cox, 1994; Borg and Groenen, 2005), Locality Preserving Projections (LPP, He and Niyogi, 2004) etc. These methods restrict the mapping to be of a linear form (i.e. of the form $\mathbf{Y} = \mathbf{AX}$,

where $\mathbf{A} \in \mathbb{R}^{D \times d}$ is the mapping). However, the manifold underlying the data $\mathbf{Y}$ is rarely linear and these kind of methods often do not give a good representation of the data. In addition, the assumptions of the methods are different and it is not clear which of the method give better result. For example, PCA selects the components along the direction of largest variance, which may not give the best performance.

Spectral methods such as Laplacian Eigenmaps (LE, Belkin and Niyogi, 2003), Locally Linear Embedding (LLE, Roweis and Saul, 2000), ISOMAP Tenenbaum et al., 2000 and Maximum Variance Unfolding (MVU, Weinberger and Saul, 2006) are more general and able to capture nonlinear structure of the original data. Spectral methods have become very popular because they have a unique solution that can be efficiently computed by an eigensolver, and yet they are able to unfold nonlinear, convoluted manifolds. That said, their embeddings are far from perfect, particularly when the data has nonuniform density or multiple manifolds.

Nonlinear embeddings (NLE) such as stochastic neighbor embedding (SNE; Hinton and Roweis, 2003), symmetric SNE (s-SNE, Cook et al., 2007), t-SNE (van der Maaten and Hinton, 2008), neighbor retrieval visualizer (NeRV, Venna et al., 2010) and elastic embedding (EE; Carreira-Perpiñán, 2010), do not have any restriction on a form of the objective function and therefore are the most general. They produce embeddings that are much better than those of linear or spectral methods, especially when the high-dimensional data have a complex cluster and manifold structure. However, the objective function of NLE is non-convex, which means that we have to retrieve to iterative optimization methods that could potentially be trapped in local minima.
In fig. 1.2 we compare the typical results of running different methods for a case of finding a 2D embedding of a simple dataset of three objects from COIL-20 (see Appendix A for a description of the dataset). Although we do not know how the ideal embedding should look like (it is unsupervised problem with not labeled data), we can say that to reveal the structure of the data the successful method should at least be able to (1) separate all three objects one from another and (2)

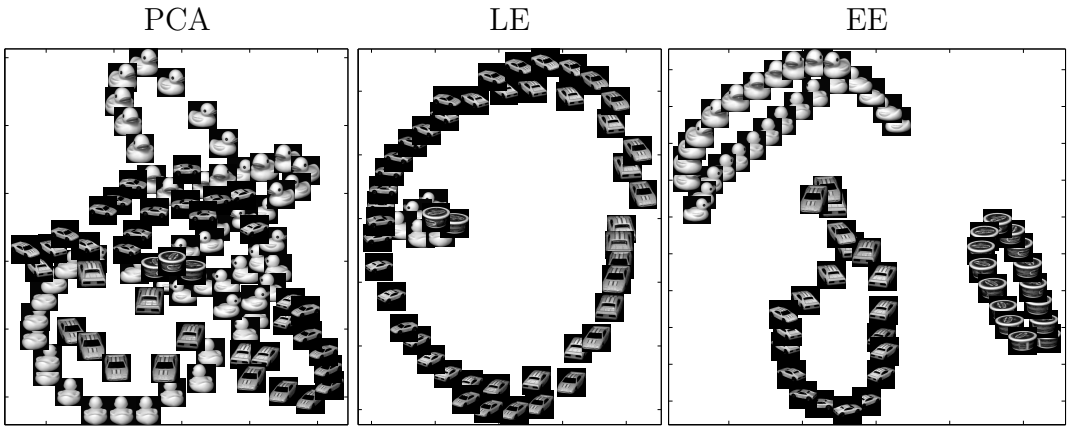

Figure 1.2: Example of 2D embedding of three objects from COIL-20 dataset obtained with linear method (PCA, *left plot*), spectral method (LE, *central plot*) and nonlinear method (EE, *right plot*). The quality of the embedding improves as we go from left to right.

separate the images of the same objects, such that the points that corresponds to the similar rotation sequence appear closest to each other and (3) separate the images of the same objects, such that the points that corresponds to the different rotation sequence appear far from each other. Ideally, we would like to see three separated concentric loops that capture the rotational sequence of the photos. On the left plot we have the result of PCA, which is a linear method. Although, it does give somewhat meaningful results (for example, the criteria (2) and (3) above are generally satisfied), it appears that the linear assumption is too restrictive to separate the manifolds one from another. In the central plot, we show the result obtained with a spectral method, LE. It shows much clearly the structure of the data comparing to the linear case, however, it still has problems. In particular, the embeddings of the ducks and the cream cheese package are collapsed on top of each other (criterion (3) is violated). Finally, the right part of the figure shows the embedding obtained using an NLE method, EE. It shows much better separation between the objects comparing to the previous two cases and also more accurately display the rotational structure of the images of each object. It is clear from the results, that the embedding quality improves as we go from linear methods to spectral to nonlinear embedding.

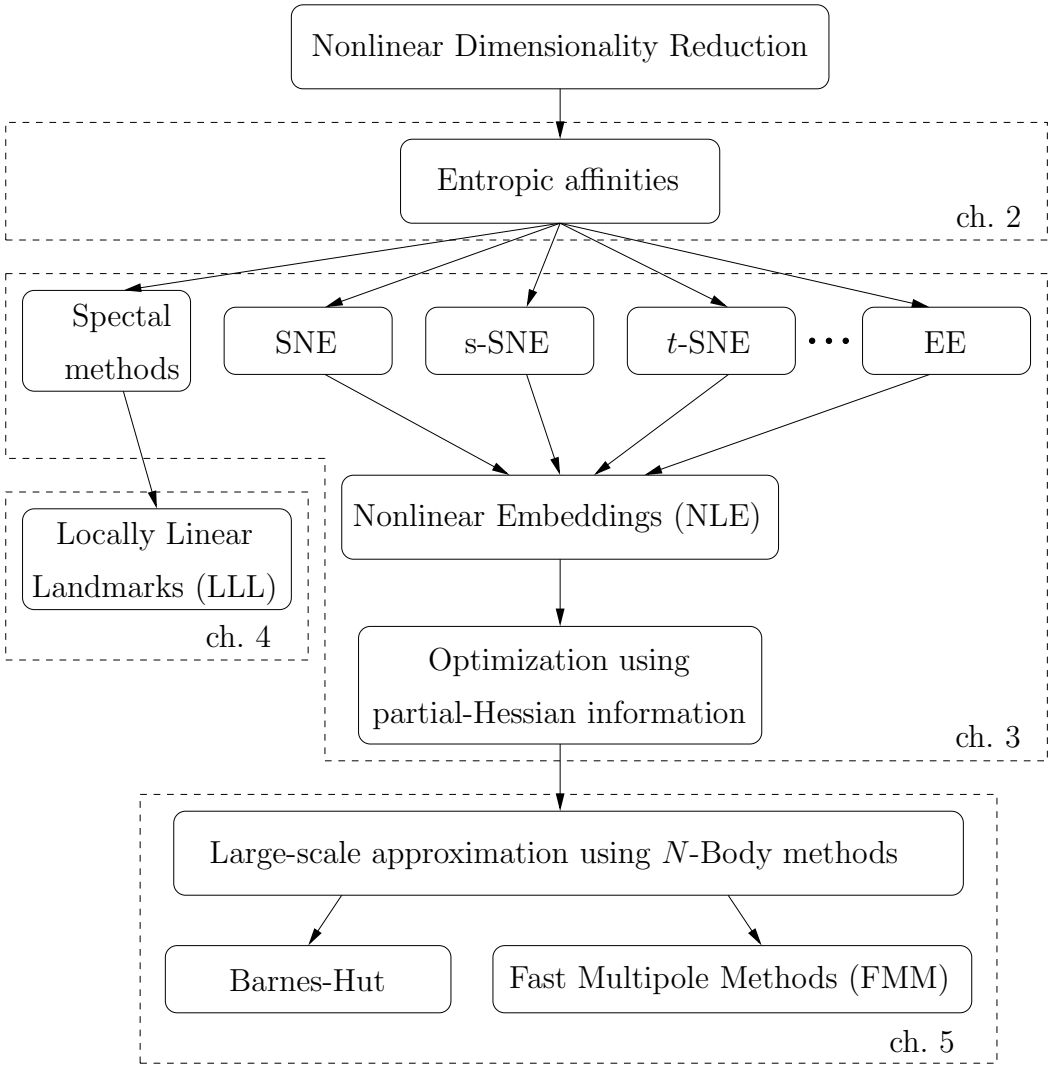

Figure 1.3: Outline of the dissertation

In this thesis we have the following contributions (see fig. 1.3 for the outline). In Chapter 2 we concentrate on the problem of the affinity matrix construction. For most of the dimensionality reduction algorithms we deal with in this thesis, the affinity matrix is the only piece of information available to the algorithm and therefore it is of critical importance to construct this matrix such that it encodes the data as accurately as possible. We would concentrate on the entropic affinities, that were first introduces by Hinton and Roweis (2003), as a better way to construct a Gaussian affinity matrix that takes into account the whole distribution of

points. This gives very good affinities that adapt locally to the data but are harder to compute. We study the mathematical properties of these affinities and show that they implicitly define a continuously differentiable function in the input space and give bounds for it. We then devise a fast algorithm to compute the widths and affinities, based on robustified, quickly convergent root-finding methods combined with a tree- or density-based initialization scheme that exploits the slowly-varying behavior of this function. This algorithm is nearly optimal and much more accurate and fast than the existing bisection-based approach, particularly with large datasets, as we show with image and text data.

Chapter 3 puts more emphasis on the nonlinear embedding methods and their properties. We start by showing the connection between different nonlinear embedding methods in a common generic simple framework. Then, we show the relations of those methods with spectral methods and graph Laplacians. This allows us to define several partial-Hessian optimization strategies, characterize their global and local convergence, and evaluate them empirically. We achieve up to two orders of magnitude speedup over existing training methods with a strategy (which we call the spectral direction) that adds nearly no overhead to the gradient and yet is simple, scalable and applicable to several existing and future embedding algorithms.

In Chapter 4 we concentrate on the fast approximation of the spectral methods. With large datasets, the eigendecomposition is too expensive, and is usually approximated by solving for a smaller graph defined on a subset of the points (landmarks) and then applying the Nyström formula to estimate the eigenvectors for the rest of the points. This has the problem that the affinities between landmarks do not benefit from the remaining points and may poorly represent the data if using few landmarks. We introduce a modified spectral problem that uses the information from all the data by constraining the latent projection of each point to be a local linear function of the landmarks' latent projections. This constructs a new affinity matrix between landmarks that preserves manifold structure even with few landmarks, which allows one to reduce the eigenproblem size, and defines a fast, nonlinear out-of-sample mapping. We show the application of this technique to

speeding up dimensionality reduction (laplacian eigenmaps) and clustering (spectral clustering) methods.

In Chapter 5 we come back to the nonlinear embedding methods and address the main computational bottleneck of those algorithms which is quadratic cost of the objective function and the gradient. We propose to deal with this problem by formulating the optimization as an N-body problem and using fast multipole methods (FMMs) to approximate the gradient in linear time. We study the effect, in theory and experiment, of approximating gradients in the optimization and show that the expected error is related to the mean curvature of the objective function, and that gradually increasing the accuracy level in the FMM over iterations leads to a faster training. When combined with standard optimizers, such as gradient descent or L-BFGS, the resulting algorithm beats the $\mathcal{O}(N \log N)$ Barnes-Hut method and achieves reasonable embeddings for one million points in around three hours' runtime.

In Chapter 6 we draw conclusions and summarize potential future directions.

Appendix A describes the datasets used for the experimental evaluation.

For all the algorithms proposed in this thesis, we made the code publicly available for everyone to try at `http://eecs.ucmerced.edu/`.

Throughout we write pd (psd) to mean positive (semi)definite, and nd (nsd) to mean negative (semi)definite.

Chapter 2

Entropic Affinities

2.1 Introduction

Many machine learning algorithms rely on the choice of meta-parameters that govern their performance. These parameters depend on the data and good values are often hard to find. One such meta-parameter is the bandwidth σ that is used in the construction of affinities in many machine learning problems. These include dimensionality reduction methods such as LLE (Roweis and Saul, 2000), Laplacian eigenmaps (Belkin and Niyogi, 2003), ISOMAP (de Silva and Tenenbaum, 2003), SNE (Hinton and Roweis, 2003), and the elastic embedding (Carreira-Perpiñán, 2010); clustering methods such as spectral clustering (Ng et al., 2002) and mean-shift algorithms (Carreira-Perpiñán, 2006); semi-supervised learning (Zhou et al., 2003; Belkin et al., 2006); and many others. In some of those algorithms σ is the only parameter to tune and a user has to try several values until the desired quality of the algorithm is achieved. When the dataset is large, such a process is not interactive and can lead to frustration and ultimately to the user refusing to use a potentially good algorithm. On top of that, the best results of the algorithm may not be achieved for a single value of σ for all the points, but rather for a separate bandwidth for every datapoint, in which case the existence of automatic procedure is vital.

In their spectral clustering algorithm, Ng et al. (2002) suggest to set σ to the value giving

This chapter is an extended version of Vladymyrov and Carreira-Perpiñán (2013a).

least distorted clusters, but this requires running the algorithm, which is expensive. In a supervised setting, Er et al. (2002) select the bandwidth per cluster of data as the one that captures the variation between points in each cluster, but minimizes the overlapping of nearest neighbors in different classes. The method requires tuning some parameters that depend on the mean and variance of the clusters. Bchir and Frigui (2010) estimate one σ per cluster in an unsupervised manner using a fuzzy logic framework. The objective function of this method maximizes the scaling parameter per cluster up until the clusters start to overlap. There also exist classic rules of thumb, such as setting σ separately for each point to the distance d_k to the kth nearest neighbor of that point, where k is a user parameter (set to 7 in Zelnik-Manor and Perona, 2004). This has the odd behavior that σ would change proportionally to changes in d_k, but would ignore any changes to the rest of the distances, no matter how large, as long as d_k remained the kth distance; or else it would change discontinuously.

Additionally, metric learning algorithms (Kaski and Peltonen, 2003; Globerson and Roweis, 2006; Weinberger and Saul, 2009) preprocess the data in the way that it emphasizes the structure of the classes. This approach is quite different from ours, since it effectively changes the whole distribution of neighbors for each point and, also, it includes additional supervised label information that we don't consider in our problem. Comparing to this, entropic affinities build an single isotropic metric that is different for each point, without labels.

In this chapter, we study a previously proposed way to set per-point bandwidths that takes into account the whole distribution of distances and is a continuous, differentiable function of them. For a given point $\mathbf{y} \in \mathbb{R}^D$, consider the posterior distribution of an isotropic kernel density estimator of width σ defined on a finite set of points $\mathbf{y}_1, \dots, \mathbf{y}_N \in \mathbb{R}^D$. Thus we have a discrete distribution $p(\mathbf{y}; \sigma)$ with probabilities for $n = 1, \dots, N$

$$p_n(\mathbf{y}; \sigma) = \frac{K\big(\left\|\frac{\mathbf{y}-\mathbf{y}_n}{\sigma}\right\|^2\big)}{\sum_{k=1}^N K\big(\left\|\frac{\mathbf{y}-\mathbf{y}_k}{\sigma}\right\|^2\big)} = \frac{K\big((\frac{d_n}{\sigma})^2\big)}{\sum_{k=1}^N K\big((\frac{d_k}{\sigma})^2\big)} \tag{2.1}$$

where $d_n = \|\mathbf{y} - \mathbf{y}_n\|$. We focus on the case where $K(\|(\mathbf{y} - \mathbf{y}_n)/\sigma\|^2)$ is the Gaussian kernel. We set σ individually for point $\mathbf{y}$ to a value such that the entropy of the distribution $p(\mathbf{y}; \sigma)$, considered as a function of σ for fixed $d_1, \dots, d_N$, equals $\log K$, where K is a user-set perplexity parameter. The perplexity, widely used in natural language processing (Manning and Schütze, 1999), has an intuitive interpretation. A perplexity of K in a distribution p over N neighbors means p provides the same surprise as if we were

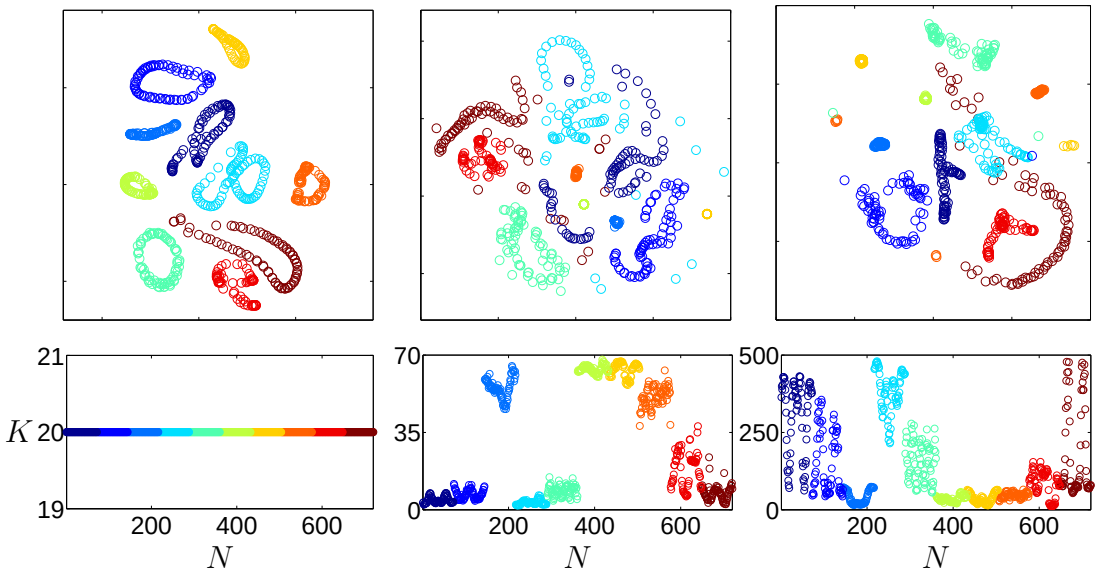

Figure 2.1: *Top plots*: embeddings of COIL dataset with the elastic embedding algorithm, using (from left to right): entropic affinities with perplexity $K = 20$; unique $\sigma = 9$ obtained by averaging σs with perplexity $K = 20$; $\sigma_n =$ distance to 7th nearest neighbor. *Bottom plots*: value of the perplexity K for each point $\mathbf{y}_n$ of the dataset. The color corresponds to different COIL manifolds.

to choose among K equiprobable neighbors. Having set σ in this way, the resulting value $p_n(\mathbf{y};\sigma)$ can be used as an affinity between $\mathbf{y}$ and $\mathbf{y}_n$. We call them *entropic affinities*. These affinities were introduced by Hinton and Roweis (2003) as a better way to define the local scaling of the Gaussian distribution in their stochastic neighbor embedding (SNE) method. Their definition of $p(\mathbf{y};\sigma)$ was particularized to $\mathbf{y}$ being one of the data points, but our generalization simplifies things later. The affinity p_{nm} between points $\mathbf{y}_n$ and $\mathbf{y}_m$ is then $p_n(\mathbf{y}_m;\sigma)$. If we consider an affinity matrix $\mathbf{W}$ with entries $K(\mathbf{y}_n, \mathbf{y}_m)$ and degree matrix $\mathbf{D} = \operatorname{diag}\left(\sum_{n=1}^{N} w_{nm}\right)$, then p defines the random-walk matrix $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}$, where each row is our distribution $p(\mathbf{y}_n;\sigma)$. Thus, the entropic affinities seek a matrix $\mathbf{P}(\sigma_1, \dots, \sigma_N)$ as a function of the kernel widths for each data point so that each row of $\mathbf{P}$ has perplexity K. To compute each σ_n, Hinton and Roweis (2003) performed a search to find a bracket for the solution, initialized at $[0, 1]$, and then used bisections. This becomes noticeably slow with large datasets.

Fig. 2.1 illustrates how the entropic affinities indeed improve over using a single σ or simple rule-of-thumb adaptive σ_n (the distance to the 7th nearest neighbor; Zelnik-Manor and Perona, 2004). We applied the elastic embedding dimensionality reduction

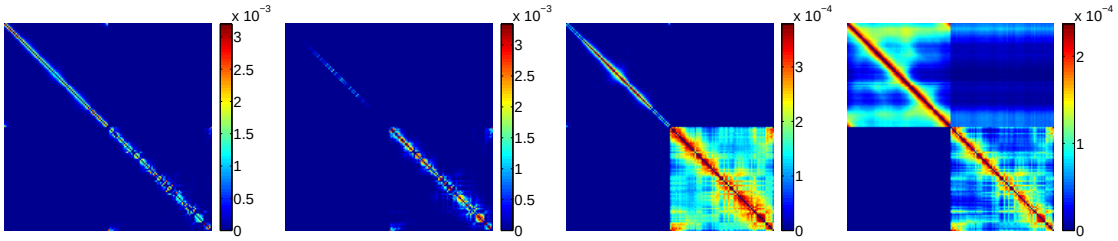

Figure 2.2: Affinity matrices for 2 objects from COIL-20 dataset build using different choice for bandwidth. *From left to right:* (1) bandwidth chosen with EA using perplexity $K = 10$, (2) fixed $\sigma = 4$, (3) fixed $\sigma = 10$, (4) σ_n for $\mathbf{y}_n$ equals to the distance to 7th nearest neighbor of $\mathbf{y}_n$.

(Carreira-Perpiñán, 2010) to the COIL-20 dataset (see Appendix A for the description). The left plot clearly shows the separation between the manifolds and the sequential structure of each manifold. The embedding resulting from a single σ or σ_n from the 7th neighbor does not show such a structure. The bottom plots show the σ values in the latter two cases result in a wide range of perplexity values.

Fig. 2.2 explains the result above from the affinity viewpoint. Points are ordered according to the rotational sequence of two objects and overall there are 144×144 points in the matrix. The plot of the left shows the entropic affinity matrix for two objects from COIL-20 dataset. Bright values along the diagonal correspond to large similarity between objects that have similar rotational position. Bright colors in the middle of each of the sides means that the rotational sequence "loops over" and last images of the sequence are similar to the first images. Finally large blue regions in the rest of the affinity matrix means that there is almost no connection between different objects and images of the same object that correspond to different rotations. Other plots show affinity matrices given by fixed σ values or by distance to the 7th neighbor. None of the choices give good affinity matrix. For $\sigma = 4$, the affinity does not connect the points of the first object. For $\sigma = 10$, it is now the second object that has large interconnection between all the images and thus does not show the rotational structure of that object. Using σ from 7th neighbor also fails to show that rotational structure of both objects.

Overall, mostly due to high-quality affinity matrix, like the one from the entropic affinities, we were able to achieve a good results of the elastic embedding that we observe on the left of Fig. 2.1. While, the entropic affinities still require a user variable K to be set, this is much more intuitive, since it defines the information theoretic quantity (effective number of neighbors), rather than spatial (measured in the distance between points). In

addition, setting a single K parameter gives a vector of bandwidth parameters one per datapoint, which would be impossible to set manually for even a medium-size dataset. In this chapter we will investigate the entropic affinities and their numerical computation. Section 2.2 proves useful properties, in particular that the function $\sigma(\mathbf{y})$ is well defined and continuously differentiable, and give simple bounds for it. Based on this, section 2.3 describes fast, scalable algorithms that compute σ and the entropic affinities themselves in very few iterations to almost machine precision, by processing points in a certain order. Section 2.4 shows experimental results with image and text datasets.

2.2 Some properties of entropic affinities

The entropy of the distribution (2.1) is defined as

$$\begin{aligned} H(\mathbf{y},\sigma) &= -\sum_{n=1}^{N} p_n(\mathbf{y},\sigma)\log(p_n(\mathbf{y},\sigma)) \\ &= -\sum_{n=1}^{N} p_n(\mathbf{y},\sigma)\log K(\|(\mathbf{y}-\mathbf{y}_n)/\sigma\|^2) + \log\sum_n K(\|(\mathbf{y}-\mathbf{y}_n)/\sigma\|^2). \end{aligned} \tag{2.2}$$

In particular, for the Gaussian kernel it becomes

$$H(\mathbf{y},\beta) = \beta\sum_{n=1}^{N} p_n(\mathbf{y},\beta)d_n^2 + \log\sum_{n=1}^{N}\exp(-d_n^2\beta), \tag{2.3}$$

where the probabilities are defined for the Gaussian kernel as

$$p_n(\mathbf{y},\beta) = \frac{\exp(-d_n^2\beta)}{\sum_{m=1}^{N}\exp(-d_m^2\beta)}, \tag{2.4}$$

with the precision parameter $\beta = 1/2\sigma^2$. We can express (2.3) and its derivatives wrt β using the partition function $Z(\beta) = \sum_{n=1}^{N}\exp(-d_n^2\beta)$ and moments $m_k(\beta) = \sum_{n=1}^{N} p_n d_n^{2k}$ as follows:

$$\begin{aligned} H(\mathbf{y},\beta) &= \beta m_1 + \log Z, \\ H'_\beta(\mathbf{y},\beta) &= -\beta(m_2 - m_1^2), \\ H''_\beta(\mathbf{y},\beta) &= \beta(m_3 - 3m_2m_1 + 2m_1^3) + m_1^2 - m_2. \end{aligned} \tag{2.5}$$

In turn, m_k can be expressed as a function of Z and its derivatives using the following recursive definition:

$$m_1 = -\frac{1}{Z}\frac{\partial Z}{\partial\beta}; \quad m_{k+1} = m_1 m_k - \frac{\partial m_k}{\partial\beta}, \text{ for } k > 1. \tag{2.6}$$

This reformulation also simplifies the evaluation of the entropy and its derivatives. To get the entropy value we need to compute Z and m_1. For each of the derivatives we need to compute one additional moment and reuse elements that we computed on the previous stages. Each of those operations takes $\mathcal{O}(N)$, however computing Z is roughly $4\times$ slower than computing the moments (due to exponentiation). Thus, the evaluation of the entropy takes approximately five times as long as evaluating its derivatives. In fig. 2.3 we show the typical example of the runtime calculation of the entropy and its first two derivatives for a 100 points with respect to a different number of neighbors.

We now consider the problem of searching for σ (or β), which is implicit given the perplexity K:

$$F(\mathbf{y}, \beta, K) := H(\mathbf{y}, \beta) - \log K = 0. \tag{2.7}$$

This is a 1D root-finding problem or an inversion problem if $H(\mathbf{y}, \beta)$ is invertible over β.

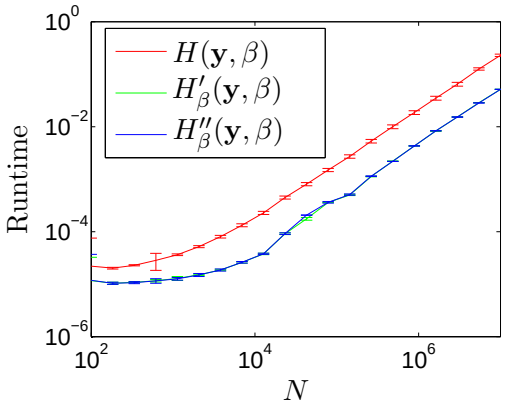

Figure 2.3: Runtime needed to compute the entropy and its derivatives for different neighborhood size.

Proposition 2.2.1. *The entropy function for Gaussian kernel is monotonically decreasing function of β that decreases from $\log N$ for $\beta = 0$ to 0 for $\beta \to \infty$.*

Proof. The first derivative of the entropy can be rewritten as

$$\frac{\partial H(\beta)}{\partial \beta} = \beta \sum_n^N \frac{\partial p_n}{\partial \beta} d_n = \beta \Big(\big(\sum_{n=1}^N p_n d_n \big)^2 - \sum_{n=1}^N p_n d_n^2 \Big) = -\beta \sum_n^N \sum_{m>n}^N p_n p_m (d_n - d_m)^2. \tag{2.8}$$

It is negative for $\beta > 0$ given that the points are not equidistant from $\mathbf{y}$. □

From Proposition 2.2.1, the problem (2.7) is well defined for any value of $\beta > 0$ and has a unique root $\beta(\mathbf{y})$ for any $K \in (0, N)$ in the same interval.

Next, notice that $H(\mathbf{y}, \beta)$ is continuously differentiable in an open neighborhood of $(\mathbf{y}_0, \beta_0)$ for some fixed $\beta_0 > 0$ and $\mathbf{y}_0 \in \mathbb{R}^D$ and that $H'_\beta(\mathbf{y}, \beta_0) \neq 0$. Thus, we can apply the implicit function theorem to show the existence of a uniquely defined local continuously differentiable function $\beta(\mathbf{y})$ that satisfies $\beta(\mathbf{y}_0) = \beta_0$. Moreover, since $H(\mathbf{y}, \beta)$ is invertible, the function $\beta(\mathbf{y})$ is also a global function defined for all $\mathbf{y}$. The

same argument can be applied to $F(\mathbf{y}, \beta, K)$, leading to the existence of a continuously differentiable global function $\beta(\mathbf{y}, K)$ defined for all $K \in (0, \log N)$ and $\mathbf{y} \in \mathbb{R}^D$.

Finally, we give bounds $[\beta_L, \beta_U]$ for $\beta(\mathbf{y}, K)$, i.e., satisfying $H(\mathbf{y}, \beta_L) > \log K > H(\mathbf{y}, \beta_U)$ for every $\mathbf{y}$ and K, that are easy to compute and reasonably tight. Assume w.l.o.g. the squared distances are sorted increasingly: $d_1^2 < d_2^2 < \cdots < d_N^2$. Define $\overline{d^2} = \frac{1}{N}\sum_{n=1}^N d_n^2$, $\overline{d^4} = \frac{1}{N}\sum_{n=1}^N d_n^4$, $\Delta_N^2 = d_N^2 - d_1^2$ and $\Delta_2^2 = d_2^2 - d_1^2$.

Theorem 2.2.2. *The lower and the upper bounds for β can be found using the formulae:*

$$\beta_L = \max\left(\frac{N}{N-1}\frac{\log\frac{N}{K}}{\Delta_N^2}, \sqrt{\frac{\log\frac{N}{K}}{d_N^4 - d_1^4}}\right), \tag{2.9}$$

$$\beta_U = \frac{1}{\Delta_2^2}\log\Big(\frac{p_1}{1-p_1}(N-1)\Big), \tag{2.10}$$

where p_1 is the only solution in the interval $[3/4, 1]$ of the equation:

$$2(1-p_1)\log\frac{N}{2(1-p_1)} = \log\big(\min(\sqrt{2N}, K)\big). \tag{2.11}$$

The proof of this theorem can be found in Vladymyrov and Carreira-Perpiñán (2013a). Practically, to obtain bounds, we can solve (2.11) using e.g. Newton's method and needs be done only once for all the points in the dataset, since p_1 depends only on K and N. Thus, the computation of the bounds is $\mathcal{O}(1)$ for each data point, since they only need d_1, d_2 and d_N. Tighter bounds can be obtained by using all distances $d_1, \ldots, d_N$, but at a cost $\mathcal{O}(N)$ per point, which defeats the purpose.

Sometimes the distances $d_1^2, d_2^2, \ldots, d_N^2$ may be equal to each other, which violates the assumption of increased distances $d_1^2 < d_2^2 < \cdots < d_N^2$. While, this should not violate the general results of the theorem above, special care should be taken to make sure that $\Delta_2^2 \neq 0$ (otherwise we cannot compute the upper bound (2.10)). We can avoid this problem by including self-affinities, in which case d_1^2 is always d_2^2. However, this will change the definition of the affinities that was proposed originally Hinton and Roweis (2003). Alternativelly, when $d_1^2 = d_2^2$ we can either (1) increase the distance to the second neighbor by a little bit or (2) take $\Delta_2^2 = d_k^2 - d_1^2$, where d_k is the first neighbor that is located father than the first neighbor.

Rescaling the data (or the distances) rescales σ as well, i.e. $H^{-1}(K; \alpha d) = \alpha H(K; d)$ for any $\alpha > 0$. This suggests that rescaling the data should rescale the bounds correspondingly, which indeed happens for our bounds.

Our results carry over, suitably modified, to some variations of our problem. The formulation (2.1) implies $p_{nn} \neq 0$. It is also possible to set self-affinities p_{nn} to 0 (as is

sometimes done) by defining p over the distances d_2 to d_N instead. One can also use sparse affinities if defining $p(\mathbf{y};\sigma)$ on the k nearest neighbors of $\mathbf{y}$ rather than all N points. This means setting $N = k$ with points sorted in increasing distance to $\mathbf{y}$. We will analyze this variation in the next section.

2.3 Computation of entropic affinities

To compute the entropic affinities we need to solve the root-finding problem (2.7) as efficiently as possible for every point in the dataset. As we show above, the solution is uniquely defined, but is difficult to find in a closed-form, thus we have to retrieve to iterative root-finding algorithms.

There exist many one-dimensional root-finding algorithms with different convergence orders both derivative-free and derivative-based. Some of the most popular derivative-free methods include the bisection method, Brent's method (1973) and Ridders' method (1979). These methods have universal convergence guarantees and take as an input an interval bracketing the root, which they iteratively shrink. The problem of those methods is that they convergence rate is usually slower that the one of the derivative-based methods.

Derivative-based methods such as Newton's, Halley's and Euler's methods (Traub, 1982) start with a single initialization point and construct a sequence of iterates that, hopefully, converge. The next iterate is found based on the value of the function and its derivatives at the current iterate. These methods usually do not have global convergence guarantees unless the function has some very specific form (Melman, 1997). However, their convergence order is usually higher than that of derivative-free methods.

In this section we discuss the most efficient ways to find the solution to the root-finding problem (2.7) to a high accuracy. First we show that it is more efficient to reformulate (2.7) to solve for $\log\beta$, rather than β. Then, we show that using sparse distance matrix leads to larger speed-up almost without compromising the accuracy. Next, we discuss different root-finding algorithms and show how we can efficiently employ the bounds derived in the previous section to propose a simple modification to any derivative-based algorithm to achieve universal convergence guarantees. Finally, we are going to analyze different warm-start initializations that allows us to initialize the problem very close to the root.

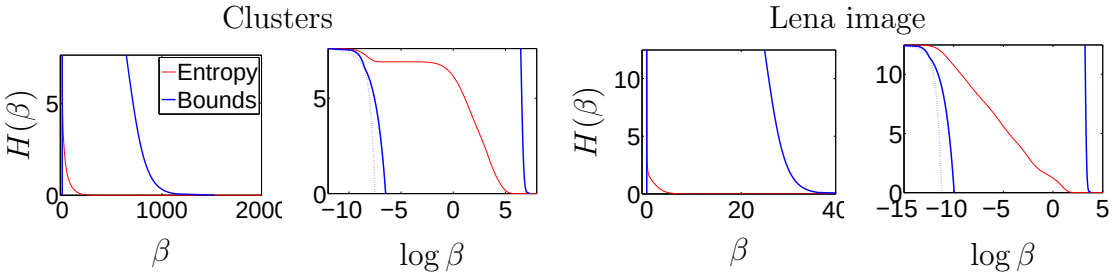

Figure 2.4: The entropy function (in red) with its bounds (in blue) for linear and log scale of β. The solid blue lines indicate the lower and the upper bound obtained using the formulae (2.9) and (2.10) for different values of K. The dashed blue line indicates both functions inside the minimum in (2.9). The entropy is computed for a typical point from two separated clusters with 50 uniformly distributed points in each (*left two plots*), and lena 512×512 image (*right two plots*).

2.3.1 Reformulation in logarithmic scale and bounds for the root

Although we can employ any root-finding methods to solve (2.7), solve it in the β domain is very impractical. As we show in fig. 2.4 the entropy changes very dramatically for small values of β. Large β, on the contrary, results in the small values of the entropy that are numerically challenging. Thus, we propose to reformulate the problem in the logarithmic domain rather than linear (i.e. solve for $\log\beta$ instead of β). This changes the domain of the problem from $(0, +\infty)$ to $(-\infty, +\infty)$. The steep section on the left that previously was bounded by 0, now spans infinitely to $-\infty$, expanding the region where the entropy changes dramatically. Because of the continuity and bijection properties of the logarithm, we can always retrieve β once we have the solution for $\log\beta$. For the same reason, the analysis from the section 2.2, including the formula for bounds and monotonic decrease of the entropy with respect to $\log\beta$ still hold in case of logarithmic scale.

This reformulation slightly modifies the expressions for the derivatives of the entropy and its derivatives (cf. (2.2)):

$$\begin{aligned} F(\mathbf{y}, \alpha, K) &= e^{\alpha} m_1 + \log Z - \log K, \\ \frac{\partial F(\mathbf{y}, \alpha, K)}{\partial \alpha} &= e^{2\alpha}(m_1^2 - m_2), \\ \frac{\partial^2 F(\mathbf{y}, \alpha, K)}{\partial \alpha^2} &= 2e^{2\alpha}(m_1^2 - m_2) + e^{3\alpha}(m_3 - 3m_1 m_2 + 2m_1^3). \end{aligned}$$

In fig. 2.4 we show few examples of such reformulation. The region corresponding to the

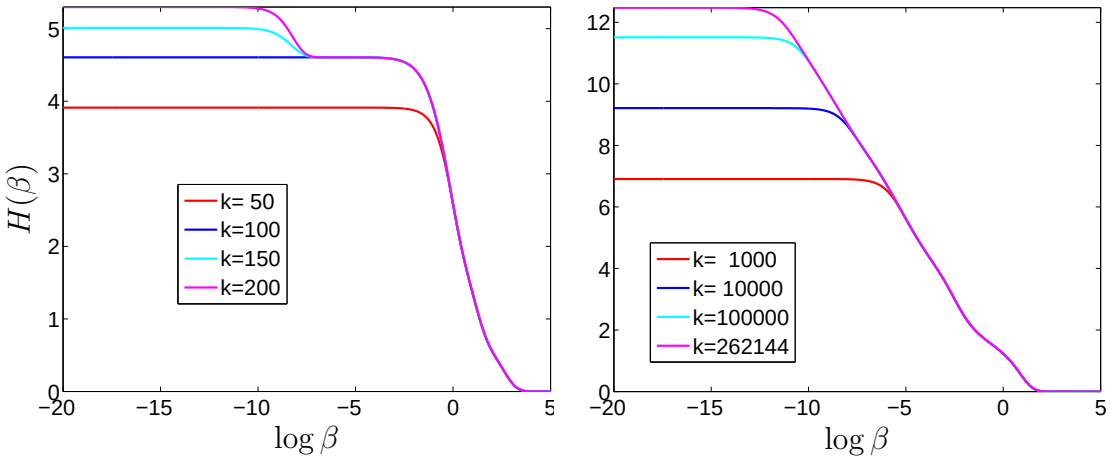

Figure 2.5: Number of nearest neighbors used in computing the entropy. *Left:* two separated clusters with 50 uniformly distributed points in each. *Right:* Lena 512×512 image.

points in the second cluster is impossible to see in the β domain, but is clearly visible in the $\log \beta$ domain. The right flat section, on the contrary, got shrank and becomes less problematic for the root-finding methods. The bounds also underline the structure of the entropy much better in case of logarithmic scale. They are quite tight and, except for the small region near the upper bound, do not include the flat, numerically challenging region of the function. Given that those bounds are computed in a constant time, this gives very good advantage to our algorithm.

2.3.2 Using sparse distance matrix

Because of the exponential decay in the entropy sum (2.7), distant points do not influence the entropy when $\log \beta$ is large. Therefore, when given perplexity K is small, we don't need to use distant neighbors, whose contribution to the entropy is almost 0. This allows sufficient decrease in runtime, as computing the entropy becomes $\mathcal{O}(k)$, instead of $\mathcal{O}(N)$, where k is a number of nearest neighbors that we keep. In fig. 2.5 we show how entropy changes with respect to $\log \beta$ for different values of nearest neighbors k. While k affects the solution when perplexity K is similar to k, when $K \ll k$, the solution is almost exactly equals to the case when $k = N$. Practically, we found that using $k \approx 5K$ gives very good speed up with almost same quality of the affinities as in the full case.

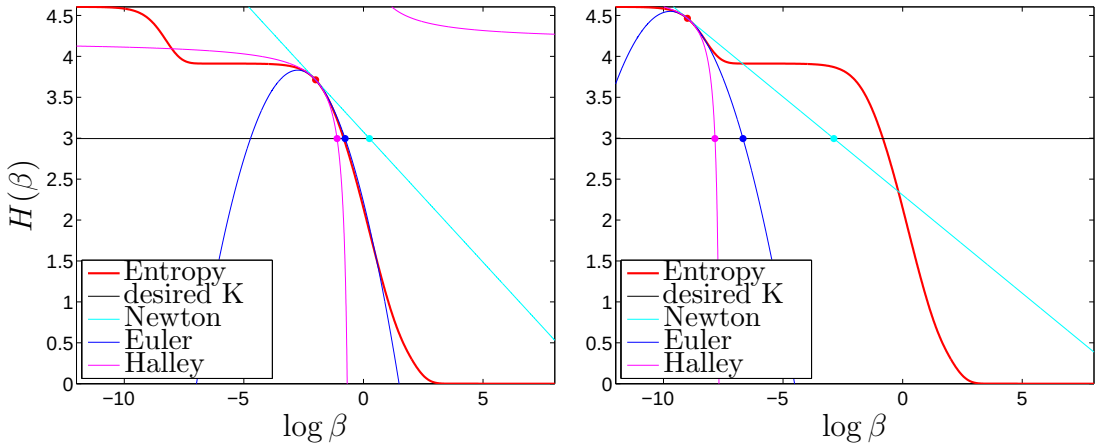

Figure 2.6: Examples of a step of different gradient-based root-finding algorithms initialized at $\beta_0 = -2$ (left plot) or $\beta_0 = -9$ (right plot).

Table 2.1: Comparison of different root-finding methods

Methods		Conv. order	Deriv. needed	$\mathcal{O}(N)$ eval.
Derivative-free	Bisection	linear	0	1
	Brent	linear	0	1
	Ridder	quadratic	0	2
Derivative-based	Newton	quadratic	1	2
	Euler	cubic	2	3
	Halley	cubic	2	3

2.3.3 Choice of the root-finding algorithm

In this section we will focus mostly on the derivative-based methods, since they have higher convergence rate. We first are going to describe three popular algorithms, namely Newton's, Euler's and Halley's methods and then propose a simple modification that gives global convergence guarantees to those algorithm. All the algorithms are iterative and for a given iteration $\log \beta_k$ they use the current value of F_k, first derivative g_k and, possibly, second derivative H_k to find the next iteration $\log \beta_{k+1}$. In fig. 2.6 we illustrate how the next iteration is computed for those methods.

Table 2.1 shows main properties of the root-finding methods described in this section.

Derivative-based root-finding algorithms

Newton's method. Newton's method is a second-order method that approximates the function with a line (i.e., up to a first derivative) and the next iteration is found by the intersection of the tangent of the current point with the x-axis. Newton's method approximates the function linearly, such that the line coincides with the current iteration $\log \beta_k$ in the function value value and its derivative. The next iteration is found using: $\log \beta_{k+1} = \log \beta_k - F_k/g_k$. The method has quadratic rate of convergence and requires only one derivative to be computed. In fig. 2.6 we show the iteration of the Newton method with a cyan line. Notice, that the Newton's iterates are the most aggressive among the others root-finding techniques.

Euler's method. This method uses first two derivatives and has a cubic rate of convergence. It approximates the function with a parabola that coincides with the current iteration $\log \beta$ in the function value and first two derivatives. The direction comes from solving the parabolic equation for the direction p_k:

$$F_k + g_k p_k + \frac{1}{2} H_k p_k^2 = 0, \tag{2.12}$$

whose solution is

$$p_k = \frac{-g_k \pm \sqrt{g_k^2 - 2H_k F_k}}{H_k}. \tag{2.13}$$

If both roots are real, we should take the one that is closer to the initial point. If roots are complex (i.e. when the parabola lies entirely above the root), we fall back to the Newton method. In fig. 2.6 we show the iteration of the Euler method with a blue line.

Halley's method. Similarly to Euler's method, Halley's method also uses first two derivative and also has cubic rate of convergence. However, instead of parabola, it approximates the function with a hyperbola that coincides with the current iteration in the function value and first two derivatives (Scavo and Thoo, 1995). The direction is given by

$$p_k = \frac{2F_k g_k}{2g_k^2 - F_k H_k}. \tag{2.14}$$

In fig. 2.6 we show the iteration of the Halley method with a magenta line. Halley's iterates are the most conservative among the other techniques.

Algorithm 1 Root-finding framework

Input: initial β, perplexity K, distances $d_1^2, \ldots, d_N^2$
compute bounds $\mathcal{B}$ using Theorem 2.2.2.
while *true* **do**
  for $k = 1$ **to** `maxit` **do**
    compute β using any derivative-based method
    if tolerance achieved **return** β
    if $\beta \notin \mathcal{B}$ **exit for loop**
    update $\mathcal{B}$
  end for
  compute β using bisection
  update $\mathcal{B}$
end while

Halley's method can also be interpreted as an order two Householder's method (with the first order being a Newton's method), that defines a direction p_k as:

$$p_k = -d\frac{(1/F_k)^{(d-1)}}{(1/F_k)^{(d)}}, \tag{2.15}$$

where d is the order of the method.

Modification to achieve convergence

As we mention above, the derivative-based methods may converge slowly or may not converge at all depending on the initialization. The most problematic are the flat regions of the entropy, where the methods are making really long step that send the iteration far away from the root. Another problem are cyclical iterations that bounce back and forth between same values. Here we propose a simple modification to all derivative-based methods that achieves a global convergence from any starting point.

We initialize the algorithm with an interval bracketing the root (obtained from the bounds in Theorem 2.2.2) and a starting point within those bounds. The algorithm consists of two nested loops: an outer loop with the bisection method, which is slow but guarantees global convergence, and an inner loop with a derivative-based method, which is fast but converges only locally. For each iteration of the inner loop, the algorithm evaluates the function, updates the bounds based on a new function value, computes

the necessary derivatives and applies a derivative-based method. If the output of the method falls outside of the current brackets or the number of iterations exceeds a certain constant `maxit`, the inner loop terminates and the next point is computed using the outer bisection loop. Thus the sequence of iterates contains a subsequence of bisection steps (every `maxit` steps at most), which necessarily converges. Practically, we use a rather big value of `maxit` = 20, since our good initialization (see below) makes it very infrequent for the derivative-based method step to fail. Algorithm 1 shows the framework.

To connect the methods, Gander (1985) shows that many of the root-finding methods can be described jointly using a simple framework. He proved that the iterative procedure $\beta_{k+1} = \beta_k - \frac{f(\beta_k)}{f'(\beta_k)} H(t(\beta_k))$, where $t(\beta_k) = \frac{f(\beta_k) f''(\beta_k)}{f'(\beta_k)^2}$ and H is some function, is of third-order convergence if $H(0) = 1$, $H'(0) = 1/2$. Indeed, the Halley's method is recovered with $H(t) = (1 - \frac{1}{2}t)^{-1}$, the Euler's method with $H(t) = 2(1 + \sqrt{1 - 2t})^{-1}$ and $H(t) = 1$ gives the Newton's method.

Moreover, it appears that locally, close to the root, it is not essential which derivative-based method is used, but how many of the derivatives it needs. Traub (1982) showed that, for any $p > 1$, given $p - 1$ derivatives of the function there exists no method with convergence order higher than p. Thus, if we use only one derivative, we cannot do better than second order convergence and locally, close to the root, the behavior of Newton's method is optimal. Similarly, for two derivatives, both Euler's and Halley's methods are optimal for third-order convergence methods. The differences between methods arise mostly when the iterations are far from the root.

Fig. 2.7 shows the difference in number of iterations for different root-finding algorithms. There, we initialized the three algorithms at different places within the bounds and computed how many and what kind of iterations of the Algorithm 1 they need in order to find the $\log \beta$ that corresponds to the perplexity $K = 30$ to an accuracy `tol` $= 10^{-10}$. We show two cases: a simpler one with nice behaving entropy for `cameraman` image dataset and more complicated case for two uniformly distributed clusters with 100 points each for perplexity $K = 99$. Notice that close to the root, Halley's and Euler' method behave almost identically to each other, while for Newton's method the region where the number of iterations equal to 1 is a lot smaller. This is caused by a higher convergence order of the former methods compared to the latter. However, in the region far from the root, the methods behave quite differently. In the flat regions of the space, the initial steps of Newton's and Euler's methods are too big and send the next iterate out of the

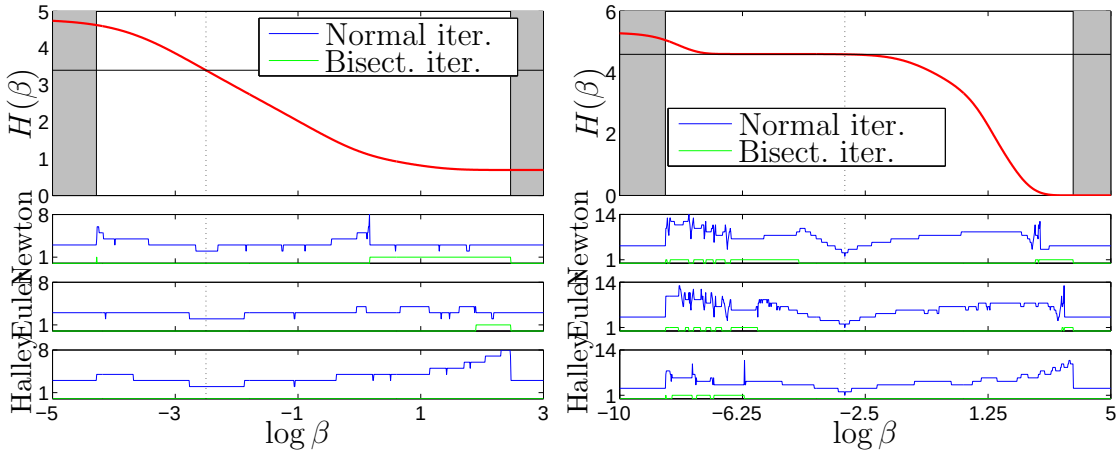

Figure 2.7: Top plot shows the entropy for different $\log \beta$ values and the bottom three plots show the number of iteration needed for different methods to achieve tolerance $tol = 10^{-10}$ if initialized from a given $\log \beta$ value. Dotted black vertical line corresponds to the desired $\log \beta$ value. Shaded region corresponds to the area outside of the bounds. *On the left:* the entropy of the typical point from cameraman image dataset, $K = 30$. *On the right:* more complicated example of two clusters far away from each other with 100 points each, $K = 99$.

bounds, causing our algorithm to use a bisection. The region where bisection iterations occur is smaller for Euler's method compared to that of Newton's method because the parabolic approximation leads to smaller steps than the linear one. Away from the root, Halley has this property of being conservative in the flat regions and having very small steps. For the most part, this is a bad thing, because a single bisection step will get much closer to the root (or even more importantly away from the flat region). The only scenario for which I see Halley method being preferred is in very beginning of the flat region, where even a small step can get us out from the flat part.

2.3.4 Warm-start initialization

We need to find a good initialization for the root-finding algorithm, i.e. as close as possible to the root. One way to do it is to provide precomputed initialization values directly to the algorithm. For example, we can initialize the algorithm from the middle of the bounds given by Theorem 2.2.2, or initialize $\log \beta$ from the distance to the kth neighbor. However, these initializations ignore the distances to most of the points, which affects the entropy and so the root of $F(\mathbf{y}, \beta, K)$. On the other side, we also do not want to include

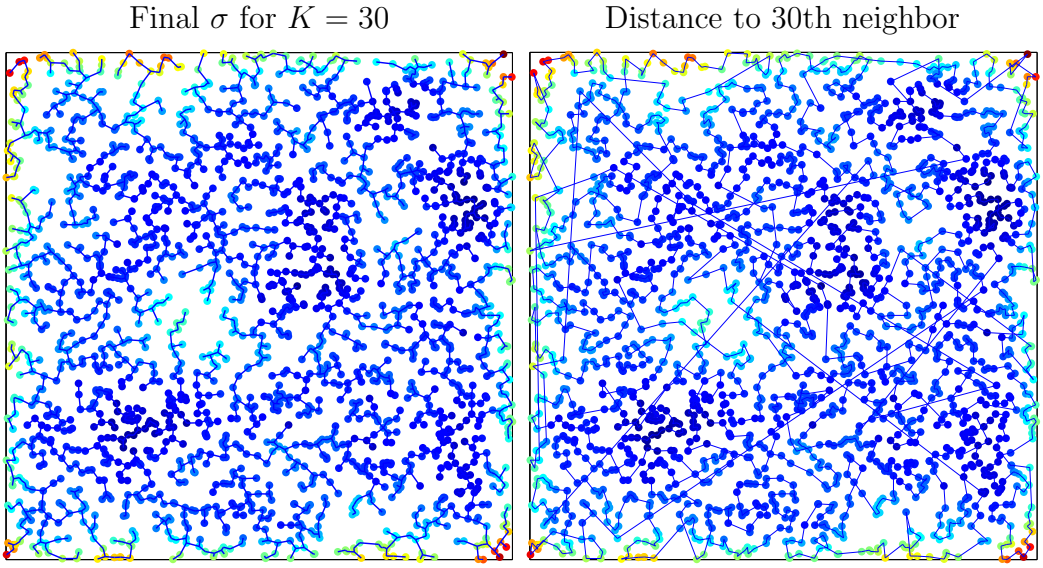

Figure 2.8: The order of points for different methods for 2 000 randomly distributed points in 2D. *Left:* MST order with $\kappa = 30$, *right:* MSP order with $\kappa = 100$. The colors correspond to the true σ value.

more information in the initialization if its computational cost becomes commensurate to evaluating the entropy function itself. Instead, we propose to capitalize on the correlation that exists between $\log \beta$ and the structure of the dataset. We can then link the points to each other based on some criterion and initialize the algorithm from the solution of the points for which $\log \beta$ was already found. Linking the points can be sequential or, more generally, based on a tree. In the sequential order each new point is initialized from the solution to the previous one. In case of the tree, the order is not linear, but forms a directed tree (or forest in general) with each point being a node. The points are then processed in an order (such as given by breadth-first search) ensuring that the root of a parent node is visited before the root of its children (which are initialized from the parent). For both sequential and tree orders, starting point can be initialized, for example, from the middle of the bounds. We now describe two different strategies for choosing the order and show how they are correlated with β.

Local strategy

This strategy is based on the existence and continuity of the function $\beta(\mathbf{y})$ defined in section 2.2. As we show in that section, continuous changes in $\mathbf{y}$ lead to continuous

changes in $\log \beta$, which means that nearby points should have similar $\log \beta$ values (except where $\beta(\mathbf{y})$ changes quickly). Therefore, it makes sense to initialize the points from their local neighbors for which we already know the solution.

First, we define a graph (V, E) where V are the vertexes that correspond to the set of data points and E are the edges that are represented by the matrix of square distances. This matrix is given to us, at least partially, in the definition of the problem. If all the distances are available, the graph is full, otherwise it is sparse. Below we propose different ways to build a local order of this graph and fig. 2.8 compares first two of them for a simple toy example.

Minimum Spanning Tree. By definition, Minimum Spanning Tree (MST) or more generally Minimum Spanning Forest (in case of disconnected components) has a minimal weight among all the possible spanning trees of the graph. Thus, traversing it we would be moving along only the smallest distances, which gives a local traversal.

To build the MST we used Kruskal's algorithm, which takes $\mathcal{O}(N\kappa \log N)$ time to construct, where κ is the degree of each of the vertex. MST is faster to compute for smaller κ, so it might be beneficial not to include far distances from the graph and use smallish κ value. However, it should be not too small for a graph to lose its connectivity.

Once the tree is constructed we need to traverse it in a manner that initializes the points from the parent that already been solved. This can be achieved by doing breadth-first search (BFS) on the tree and solving for the points that we encounter along the way. If we have multiple components we can either use initial starting point or initialize from the solution of the last point in previous component. We observed that the choice of the root point(s) for the tree does not critically affect the results. In fig. 2.8 we see that the edges of the tree roughly follow the σ values for the points (i.e. there are no edges between the points of drastically different color).

Minimum Spanning Path. However, building MST sometimes can be too expensive. In this case we can create a faster, but less accurate solution by approximating a minimum spanning path (the exact solution is TSP, witch is NP-complete). We can do it in $\mathcal{O}(N\kappa)$ by moving through the points, each time jumping to the closest unvisited point among the available neighbors. In case when all the points in the neighborhood are visited, we jump at random to some unvisited point. Although in most cases the sum of the distances of the path is not optimal, as we show in the experimental section, the

Algorithm 2 Minimum Spanning Path

```
Input: graph G = {E, V}, root r ∈ V
i = r, P = ∅
add i to P and remove it from V
while true do
  while exist neighbors of i in V do
    set j as a closest neighbor of i in V
    add j to P and remove it from V
    i = j
  end while
  if V = ∅ then
    return P
  else
    pick i at random from V
  end if
end while
```

performance usually is very close to the one from MST. Similar to MSP, we can farther sparsify the distance graph by leaving only κ nearest neighbor graph for some small κ. Algorithm 2 shows the pseudocode. The cost is linear in the number of edges of the graph: $\mathcal{O}(N\kappa)$. Fig. 2.8 demonstrates that the order also gives local result and roughly follows the σ values.

Raster order. For some specific domain, we can create an additional order that leverages the neighborhood structure that might be given indirectly as a part of the problem. For example, if the data points represent the pixels in the image (e.g. for image segmentation using spectral clustering), it is common to decode the spatial coordinates of the pixels as a parameters of that pixel (along with the range features such as pixel intensity or its color). In this case we can create a raster order by zigzagging edge-to-edge left to right and then right to left from the top to the bottom of the image. In this order, the spatial features vary only by an offset of 1 and the range features also change slowly (except at edges maybe). It is an example of local order with essentially no extra computation cost.

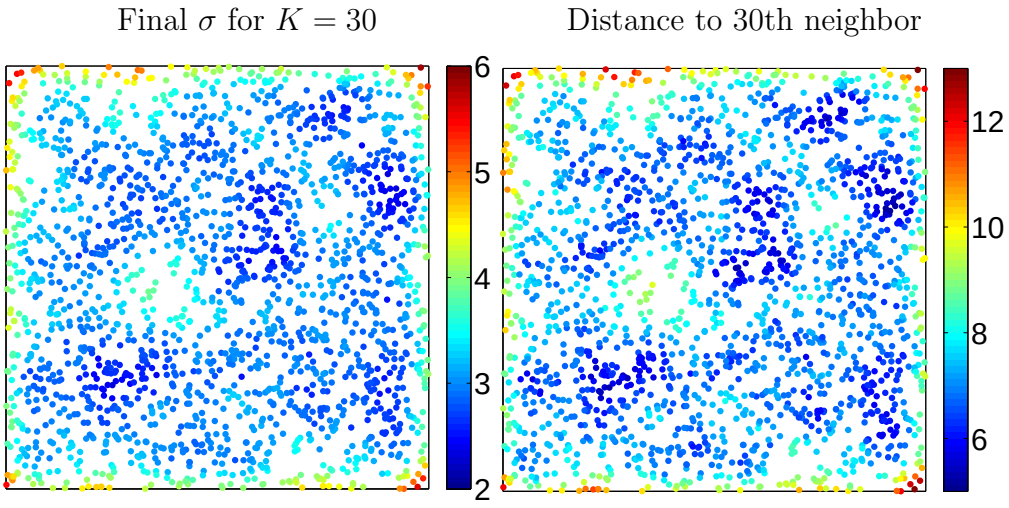

Figure 2.9: The order of points for different methods for 2 000 randomly distributed points in 2D. *Left:* final σ values with perplexity $K = 30$, *right:* distance to 30th neighbor of each point. The data is the same as in fig. 2.8.

Density strategy

Our second strategy, which we defined as $\mathcal{D}_K$, takes into account the *density* around the points. If we make the region denser by moving the points closer to $\mathbf{y}$, the distances d_k would become smaller and so will the entropy $H(\mathbf{y}, \beta)$. Therefore, for the entropy to remain constant, the resulting $\log \beta$ must be larger in dense regions and smaller in sparser ones. Indeed, β is related to a nonparametric density estimate of the dataset. Estimating the density robustly in the first place is not trivial, but we can use a simple estimate. First, we take the distance from the query point $\mathbf{y}$ to its kth neighbor. Then, we can sort the points $\mathbf{y}_1, \ldots, \mathbf{y}_N$ in increasing distance of its κth nearest neighbor, which gives a sequential order. Note this is different from the rule-of-thumb strategy of setting σ directly as the distance to kth neighbor (e.g. $k = 7$ in Zelnik-Manor and Perona, 2004).

Fig. 2.9 shows the final σ vales as well as the distance to 30th neighbor for some toy dataset. Notice that the right plot takes dramatically different values and therefore is not good for the initialization of root-finding algorithm. However, the order of points looks similar to the final case.

2.4 Experimental evaluation

2.4.1 Bounds quality.

We will start by evaluating the performance of the bounds. We have already shown in fig. 2.4 that the bounds are quite tight and mostly omit challenging flat regions. In addition to that experiment, in fig. 2.10 we show the upper and lower bound for two datasets. First dataset is 128×128 image from `cameraman`. Each data point in this dataset is a pixel represented by spatial and range features $(i, j, L, u, v) \in \mathbb{R}^5$ where (i, j) is the pixel location in the image and (L, u, v) the pixel value in a color image (overall $N = 16\,384$ points in $D = 5$ dimensions). For this dataset we seek $\log \beta$ that corresponds to the perplexity $K = 50$. As we see, both bounds are quite tight and also, from the visualization of the bounds and the solution, we can see that the bounds vary individually from point to point and the general structure of the bounds is related to the structure of the solution. The second dataset that we tried is three clusters with 500 points each that were generated with different means and variances. Here we are after finding the $\log \beta$ that corresponds to the perplexity $K = 10$. The tighter the points are inside their clusters, the larger the final $\log \beta$ should be. We clearly see it on the right plot. The final bounds are quite tight. Also, note that the upper bound follows the same structure as the final $\log \beta$.

2.4.2 Order comparison.

In order to have a benchmark for comparing the order, in addition to different local and density orders that we described in the section 2.3.4, we are going to compare to two more unrealistic orders. First, the *random* order, where the order comes from initializing each new point randomly without replacement. This order can serve us as an upper bound of our initializations. Second, the *oracle* order processes the points in the order of their true $\log \beta$ values, which is optimal in terms of initializing the points closest to the solution (although not practical, obviously). Finally, we are going to compare to the *bounds* initialization, which is not technically an order, but an initialization from the middle of the bounds. In fig. 2.11 we show different orders as a colormap for a `cameraman` image dataset.

First, we are going to see how MST, MSP and $\mathcal{D}_\kappa$ are affected by the changes in κ. For MST and MSP κ changes the number of nearest neighbors available for the construction

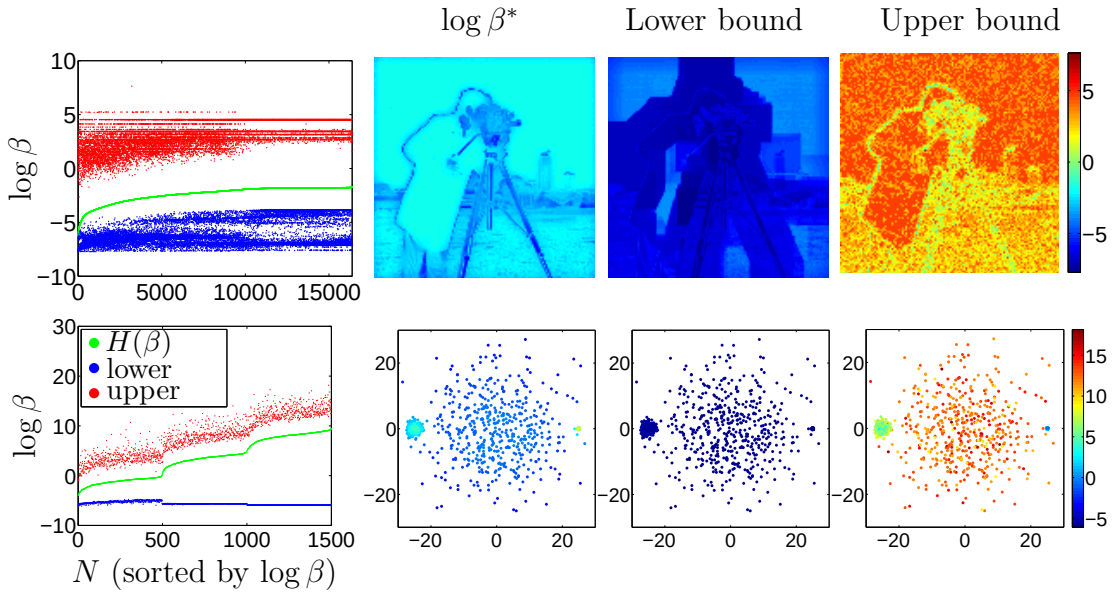

Figure 2.10: Bounds and final $\log \beta^*$ values for cameraman image dataset for perplexity $K = 50$ (*top plot*) and three Gaussian clusters dataset with 500 points each for $K = 10$ (*bottom plot*). *From left to right:* (1) lower, upper and final $\log \beta^*$ sorted by the value of the $\log \beta^*$; (2) final $\log \beta^*$ for every point in the dataset (3) lower bound for every point in the dataset (4) upper bound for every point in the dataset.

of the order. For $\mathcal{D}_\kappa$ κ represents which nearest neighbor we take to measure the density of points. As a measure of order quality we used the following metric. Assume that the final $\log \beta$ are already given. Then, a good order will minimize the distance between the final $\log \beta$ and the value where it has been initialized from, i.e. a previous $\log \beta$ value:

$$S_\beta = \sum_{n=1}^{N} \left| \log \beta_n - \log \beta_{k(n)} \right|, \tag{2.16}$$

where $k(n)$ is the index of the parent of the element n along the path.

In fig. 2.12 we measure the quality of different orders as κ grows for a `cameraman` image dataset as we look for β that corresponds to the perplexity $K = 30$. Apart from S_β, as a measure of the order distance in the $\log \beta$ space, we also show $S_{\mathbf{y}}$, which measures the path length but in the original Euclidean space: $S_{\mathbf{y}} = \sum_{n=1}^{N} \left\| \mathbf{y}_n - \mathbf{y}_{k(n)} \right\|^2$. Last plot shows the runtime needed to compute the order for different κ.

First, notice that the oracle order is much better in β space comparing to all the other orders. Second, the MST order gives robustly good performance for a large range of κ values. Given the linear runtime with respect to κ (see the last plot), this suggests using

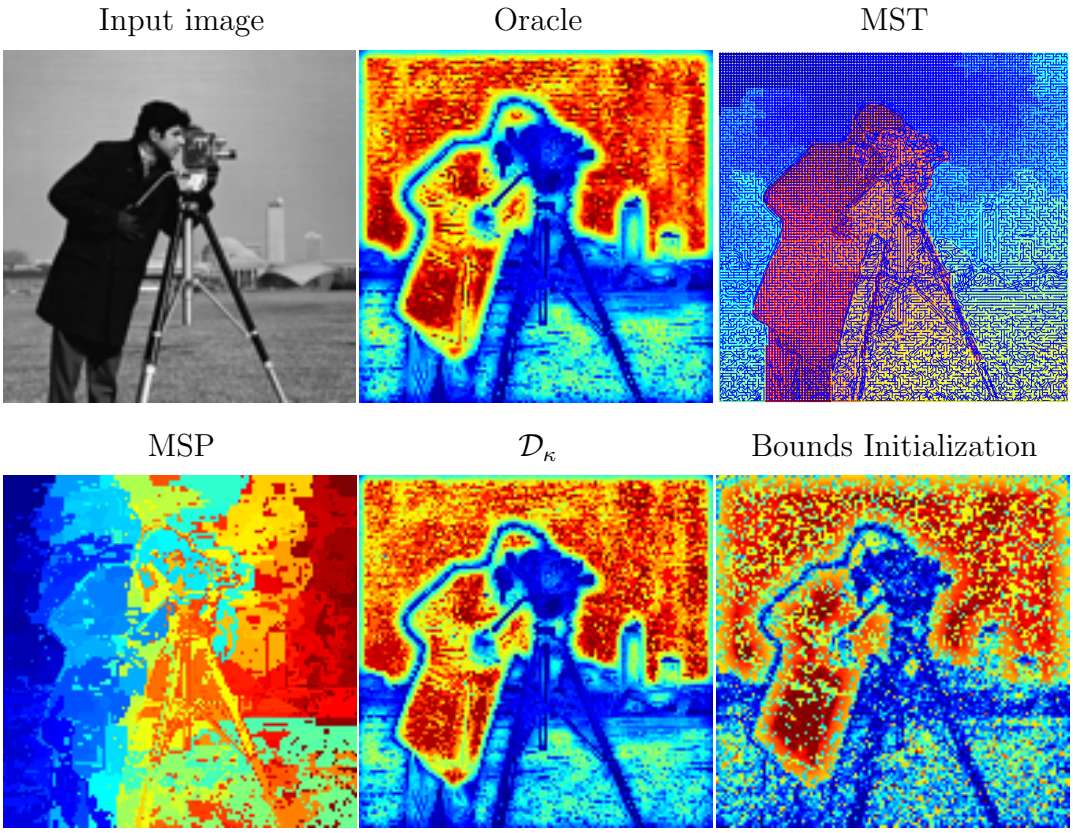

Figure 2.11: Example of different orders for a cameraman image dataset. Colormap in MST order corresponds to the initialization order of the points (BFS traversal). Bounds initialization is given by sorting σ values given by the middle of the bounds. For all the orders $\kappa = 30$.

MST with small κ to achieve larger speedups. Next, the MSP results in smaller path in β space as we increase κ, while the runtime almost doesn't change. Thus, we suggest using largest κ possible that gives better performance with only small increase over the runtime. For $\mathcal{D}_\kappa$ order the smallest path β space occurs for κ that is slightly larger than K. In our experiments we would take $\kappa = K$ as it gives reasonable performance. For the other orders, as we expected we see that MST and MSP are both better than the raster order, which in turn worse than initialization from the middle of the bounds, which is worse than random. By looking at $S_{\mathbf{y}}$ plot we see that MST, MSP and raster also give smaller path distance in the original Euclidean space, which make sense, because those orders are local.

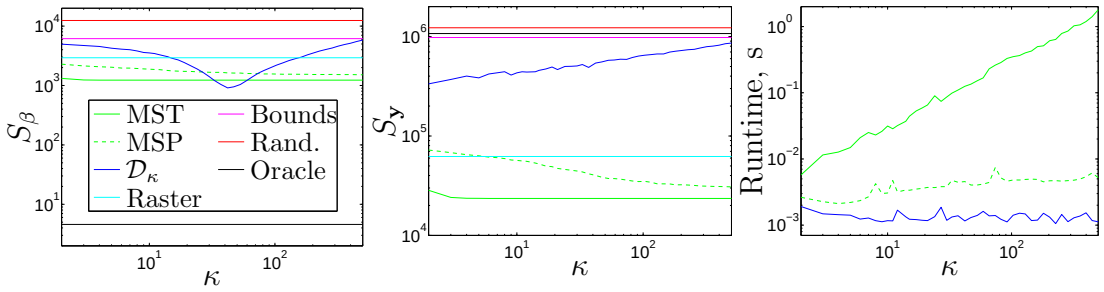

Figure 2.12: Analysis of κNN orders for cameraman image for perplexity $K = 30$. *Left plot* shows the path distance in the $\log \beta$ space for different orders, *central plot* shows the path distance in the original space and *right plot* shows the runtime needed to construct different orders.

2.4.3 Root-finding comparison.

We compare several different root-finding algorithms: Ridder's, Brent's and bisection for derivative-free methods, and Euler's, Halley's and Newton's for derivative-based methods. The former group of methods were run as is without any modification. The latter group were incorporated into our globally convergent framework (Algorithm 1). First we show how well those root-finding algorithms converge. In fig. 2.13 we show the number of points converge to a certain tolerance at the initialization and after 2nd, 4th, 6th and 20th iteration as we look for $\log \beta$ that corresponds to the perplexity $K = 30$ in cameraman image dataset. For the order, we used MST local order. At the initialization, the derivative-based method are already give better solution than derivative-free methods with more than half points already converged to 10^{-3} tolerance. After two iterations, third order method (Euler and Halley) already have more than 99% of the points already converged to the machine precision tolerance (we limit it to 10^{-14} because the lower values are already too close to the machine precision and oscillations start to occur because of precision loss). Second order methods converge a bit slower, Newton's method takes four iterations for 99% of the points converge to the tolerance $\leq 10^{-14}$ and Ridder's method – six iterations. Finally, for the first order methods, Brent's method needs around 20 iterations and bisection needs around 50 iterations.

2.4.4 Evaluation of β for different datasets.

All the user needs to do to obtain entropic affinities for a given dataset is to set the perplexity K and possibly their sparsity level. For the experiments, we used $K = 30$ and a

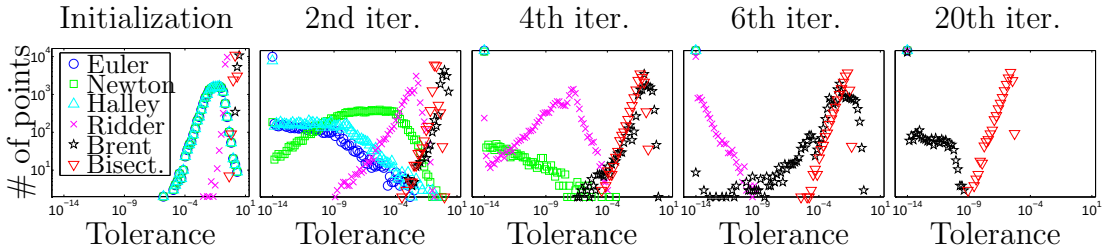

Figure 2.13: *From left to right:* tolerance achieved by different root-finding methods for every point in the dataset at initialization and after 2nd, 4th, 6th and 20th iteration. The dataset is 128×128. cameraman image. Order is given by MST algorithm.

Figure 2.14: Lena test image and the learned β values for every pixel for a perplexity $K = 30$.

sparse distance matrix with nonzero elements corresponding to the 250 nearest neighbors. This almost did not alter the final result, because the terms in (2.2) responsible for farther distances are negligible due to the fast decay of the Gaussian tails. For all the algorithms we set the convergence tolerance to 10^{-10}. We chose such an accurate value because, with quadratic or cubic convergence rate, one needs very few additional iterations to achieve such accuracy, and the user need not worry about setting the tolerance (as we show in fig. 2.13). We used three datasets from very different domains: pixels of a single image, an image collection, and word-count vectors from documents.

The first dataset is the 512×512 Lena image (fig. 2.14 left). Similar to the cameraman in the experiment above, each data point in this dataset is a pixel represented by spatial and range features $(i, j, L, u, v) \in \mathbb{R}^5$ where (i, j) is the pixel location in the image

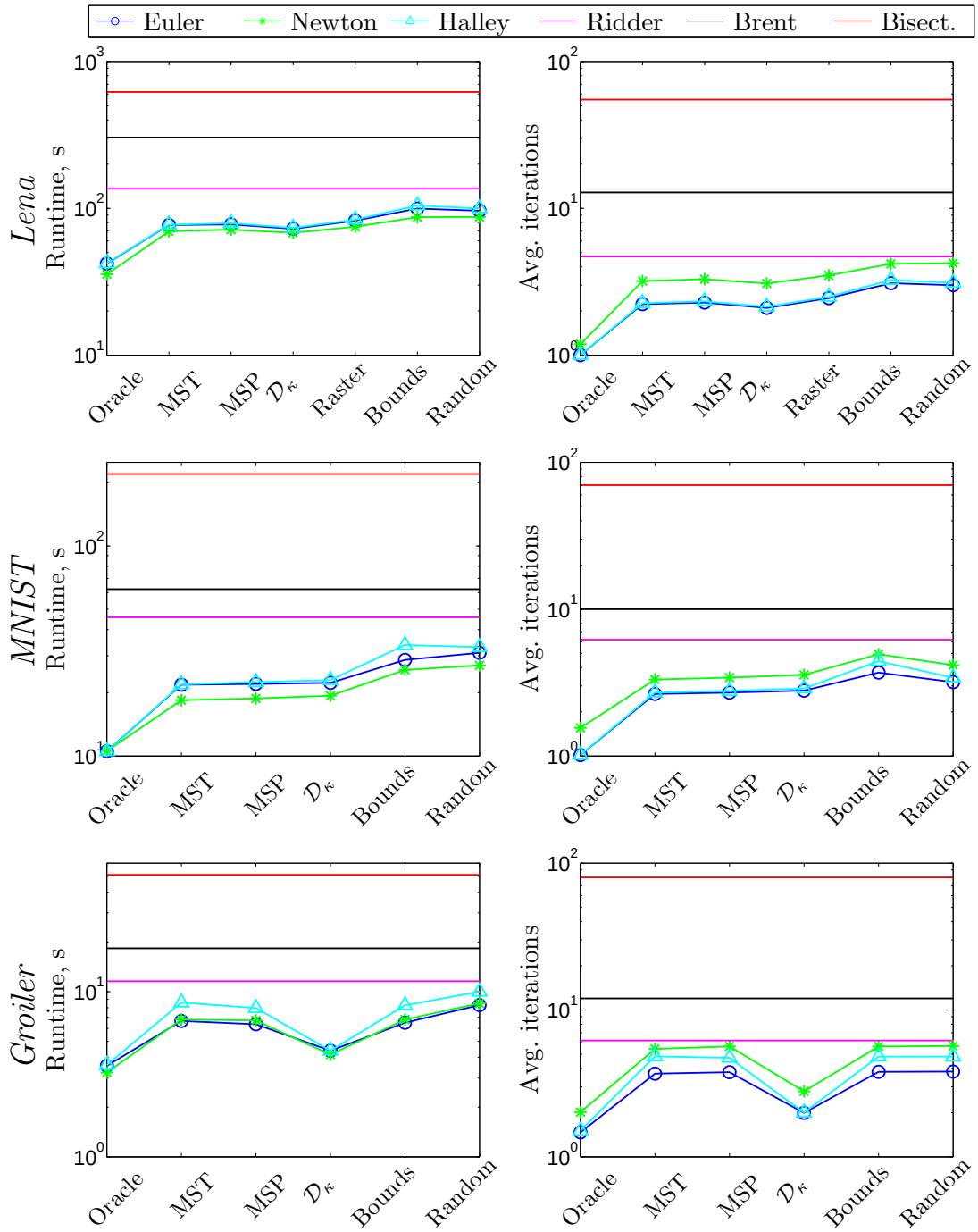

Figure 2.15: Runtime statistics for 3 different datasets, with 6 root-finding methods (color-coded as in the legend). *From top to bottom*: 512×512 Lena image, 60 000 MNIST digits, 30 991 articles from Grolier's encyclopedia. runtime (*left*) and average number of iterations (*right*) for different methods with different initializations. Note the log scale in many of the plots.

and (L, u, v) the pixel value in a color image (overall $N = 262\,144$ points in $D = 5$ dimensions).

Fig. 2.14 (right) shows the resulting β. It preserves much information about the image, in particular edges. β tends to change gradually from pixel to pixel; small values correspond to dark regions of space and big ones to bright regions. The final β and the corresponding affinity matrix can be used for segmenting the image using mean-shift or spectral methods, for example.

For the second dataset we used 60 000 handwritten digits from the MNIST dataset. Finally, the third dataset is a subset from Grolier's encyclopedia dataset. There, each datapoint represents the word count of the most popular 15 275 words from one of the 30 991 articles of the encyclopedia. Compared with the first two datasets (especially the first one), where there are reasonably densely populated areas of input space, the third dataset mostly consists of empty space. The points are located far away from each other and β values of neighboring points are not similar anymore. This will be clearly visible in the results. The affinities resulting from the MNIST and Grolier datasets can be used for visualization using dimensionality reduction, spectral clustering or semi-supervised learning, for example.

For each of the datasets, we present three statistics. Fig. 2.15 shows the total runtime for different root-finding algorithms with different initializations and also the average number of iterations per point required to achieve the tolerance. Fig. 2.16 we show the number of the points that converged for a given number of iterations for different orders and different datasets. For the derivative-based orders, most of the points need only one or two iterations to converge. Comparing to that, derivative-free methods, such a Ridder and Brent need anywhere from 5 to 15 iteration and Biseection method need 30 to 100 iterations.

First of all, notice that Halley's and Euler's methods have very similar performance for all the datasets. Both methods require only two iterations for most of the points. However, there is a small difference, in particular for the *bounds* initialization of the MNIST dataset and for the *MST*, *bounds* initialization and *random* order in the Grolier dataset. The reason is the initialization in the flat region for many of the points (note that the result of those initializations is not good compared to e.g. the $\mathcal{D}_\kappa$ order). Similar to what we see in the flat region of fig. 2.7, Halley's method is more conservative and moves slowly towards the solution, whereas Euler's method uses steps that are too big

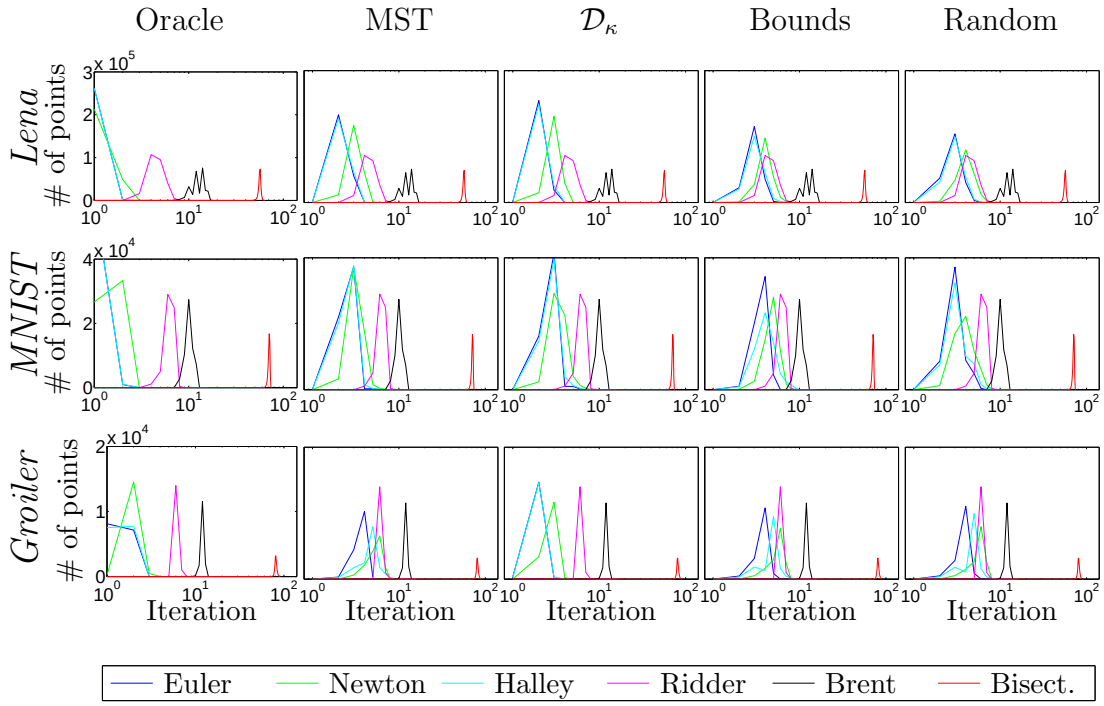

Figure 2.16: Cumulative number of points that converged for a given number of iterations for different orders and different datasets. Notice the log plot for the number of iterations.

and retreats to bisection, which moves away from the flat region in one iteration.

Compared to the other derivative-based root-finding methods, Newton's method is a second-order method and requires slightly more iterations than Halley's or Euler's methods. However, its runtime is lower because each iteration does not need to compute the second derivative of the entropy, which costs $\mathcal{O}(N)$. For the derivative-free methods, Ridder's method is fastest, but is still approximately twice as slow as Newton's method. Brent's method and the bisection are approximately $5\times$ and $10-20\times$ as slow as Newton's method, respectively.

For different initializations, *MST* and $\mathcal{D}_\kappa$ have very similar results for the MNIST and Lena datasets, with $\mathcal{D}_\kappa$ being only slightly better (e.g. for Euler's method in the Lena dataset it takes 2.09 iterations on average for $\mathcal{D}_\kappa$ compared to 2.22 for *MST*). However, for the Grolier dataset the *MST* order does almost as badly as the *random* order. This is due to the spatial emptiness that we described above. This does not seem to affect the $\mathcal{D}_\kappa$ order, which is only 22% slower than the *oracle* order. The *bounds* initialization and *random* order perform almost identical to each other and not terribly bad, only about

1–2 iterations more than the other initializations. However, for 50% of the points, the extra iterations are the bisection iterations during the first steps of the algorithm when the initial region is flat and the root-finding methods send us away from the bounds. This also indicates that the bounds are quite tight and one or two extra iterations are able to move very close to the root no matter where the initialization is. The *raster* order for Lena dataset does almost as good as the *MST* order, suggesting it as an fast alternative local order for image pixels. Finally, the *oracle* order achieves nearly 1 iteration per point for Euler's and Halley's method on Lena and MNIST. For example, for Euler's method only 0.1% of all the points needed 2 iterations to converge. However, in terms of the runtime the *MST* and $\mathcal{D}_\kappa$ orders achieve a speed that is only twice as slow as the optimal one. For the Grolier dataset the average number of iterations per point is more than one even for the *oracle* order. This is because, even in the best case, the σs are not as close to each other as in the Lena and MNIST datasets.

2.5 Discussion

The entropic affinities can give high-quality results with many different machine learning algorithms that are based on graphs (as illustrated in fig. 2.1), and only require the user to set the perplexity K and possibly the sparsity level of the affinity matrix. However, up to now they have not been in widespread use outside nonlinear embedding methods such as SNE (Hinton and Roweis, 2003) or EE (Carreira-Perpiñán, 2010). Reasons for this could be the lack of a closed-form expression for the bandwidth of each point given the perplexity, and the (up to now) computational cost involved in solving for it. With our numerical algorithms, there is now very little difference between applying a closed-form formula and solving for the implicit bandwidths almost exactly. This is for two reasons. First, even if a user is able to compute bandwidths very efficiently (e.g. with a rule-of-thumb formula), computing the elements of the affinity matrix themselves is still $\mathcal{O}(N^2)$ or $\mathcal{O}(N\kappa)$ in the full and sparse case, respectively. Each iteration of our root-finding method has this same cost, but (1) we require just a few such iterations, and (2) the affinities are produced for free in our last iteration. Thus, the cost of applying the rule-of-thumb formula to compute the affinity values given the bandwidths is comparable to that of computing entropic affinities and their bandwidths. Second, we can achieve near-machine-precision at nearly no extra cost because of the high order of convergence

of the root-finding methods.

The bisection algorithm used by Hinton and Roweis (2003), although slow, was not much of a problem up to now because the optimization in methods such as SNE was so costly that the number of points N was limited to a few thousands, for which the bisection time was acceptable. However, due to recent improvements in embedding optimization that we are going describe in details in chapters 3,4 and 5 have significantly increased the values of N that are practical: for example, using the spectral direction, the embedding optimization takes 10 min for 20 000 MNIST images in a workstation, while the bisection-based computation of the entropic affinities takes over 20 min and becomes a bottleneck. With our algorithm, this time is reduced to 55 seconds.

It is interesting to problem to use entropic affinities for the problems other than manifold learning, but that still require an affinity matrix. Such problems include, e.g. spectral clustering Ng et al. (2002) and mean-shift clustering Carreira-Perpiñán (2006). In these problems, instead of showing the underlying manifold, the problem is to emphasize the clustering structure of the data. The latter can be formulated inside the affinity matrix as having high affinity between points in the same cluster and no or small affinity for the points in different clusters. The entropic affinities can produce the desired matrix as long as the perplexity K is smaller than the number of points in each cluster. Otherwise, the bandwidth would be too big and start affecting other classes.

2.6 Conclusion

By extending the entropic affinity function to the entire Euclidean space, we have been able to characterize its behavior, show that it is a well-defined function and give explicit bounds for its implicitly defined value. Based on these properties, we have analyzed different algorithms for the computational problems involved: root-finding and ordering points for best initialization. One of the best and simplest choices is a Newton-based iteration, robustified with bisection steps, using a tree- or density-based order. This achieves just above one iteration per data point on average, which is the optimally achievable performance.

Entropic affinities work better than using a single bandwidth or multiple bandwidths set with a rule of thumb, provide a random-walk matrix for a dataset, and only require a user to set the global number of neighbors. The fact that they define the scale implicitly

and require an iterative computation may have prevented their widespread application, but our algorithm makes the computation scale up almost as if they were given in explicit form.

Chapter 3

Partial-Hessian Strategies for Fast Learning of Nonlinear Embeddings

3.1 Introduction

In this chapter we are going to consider the that the affinity matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ is already given (e.g. using entropic Gaussian affinities set with an entropic affinities discussed in chapter 2) and it corresponds to pairs of high-dimensional points $\mathbf{y}_1, \dots, \mathbf{y}_N$, which need not be explicitly given, and we want to obtain corresponding low-dimensional points $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^d$ whose Euclidean distances optimally preserve the similarities. As we already discussed in the introduction, there exist many methods that are able to obtain those images, such as linear methods (e.g. PCA or MDS), spectral methods (e.g. LE or ISOMAP) and nonlinear embeddings (NLE; e.g. SNE, t-SNE, EE). We showed that latter class of methods are able to give much better embedding quality comparing to other methods. However, a fundamental problem with NLE has been their difficult optimization. First, they can converge to bad local optima. In practice, this can be countered by using a good initialization (e.g. from spectral methods), by simulated an-

This chapter is an extended version of Vladymyrov and Carreira-Perpiñán (2012).

nealing (e.g. adding noise to the updates; Hinton and Roweis, 2003) or by homotopy methods (Memisevic, 2006; Carreira-Perpiñán, 2010). Second, numerical optimization has been found to be very slow. Most previous work has used simple algorithms, some adapted from the neural net literature, such as gradient descent with momentum and adaptive learning rate, or conjugate gradients. These optimizers are very slow with ill-conditioned problems and have limited the applicability of nonlinear embedding methods to small datasets; hours of training for a few thousand points are typical, which rules out interactive visualization and allows only a coarse model selection.

The goal of this chapter is to devise training algorithms that are not only significantly faster but also scale up to larger datasets and generalize over a family of NLE algorithms (SNE, t-SNE, EE and others). We do this not by simply using an off-the-shelf optimizer, but by understanding the common structure of the Hessian in these algorithms and their relation with the graph Laplacian of spectral methods. Thus, our first task is to provide a general formulation of NLE (section 3.2) and understand their Hessian structure, resulting in several optimization strategies (section 3.3). We then empirically evaluate them (section 3.4) and conclude by recommending a strategy that is simple, generic, scalable and typically (but not always) fastest — by up to two orders of magnitude over the existing methods.

3.2 A General Embeddings Formulation

Call $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ the $d \times N$ matrix of low-dimensional points, and define an objective function:

$$E(\mathbf{X}; \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \qquad \lambda \geq 0 \tag{3.1}$$

where E^+ is the *attractive term*, which is often quadratic psd and minimal with coincident points, and E^- is the *repulsive term*, which is often nonlinear and minimal when points separate infinitely. Optimal embeddings balance both forces. Both terms depend on $\mathbf{X}$ through Euclidean distances between points and thus are shift and rotation invariant. We obtain several important special cases:

Normalized symmetric methods minimize the KL divergence between a posterior probability distribution Q over each point pair normalized by the sum over all

point pairs (where K is a kernel function):

$$q_{nm} = \frac{K(\|\mathbf{x}_n - \mathbf{x}_m\|^2)}{\sum_{n',m'=1}^{N} K(\|\mathbf{x}_{n'} - \mathbf{x}_{m'}\|^2)}, \quad q_{nn} = 0$$

and a distribution P analogously defined on the data $\mathbf{Y}$ (thus constant wrt $\mathbf{X}$) with possibly a different kernel and width. Notice that the probabilities Q are defined explicitly with bandwidth $\sigma = 1/\sqrt{2}$. setting the bandwidth to some other value just rescales the final embedding. This is equivalent to choosing

$$\begin{aligned} E^+(\mathbf{X}) &= -\sum_{n,m=1}^{N} p_{nm} \log K(\|\mathbf{x}_n - \mathbf{x}_m\|^2), \\ E^-(\mathbf{X}) &= \log \sum_{n,m=1}^{N} K(\|\mathbf{x}_n - \mathbf{x}_m\|^2) \end{aligned}$$

and $\lambda = 1$ in eq. (3.1). Particular cases are s-SNE (Cook et al., 2007) and t-SNE, with Gaussian and Student's t kernels, respectively. We will call $p_{nm} = w^+_{nm}$ from now on.

Normalized nonsymmetric methods consider instead per-point distributions P_n and Q_n, as in the original SNE (Hinton and Roweis, 2003). Their expressions are more complicated and we focus here on the symmetric ones.

Unnormalized models dispense with distributions and are simpler. For a Gaussian kernel, in the elastic embedding (EE; Carreira-Perpiñán, 2010) we have

$$E^+(\mathbf{X}) = \sum_{n,m=1}^{N} w^+_{nm} \|\mathbf{x}_n - \mathbf{x}_m\|^2, \qquad E^-(\mathbf{X}) = \sum_{n,m=1}^{N} w^-_{nm} e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2},$$

where $\mathbf{W}^+$ and $\mathbf{W}^-$ are symmetric nonnegative (with $w^+_{nn} = w^-_{nn} = 0$, $n = 1, \dots, N$).

Spectral methods such as Laplacian eigenmaps or LLE define

$$E^+(\mathbf{X}) = \sum_{n,m=1}^{N} w^+_{nm} \|\mathbf{x}_n - \mathbf{x}_m\|^2, \qquad E^-(\mathbf{X}) = 0, \tag{3.2}$$

with nonnegative affinities $\mathbf{W}$, but add quadratic constraints to prevent the trivial solution $\mathbf{X} = \mathbf{0}$. So E^+ is as in EE and SNE.

This formulation suggests previously unexplored algorithms, such as using an Epanechnikov kernel, or a t-EE, or using homotopy algorithms for SNE/t-SNE, where we follow the optimal path $\mathbf{X}(\lambda)$ from $\lambda = 0$ (where $\mathbf{X} = \mathbf{0}$) to $\lambda = 1$. It can also be extended to closely related methods for embedding (kernel information embedding; Memisevic, 2006) and metric learning (Peltonen and Kaski, 2005; Goldberger et al., 2004), among others. However, this formulation is not complete. For example, it does not cover NeRV Venna et al. (2010), which propose to have a sum of two divergences, P over Q and Q over P, that are weighted by some user parameter.

We express the gradient and Hessian (written as matrices of $d \times N$ and $Nd \times Nd$, resp.) in terms of Laplacians, following Carreira-Perpiñán (2010), as opposed to the forms used in the SNE papers. This brings out the relation with spectral methods and simplifies the task of finding pd terms. Given an $N \times N$ symmetric matrix of weights $\mathbf{W} = (w_{nm})$, we define its graph Laplacian matrix as $\mathbf{L} = \mathbf{D} - \mathbf{W}$ where $\mathbf{D} = \operatorname{diag}(\sum_{n=1}^{N} w_{nm})$ is the degree matrix. Likewise we get $\mathbf{L}^+$ from w^+_{nm}, $\mathbf{L}^q$ from w^q_{nm}, etc. $\mathbf{L}$ is psd if $\mathbf{W}$ is nonnegative (since $\mathbf{u}^T\mathbf{L}\mathbf{u} = \frac{1}{2}\sum_{n,m=1}^{N} w_{nm}(u_n - u_m)^2 \geq 0$). The Laplacians below always assume summation over points, so that the dimension-dependent $Nd \times Nd$ Laplacian $\mathbf{L}^{xx}$ (from weights $w^{xx}_{in,jm}$) contains $N \times N$ Laplacian for each pair of dimensions indexed by (i, j). All other Laplacians are dimension-independent, of $N \times N$. Using this convention, we have for normalized symmetric models:

$$\begin{aligned} \nabla E &= 4\mathbf{X}\mathbf{L} \\ \nabla^2 E &= 4\mathbf{L} \otimes \mathbf{I}_d + 8\mathbf{L}^{xx} - 16\lambda \operatorname{vec}(\mathbf{X}\mathbf{L}^q) \operatorname{vec}(\mathbf{X}\mathbf{L}^q)^T \end{aligned} \tag{3.3}$$

where $\mathbf{I}_d$ is the $d \times d$ identity matrix and we define the following scalar functions ($'$, $''$ are derivatives):

$$K = \text{kernel}, \quad K_1 = (\log K)' = K'/K, \quad K_2 = K''/K$$
$$K_{21} = (\log K)'' = (KK'' - (K')^2)/K^2 = K_2 - K_1^2$$

and weights (K_1 means $K_1(\|\mathbf{x}_n - \mathbf{x}_m\|^2)$, etc.)

$$w_{nm} = -K_1\,(p_{nm} - \lambda q_{nm}) \qquad w^q_{nm} = K_1\,q_{nm}$$
$$w^{xx}_{in,jm} = -\,(K_{21}\,p_{nm} - \lambda K_2\,q_{nm})\,(x_{in} - x_{im})(x_{jn} - x_{jm}).$$

In particular, for s-SNE the weights are as follows:

$$w_{nm} = p_{nm} - \lambda q_{nm} \qquad w^q_{nm} = -q_{nm}$$
$$w^{xx}_{in,jm} = \lambda q_{nm}(x_{in} - x_{im})(x_{jn} - x_{jm})$$

and for t-SNE they are (K means $1/(1+\|\mathbf{x}_n-\mathbf{x}_m\|^2)$):

$$w_{nm} = (p_{nm} - \lambda q_{nm})K \qquad w^q_{nm} = -q_{nm}K$$
$$w^{xx}_{in,jm} = -(p_{nm} - 2\lambda q_{nm})(x_{in} - x_{im})(x_{jn} - x_{jm})K^2.$$

For the elastic embedding (an unnormalized model):

$$\nabla E = 4\mathbf{X}\mathbf{L} \qquad \nabla^2 E = 4\mathbf{L}\otimes\mathbf{I}_d + 8\mathbf{L}^{xx} \tag{3.4}$$
$$w_{nm} = w^+_{nm} - \lambda w^-_{nm} e^{-\|\mathbf{x}_n-\mathbf{x}_m\|^2}$$
$$w^{xx}_{in,jm} = \lambda w^-_{nm} e^{-\|\mathbf{x}_n-\mathbf{x}_m\|^2}(x_{in} - x_{im})(x_{jn} - x_{jm}).$$

Note the Hessian of the spectral method (i.e., for $\lambda = 0$, with constant weights w^+_{nm}) is constant: $\nabla^2 E = 4\mathbf{L}^+ \otimes \mathbf{I}_d$.

3.3 Partial-Hessian Strategies

Our goal is to achieve search directions that are fast to compute, scale up to larger N, and lead to global, fast convergence. This rules out computing the entire Hessian. Carreira-Perpiñán (2010) derived pd directions for EE by using splits of the gradient such as $\nabla E = 4\mathbf{X}(\mathbf{D}^+ + (\mathbf{L}-\mathbf{D}^+)) = \mathbf{0}$ (where $\mathbf{D}^+$ is the degree matrix of $\mathbf{L}^+ = \mathbf{D}^+ - \mathbf{W}^+$), then deriving a fixed-point iterative scheme (à la Jacobi) such as $\mathbf{X} = \mathbf{X}(\mathbf{D}^+ - \mathbf{L})(\mathbf{D}^+)^{-1}$ and a search direction $\mathbf{X}(\mathbf{D}^+ - \mathbf{L})(\mathbf{D}^+)^{-1} - \mathbf{X}$. Here we use a more general approach that illuminates the merits of each method, by directly working with the Hessian $\nabla^2 E$. We define directions $\mathbf{p}_k \in \mathbb{R}^{Nd}$ of the form $\mathbf{B}_k\mathbf{p}_k = -\mathbf{g}_k$ where $\mathbf{g}_k$ is the gradient at iteration k and $\mathbf{B}_k$ is a pd matrix (this ensures a descent direction: $\mathbf{p}_k^T\mathbf{g}_k < 0$), and use a line search on the step size $\alpha_k > 0$ satisfying the Wolfe conditions to obtain the next iterate $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$ (Nocedal and Wright, 2006). This defines a range of methods from $\mathbf{B}_k = \mathbf{I}$ (gradient descent, very slow) to $\mathbf{B}_k = \nabla^2 E(\mathbf{X}_k)$ (Newton's direction, which would require modification to ensure descent but is too expensive anyway). We construct $\mathbf{B}_k$ as a psd part of the Hessian at $\mathbf{x}_k$ (our *partial Hessian*). Inspection of the very special structure of $\nabla^2 E$ in eqs. (3.3) and (3.4) immediately shows what parts we can use. Our driving principle is to use as much Hessian information as possible that is psd, fast to compute and leads to an efficient solution of the $\mathbf{p}_k$ linear system (e.g. sparse or constant $\mathbf{B}_k$). Note computing E or ∇E is $\mathcal{O}(N^2 d)$, but solving a Hessian nonsparse linear system is $\mathcal{O}(N^3 d^3)$.

Search directions For normalized symmetric (and nonsymmetric) models (3.3), we consider functions K with a nonnegative argument $t \geq 0$ and satisfying $K(t) > 0$ and $K'(t) < 0$, i.e., positive and decreasing. The term on $\mathbf{L}$ contains a psd part $-K_1 p_{nm}$ (which is constant for SNE and EE) and a nsd part $\lambda K_1 q_{nm}$; the term on $\mathbf{L}^{xx}$ is only guaranteed to contain a psd part for $i = j$ and depending on the signs of K_2 and K_{21}; and the term on $\mathbf{L}^q$ is always nsd. These psd parts can be used to construct descent directions. Two important existing cases are s-SNE ($K(t) = e^{-t}$, $K_1 = -1$, $K_2 = 1$, $K_{21} = 0$) and t-SNE ($K(t) = \frac{1}{1+t}$, $K_1 = -K$, $K_2 = 2K^2$, $K_{21} = K^2$). For EE (an unnormalized model with $K(t) = e^{-t}$), we follow an analogous but simpler process: the Hessian lacks some of the nsd parts in normalized models, e.g. the $\operatorname{vec}(\cdot)$ term, so it should afford better psd Hessian approximations.

The Spectral Direction (SD) We have found that in most cases a particular partial Hessian strikes the best compromise between deep descent and efficient computation, and yields what we call the *spectral direction* (SD). It is constructed purely from the attractive Hessian $\nabla^2 E^+(\mathbf{X}) = 4\mathbf{L}^+ \otimes \mathbf{I}_d$, which as noted earlier is psd, and consists of d identical diagonal blocks of $N \times N$. For EE and s-SNE this amounts to taking $\lambda = 0$ and so using the Hessian of the spectral method, thus it would achieve quadratic convergence in that case. We find it works surprisingly well for $\lambda > 0$. Effectively, we "bend" the exact gradient of the nonlinear E using the curvature of the spectral E^+.

This basic direction is refined as follows. (1) Owing to the shift invariance of E, the resulting linear system is not pd but psd. To prevent numerical problems we add a small $\mu_k \mathbf{I}$ to it ($\mu_k = 10^{-10} \min(L^+_{nn})$ works well). (2) Instead of $\mathbf{B}_k \mathbf{p}_k = -\mathbf{g}_k$ (which is $\mathcal{O}(N^3 d)$) we solve two triangular systems $\mathbf{R}_k^T(\mathbf{R}_k \mathbf{p}_k) = -\mathbf{g}_k$ (which is $\mathcal{O}(N^2 d)$) where $\mathbf{R}_k$ is the upper triangular Cholesky factor of $\mathbf{B}_k$; it can be computed in place in $\mathcal{O}(\frac{1}{3}N^3)$ with standard linear algebra routines, and is sparse if $\mathbf{B}_k$ is sparse. This is crucial for scalability. For Gaussian kernels (SNE, EE) $\mathbf{L}^+$ is constant and it need only be factorized once in the first iteration. If $\mathbf{L}^+$ depends on $\mathbf{X}$, as in t-SNE, scalability is achieved by taking it constant (e.g. $\mathbf{L}^+$ at $\mathbf{X} = \mathbf{0}$). (3) We allow the user to sparsify $\mathbf{L}^+$ through (say) a κ-nearest-neighbor graph, which is often available as part of the data (the affinities w_{nm} or probabilities p_{nm}). This establishes a family from $\kappa = N$ (no sparsity), which

The functions K that result in the simplest Hessians would have $K_{21} = 0$ or $K_2 = 0$, which imply the Gaussian or Epanechnikov kernels, respectively. The functions K that result in the Hessians having most pd parts would have $K_1 \leq 0$ (always satisfied), $K_{21} \leq 0$ and $K_2 \geq 0$; the Gaussian or Epanechnikov kernels also satisfy these conditions.

yields $\mathbf{B}_k = \mathbf{L}^+$, to $\kappa = 0$ (most sparsity), which yields $\mathbf{B}_k = \operatorname{diag}(\mathbf{L}^+) = \mathbf{D}^+$ (the diagonal fixed-point method of Carreira-Perpiñán, 2010).

We explored further variations in the experiments, such as updating the diagonal of $\mathbf{R}_k$ with the pd diagonal part of the full Hessian, with little improvement. Using the technique of Carreira-Perpiñán (2010) of fixed-point iteration from gradient splits, van der Maaten (2010) derives a nonsparse spectral direction for t-SNE, but he overlooks the fact that the resulting linear system is psd. In order to introduce spectral information during the optimization, Memisevic and Hinton (2005) use a search direction where $\mathbf{B}_k^{-1}$ (rather than $\mathbf{B}_k$) is the Laplacian. This can improve over the gradient but, as one would expect, experimentally it is not competitive with our spectral direction.

From the user point of view this yields a simple recipe that, given the gradient of E, does not need the more complex Hessian of E^-. *The only user parameter is the sparsity level κ (number of neighbors) to tune the speed of convergence; convergence itself is guaranteed for all κ by th. 3.3.1.* κ should be simply tuned to as large as computation will allow, while thresholding otherwise negligible values. The cost of computing the direction is $\mathcal{O}(N^2 d)$, the same order (less if sparse) than computing the gradient or E in the line search, and we find its overhead negligible in practice. This affords directions that descend far deeper than gradient or diagonal-Hessian at the same cost per iteration.

In summary, the spectral direction works as follows. Before starting to iterate, compute the attractive Hessian $\nabla^2 E^+(\mathbf{X}) = 4\mathbf{L}^+ \otimes \mathbf{I}_d$, sparsified to κ nearest neighbors, add the small $\mu\mathbf{I}$ to it, and cache its sparse Cholesky factor $\mathbf{R}$. At iteration k, given the gradient $\mathbf{g}_k$, do two backsolves $\mathbf{R}^T(\mathbf{R}\mathbf{p}_k) = -\mathbf{g}_k$ to obtain the spectral direction $\mathbf{p}_k$.

Convergence The following theorem guarantees global convergence (to a stationary point from any initial $\mathbf{x}_0$). It can be derived from Zoutendijk's condition and exercise 3.5 in Nocedal and Wright (2006, p. 39,63) and is proved in Vladymyrov and Carreira-Perpiñán (2012).

Theorem 3.3.1. *Consider the iteration $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ where $\mathbf{p}_k = -\mathbf{B}_k^{-1}\mathbf{g}_k$, $\mathbf{g}_k$ is the gradient, $\mathbf{B}_k$ is symmetric pd and α_k satisfies the Wolfe conditions for $k = 0, 1, 2 \dots$ If E is bounded below in $\mathbb{R}^{Nd}$ and continuously differentiable in an open set $\mathcal{N}$ containing the level set of $\mathbf{x}_0$, ∇E is Lipschitz continuous in $\mathcal{N}$, and the condition number of $\mathbf{B}_k$ is bounded in $\mathcal{N}$, then $\|\nabla E(\mathbf{x}_k)\| \to 0$ as $k \to \infty$.*

In our case, we can ensure the condition number is bounded by simply adding $\mu_k \mathbf{I}$ to $\mathbf{B}_k$

with $\mu_k \geq \mu > 0$ (since $\nabla^2 E$ is bounded), which we do in practice anyway since some of our $\mathbf{B}_k$ are psd. The other conditions hold for the E functions we use. From eq. (10.30) in Nocedal and Wright (2006) and with bounded condition number, it follows that

$$\|\mathbf{x}_k + \mathbf{p}_k - \mathbf{x}^*\| \lesssim r \|\mathbf{x}_k - \mathbf{x}^*\| + \mathcal{O}(\|\mathbf{x}_k - \mathbf{x}^*\|^2)$$

where $\mathbf{x}^*$ is a minimizer of E, $\mathbf{H}(\mathbf{x}^*)$ and $\mathbf{B}(\mathbf{x}^*)$ its Hessian and matrix $\mathbf{B}$, and $r = \left\|\mathbf{B}^{-1}(\mathbf{x}^*)\mathbf{H}(\mathbf{x}^*) - \mathbf{I}\right\|$. Thus the iterations have locally linear convergence with rate r if we use unit step sizes (which we see in practice). The better the Hessian approximation $\mathbf{B}$ the smaller r and the faster the convergence. This is quantified in the experiments.

Other Partial-Hessians $\mathbf{B}_k$ These typically need to solve a nontrivial linear system $\mathbf{B}_k\mathbf{p}_k = -\mathbf{g}_k$. This can be accelerated in several ways: (1) by solving the system in an inexact way using linear CG initialized at the previous iteration's solution, exiting the solver after a certain tolerance $\epsilon > 0$ is achieved. (2) By updating $\mathbf{B}_k$ and its Cholesky factor every $T \geq 1$ iterations. The user has control on the exactness of the solution through ϵ or T. The gradient is always updated at each iteration. For the experiments in this chapter we will focus on strategies with $T = 1$.

3.4 Experimental Evaluation

We have explored a number of partial Hessians as well as different strategies for efficient linear system solution, in datasets with s-SNE, t-SNE and EE. Here we report a representative subset of results, including what we consider the overall winner (the spectral direction). We compare the following methods: gradient descent (GD), used in SNE (Hinton and Roweis, 2003) and t-SNE (van der Maaten and Hinton, 2008); fixed-point diagonal iteration (FP), used in EE (Carreira-Perpiñán, 2010), much faster than GD; the diagonal of the full Hessian (DiagH); nonlinear conjugate gradients (CG), used in NeRV (Venna et al., 2010) and L-BFGS (typical choice for large problems); spectral direction (SD), possibly sparsified and caching the Cholesky factor before the first iteration; and a partial Hessian $4\mathbf{L}^+ + 8\lambda\mathbf{L}^{xx}_{i*,i*}$ (which we call SD–). The latter consists of positive block-diagonal elements of $8\lambda\mathbf{L}^{xx}$ corresponding to entries associated with the same dimension ($i = j$ in $w^{xx}_{in,jm}$). This ensures a psd approximation and adds information about the Hessian of the repulsive term $E^-(\mathbf{X})$. Except for GD, FP and CG, all the other methods have not been applied to SNE-type methods that we know. Several of these methods

require the user to set parameter values. For L-BFGS we tried several values for its user parameter m (the number of vector pairs to store in memory) and found $m = 100$ best. For SD–, we solve the linear system with linear CG, exiting early when the relative tolerance ϵ drops below 0.1 or we reach 50 linear CG iterations. Generally, these parameters are hard to tune and there is little guidance on which values are the best. This is an important reason why the spectral direction, which requires no parameters to tune and performs very well, is our preferred method.

We also tried other methods that were not generally competitive. For example, adding to the SD Hessian the diagonal of the full Hessian (which depends on $\mathbf{X}$ and so varies over iterations), and solving the linear system by approximately updating the Cholesky factorization or by using CG.

Once the direction is obtained for a given method, we use a backtracking line search (Nocedal and Wright, 2006) to find a step size satisfying the first Wolfe condition (sufficient decrease). As initial step size we always try the natural step $\alpha = 1$ (recommended for quasi-Newton updates). However, we observed that some methods (in particular SD) tend to settle to accepted step sizes that are somewhat less than 1. For such cases we used an adaptive strategy: the initial backtracking step at iteration k equals the accepted step from the previous iteration, $k - 1$. This is a conservative strategy because once the step decreases it cannot increase again, but it compensates in saving line searches with require expensive evaluations of the error E. For nonlinear CG, we use Carl Rasmussen's implementation `minimize.m`, which uses a line search that is more sophisticated than backtracking, and allows steps longer than 1.

We evaluated these methods in a small dataset in three conditions (converging to the same minimum, converging to different minima, and homotopy training), and in a large dataset. For EE we used $\lambda = 100$.

3.4.1 Small dataset: COIL-20 image sequences

The COIL-20 dataset contains rotation sequences of objects every 5 degrees, so each data point is a grayscale image of 128×128 pixels. We selected sequences of ten objects for a total of $N = 720$ points in $D = 16\,384$ dimensions, corresponding to ten loops (1D closed manifolds) in $\mathbb{R}^D$. In all the experiments we used the entropic affinities with

available e.g. at `http://http://learning.eng.cam.ac.uk/carl/code/minimize/minimize.m`

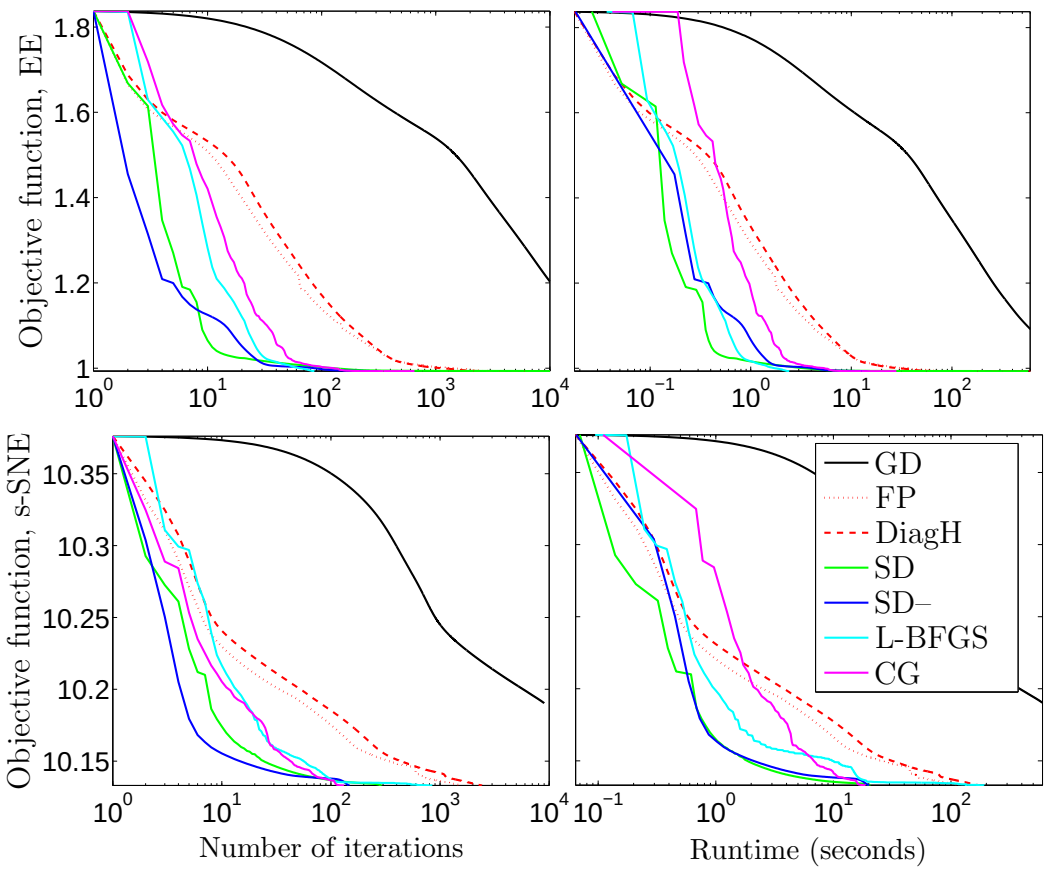

Figure 3.1: COIL-20 dataset, optimization with fixed initial and final points, for EE (top) and s-SNE (bottom). Learning curves as a function of the number of iterations (left) and runtime (right).

perplexity $K = 20$ and reduced dimension to $d = 2$, so visual inspection could be used to assess the quality of the result. For SD we used no sparsification ($\kappa = N$).

Convergence to the same minimum from the same initial X We determined embeddings $\mathbf{X}_0$ and $\mathbf{X}_\infty$ such that $\mathbf{X}_\infty$ is a minimum of $E(\mathbf{X})$ and $\mathbf{X}_0$ is close enough to $\mathbf{X}_\infty$ that all methods converged to $\mathbf{X}_\infty$ when initialized from $\mathbf{X}_0$. Thus, all methods have the same initial and final destination. This allows us to reduce effects due to different local minima of the error E having possibly different characteristics. Fig. 3.1 shows learning curves for EE and s-SNE as a function of the number of iterations and the runtime. In decreasing runtime, the methods can be roughly ordered as GD $\gg$ (FP,DiagH) $>$ (CG,SD–) $>$ (L-BFGS,SD), with GD being over an order of magnitude slower than FP, and FP about an order of magnitude slower than SD (note the log X

axis). The runtime behavior and the number of iterations required agrees with the intuition that the more Hessian information the better the direction, as long as the iterations are not too expensive. Note how the method using most Hessian information, SD–, uses the fewest iterations (left panels), but these are also the slowest, which shifts its runtime curves right. For all the other methods, computing the direction costs less than computing the gradient itself. FP is very similar to DiagH. GD was the only method that did not reach the convergence value even after 10 000 iterations (20 minutes runtime).

L-BFGS is a leading method for large-scale problems. It estimates inverse Hessian information through rank-2 updates, which gives better directions than the gradient, and obtains the direction from a series of outer products rather than solving a linear system, which is fast. The main problems of L-BFGS (Nocedal and Wright, 2006, p. 180,189) are that it converges slowly on ill-conditioned problems and that, with large Nd, it requires an initial period of many iterations before its Hessian approximation is good. While for the small problem of fig. 3.1 L-BFGS is almost competitive with the SD, in the larger problem of fig. 3.4 it is not: 70 iterations (for EE) give a rank-140 approximation to a $40\text{k} \times 40\text{k}$ Hessian matrix, which fails to decrease the error.

Nonlinear CG is generally inferior to L-BFGS and this is seen in the figure too. (Our results unfairly favor CG because its `minimize.m` implementation uses a better line search than in our implementation of the other methods.)

From the beginning, the SD has an exact part of the Hessian that is pd, and obtains the direction from triangular backsolves (same cost as matrix-vector product, and dominated by the cost of computing the gradient). The only overhead is in the initial Cholesky decomposition of $\mathbf{L}^+$ (done only once, since $\mathbf{L}^+$ is a constant for a Gaussian kernel and is set to constant by using $\mathbf{X}_0$ for other kernels), which is small, and progress thereafter is consistently fast.

Convergence from random initial X to possibly different minima We generated 50 random points $\mathbf{X}_0$ (with small values) and ran each method initialized from each $\mathbf{X}_0$, stopping after 20 seconds runtime. Fig. 3.2 shows the error E and number of iterations for each initialization, for EE and s-SNE. They confirm the previous observations, in particular SD and L-BFGS achieve the lower errors, but SD does so more reliably (less vertical spread). GD (outside the plot) barely moved from the initial $\mathbf{X}_0$.

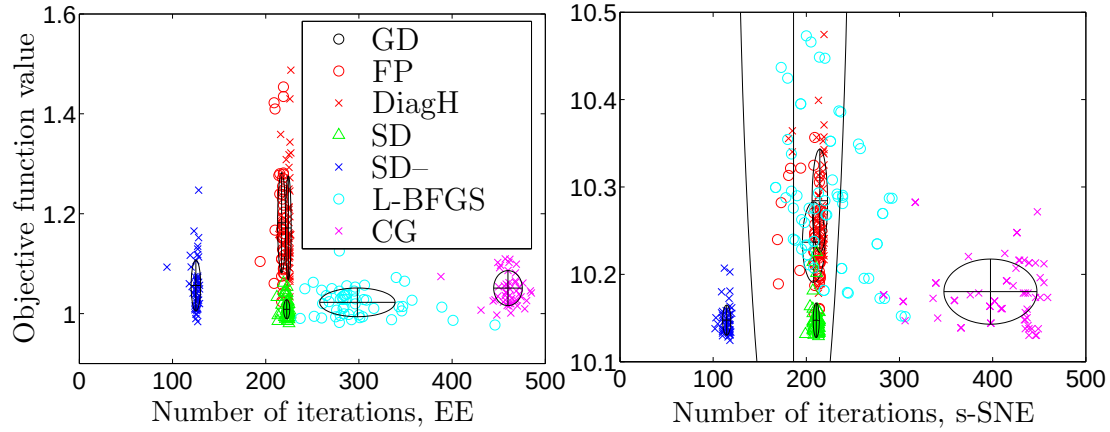

Figure 3.2: COIL-20 dataset, optimization ran for 20 s from 50 random initializations, for EE (left) and s-SNE (right).

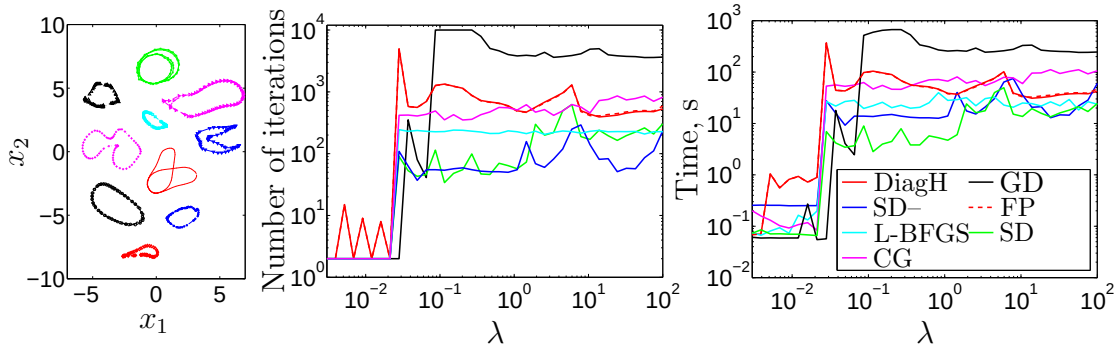

Figure 3.3: COIL-20. Homotopy optimization of EE. *Left*: final convergence point for nearly all methods. *Right two plots*: number of iterations and runtime required to achieve the target tolerance for each λ.

Homotopy optimization for EE The EE error function $E(\mathbf{X}; \lambda)$ can be optimized by homotopy, by starting to minimize over $\mathbf{X}$ at $\lambda \approx 0$, where E is convex, and following a path of minima to a desired λ by minimizing over $\mathbf{X}$ as λ increases (Carreira-Perpiñán, 2010). This is slower than directly minimizing at the desired λ from a random initial $\mathbf{X}$, but usually finds a deeper minimum. We used 50 log-spaced values of λ from 10^{-4} to 10^2 and minimized E at each λ value until the relative error decrease was less than 10^{-6} or we reached 10^4 iterations. Tracking the path $\mathbf{X}(\lambda)$ so closely, we were able to have all methods converge to essentially the same embedding (shown in fig. 3.3), except for GD, whose embedding was still evolving (it would have required many more iterations

Table 3.1: Total number of error function evaluations and runtime for homotopy optimization of EE for COIL-20 dataset.

Method	GD	FP	DiagH	SD	SD–	L-BFGS	CG
E evals	143 237	26 219	26 235	5 183	**2 775**	6 816	16 600
Time, s	9 291	2 015	2 016	**402**	703	756	2 154

to converge). Fig. 3.3 shows the runtime and the number of iterations for each λ value and Table 3.1 shows the total number of the function evaluations and the total runtime for the whole homotopy optimziation.

The results again demonstrate a drastic improvement of our SD over existing methods (GD and FP from Carreira-Perpiñán, 2010), and confirm that more Hessian information results in fewer iterations and function evaluations required. Also, we observe that the SD step sizes decrease from 1 for $\lambda < 0.02$ to 0.1 for the final λ (even though we reset to 1 the initial backtracking step every time we increase λ). Presumably, as λ increases, so does the effect of the term $E^-(\mathbf{X})$, which the SD Hessian ignores.

3.4.2 Large dataset: MNIST handwritten digit images

While more Hessian information enables deeper decreases of the error per iteration, this comes at the price of solving a more complex linear system. To see how the different optimization methods scale up, we tested them on a dataset considerably larger than those in the literature ($N = 6\,000$ points in van der Maaten and Hinton, 2008). We used $N = 20\,000$ MNIST images of handwritten digits (each a 28×28 pixel grayscale image, i.e., of dimension $D = 784$). We used entropic affinities with perplexity $K = 50$ and reduced dimension to $d = 2$. All our experiments were run in a 1.87 GHz workstation, without GPUs or parallel processing. We ran several optimization methods (GD, FP, L-BFGS, SD, SD–) for 1 hour each, for both EE and t-SNE. For the SD we used a sparse $\mathbf{L}$ matrix with $\kappa = 7$.

As noted in section 3.3, for EE and s-SNE the Hessian of $E^+(\mathbf{X})$ (i.e., the matrix $\mathbf{L}^+$) is constant, so we cache its Cholesky factor before starting to iterate. For t-SNE, this Hessian depends on $\mathbf{X}$, and recalculating it and solving a linear system (even sparse and using linear CG) at each iteration is too costly. Thus, we fix it to the Hessian at the initial $\mathbf{X}$ and cache its Cholesky factor just as with EE. This still gives descent directions

that work very well.

Fig. 3.4 shows the resulting learning curves for EE and t-SNE as a function of the number of iterations and the runtime. Some methods' deficiencies that were already detectable in the small-scale experiments become exaggerated in the larger scale. The SD– direction, while still able to produce good steps, now takes too much time per iteration (even though it is solved inexactly by CG), and is able to complete only 37 iterations for EE and 13 for t-SNE within the allotted time (1 hour). Note SD– does worse than SD in number of iterations even though it uses more Hessian information; this is likely due to the inexact linear system solution. In general, all methods run more iterations for EE than for s-SNE and t-SNE, indicating EE's simpler error function E is easier to minimize. GD is omitted, because it showed no decrease of the objective function. For both EE and t-SNE we never observe any decrease with L-BFGS within 1 hour, although we have tried various values m for the numbers of recent updates to store in the Hessian approximation ($m = 5, 50, 100$); it does decrease a little after 3 hours. This is due to the long time needed to approximate the enormous Hessian. Nonlinear CG does decrease the objective function for EE, but most of the computational resources are spent on the line search. Thus CG did least number of iterations compared to other methods. Our SD has mostly converged already in 15 minutes. SD has a reasonable setup time of 5 min. in both EE and t-SNE to compute the Cholesky factorization (this time can be controlled with the sparsification κ), and it is amply compensated for by the speed of the sparse backsolves in computing the direction at each iteration (which are essentially for free compared to computing the gradient). SD decreases the objective consistently and efficiently from the first iterations. FP does scale up in terms of cost per iteration, but, as in the small dataset, each step makes considerably less progress than a SD step. In summary, FP, SD– and L-BFGS are clearly not competitive with SD, which is able to scale its computational requirements and still achieve good steps.

Fig. 3.4.2 also shows the resulting embeddings for FP from Carreira-Perpiñán (2010) (itself much better than GD) and SD at an intermediate stage (after 20 runtime for EE and 1 hour for t-SNE). The difference is qualitatively obvious. The SD embedding already separates well many of the digits, in particular zeros, ones, sixes and eights. The FP embedding shows no structure whatsoever.

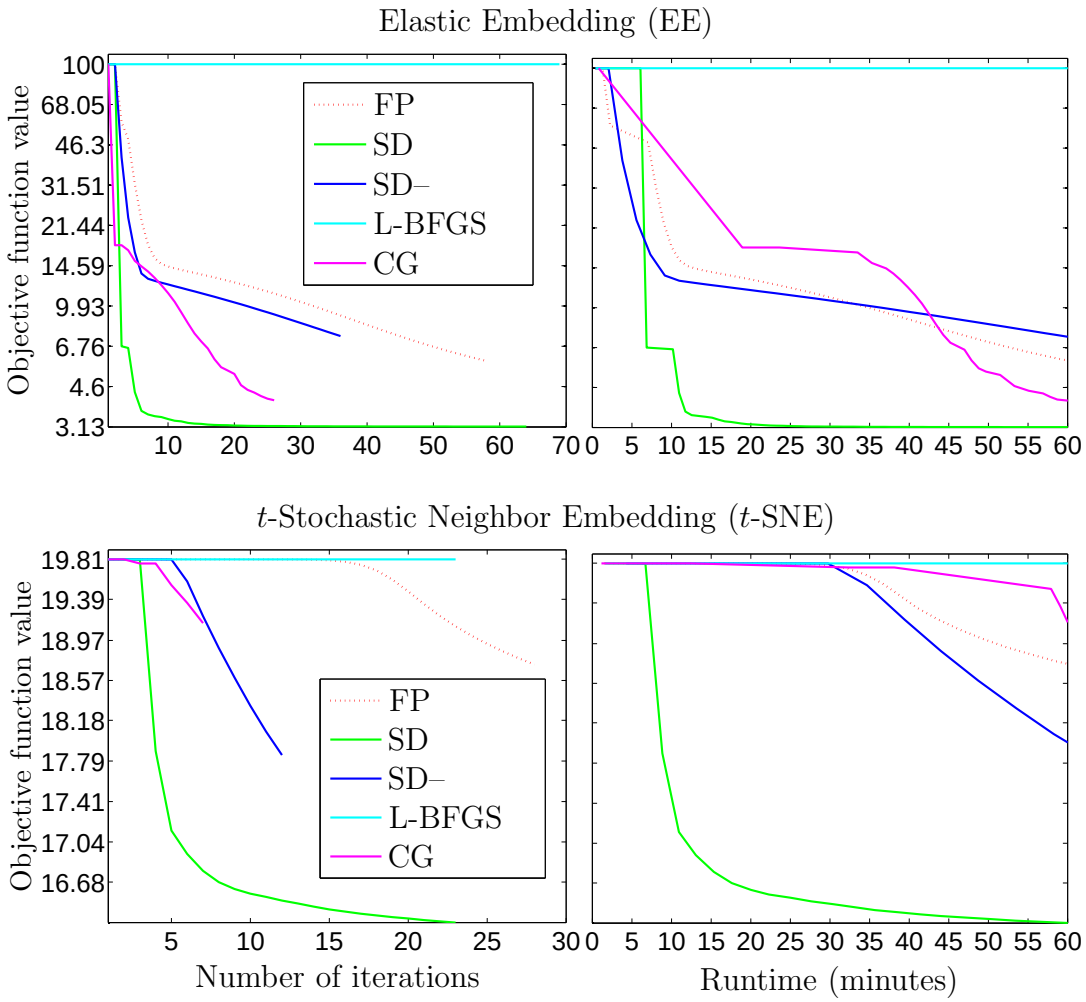

Figure 3.4: Optimization of MNIST dataset, 20 000 points, for EE (left) and t-SNE (right). Learning curves for different methods as a function of the number of iterations (left panels) and runtime (right panels).

3.5 Discussion

Given the exceedingly long runtimes of the gradient descent, we suspect some of the embeddings obtained in the literature of SNE using gradient descent could be actually far from a minimum, thus underestimating the power of SNE. The optimization methods we present, in particular the spectral direction, should improve this situation.

Experimentally, no single method is always the best. If we weigh efficiency, robustness (to user parameters) and simplicity (of implementation using existing linear algebra code and of user parameter setting), we believe that the spectral direction with cached

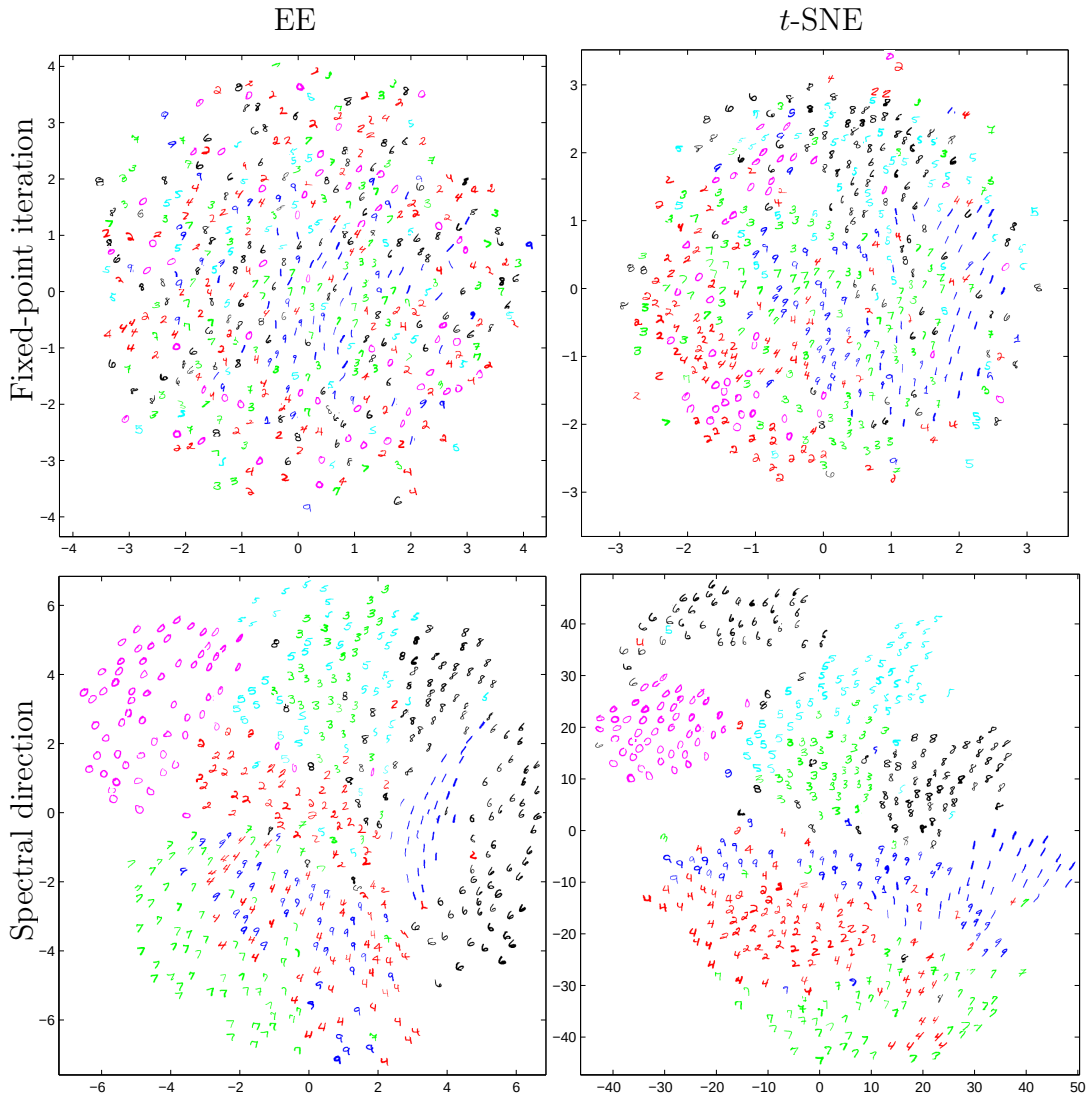

Figure 3.5: Comparison of the embeddings of 20 000 points from MNIST dataset optimized using fixed-point iteration and the spectral direction. *Left:* EE optimization after 20 min., *right:* t-SNE optimization after 1 hour.

Cholesky factor, possibly sparsified, is the preferred strategy. It achieves good steps and can be computed in less time than the gradient and objective function E. However, in really large problems even computing E and ∇E may be too time consuming. Note that, for SNE and t-SNE, even if p_{nm} are sparse in the attractive term, the negative term is still a full $N \times N$ matrix (though the matrix itself need not be stored for E or ∇E to be computed). One solution to this is to use there a sparse graph $\mathbf{W}^-$ as

in EE. However, the quality of the resulting embedding may be affected depending on the sparsity level. (Note this would not affect the construction of our spectral direction, since it does not depend on E^-.) In Chapter 5 we would explore another strategy to accelerate the computations of sums of many Gaussians, needed in E and ∇E, using fast multipole methods (Greengard and Strain, 1991; Raykar and Duraiswami, 2006a), which can reduce the time to $\mathcal{O}(N)$ if the dimension d is low (which is the case e.g. for visualization purposed).

3.6 Conclusion

We have provided a generalized formulation of embeddings resulting from the competition between attraction and repulsion that includes several important existing algorithms and suggests future ones. We have uncovered the relation with spectral methods and the role of graph Laplacians in the gradient and Hessian, and derived several partial-Hessian optimization strategies. A thorough empirical evaluation shows that among several competitive strategies one emerges as particularly simple, generic and scalable, based on the Cholesky factors of the (sparsified) attractive Laplacian. This adds a negligible overhead to the computation of the gradient and objective function but improves existing algorithms by 1–2 orders of magnitude. The quadratic cost of the gradient and objective function remains a bottleneck which we will address in Chapter 5.

Chapter 4

Locally Linear Landmarks

4.1 Introduction

In this chapter we would concentrate specifically on a spectral manifold learning methods (Saul et al., 2006). The input to these algorithms is a symmetric positive (semi)definite matrix $\mathbf{A}_{N\times N}$ (affinity matrix, graph Laplacian, etc.) that contains information about the similarity between pairs of data points $\mathbf{Y} \in \mathbb{R}^{D\times N}$, and a symmetric positive definite matrix $\mathbf{B}_{N\times N}$ that sets the scale of the solution. For example, for Laplacian Eigenmaps (Belkin and Niyogi, 2003), $\mathbf{A}$ is equal to the graph Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$ and $\mathbf{B}$ is the degree matrix $\mathbf{D}$. In case of ISOMAP (Tenenbaum et al., 2000) the similarity matrix is the matrix of geodesic distances between pairs of points in the dataset.
Given these two matrices, the *generalized spectral problem* seeks a solution $\mathbf{X} \in \mathbb{R}^{d\times N}$ to the following optimization problem:

$$\min_{\mathbf{X}} \operatorname{tr}\left(\mathbf{X}\mathbf{A}\mathbf{X}^T\right) \text{ s.t. } \mathbf{X}\mathbf{B}\mathbf{X}^T = \mathbf{I}. \tag{4.1}$$

Within this framework it is possible to represent manifold learning methods such as Principal Component Analysis, Linear discriminant analysis, Laplacian Eigenmaps (LE) (Belkin and Niyogi, 2003), Kernel PCA (Schölkopf et al., 1998), MDS (Cox and Cox, 1994), ISOMAP (Tenenbaum et al., 2000), SDE (Weinberger and Saul, 2004) and LLE (Saul and Roweis, 2003), as well as spectral clustering (Ng et al., 2002).

This chapter is an extended version of Vladymyrov and Carreira-Perpiñán (2013b).

The solution of the spectral problem (4.1) is given by $\mathbf{X} = \mathbf{U}_d^T \mathbf{B}^{-\frac{1}{2}}$, where $\mathbf{U}_d = (\mathbf{u}_1, \ldots, \mathbf{u}_d)$ are d trailing eigenvectors of the matrix $\mathbf{C} = \mathbf{B}^{-\frac{1}{2}} \mathbf{A} \mathbf{B}^{-\frac{1}{2}}$. In large problems (large N), the computational cost means the matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ have to be sparse, and these eigenvectors are found with numerical linear algebra techniques such as restarted Arnoldi iterations (Lehoucq and Sorensen, 1996). However, the resulting cost is still large when N and d are large. The primary goal of this chapter is to find fast, approximate solutions to the spectral problem (4.1) (and thus the solution of LE, spectral clustering, etc.). We propose a method we call *Locally Linear Landmarks (LLL)*, based on the idea of selecting a subset of $L \ll N$ landmarks $\widetilde{\mathbf{Y}}_{L \times N}$ from the data, approximating the data manifold by a globally nonlinear but locally linear manifold around these landmarks, and then constraining the solution $\mathbf{X}$ to follow this locally linear structure. The locally linear mapping is given by a projection matrix $\mathbf{Z} \in \mathbb{R}^{L \times N}$ that satisfies

$$\mathbf{Y} \approx \widetilde{\mathbf{Y}} \mathbf{Z} \tag{4.2}$$

in the high-dimensional space (the approximation error will be defined as a minimization problem 4.6), and by enforcing it in the low-dimensional space, we can re-express the problem (4.1) as a new spectral problem on a smaller number of variables L. This reduces the cost of the eigendecomposition dramatically and, as we will show, constructs affinity matrices that preserve much manifold information because the problem still involves the entire dataset. Note that LLL is not a new manifold learning method, but a fast, approximate way to solve an existing methods of the form (4.1).

The LLL algorithm can be used for purposes beyond fast solutions of spectral problems. First, it is particularly useful for model selection. The similarity matrices $\mathbf{A}$ and $\mathbf{B}$ are usually constructed using some meta-parameters, such as a bandwidth σ of Gaussian affinities and a sparsity level K_W (number of neighbors). In practice, a user has to tune these parameters to the dataset by hand by solving the spectral problem for each parameter value (for example, in spectral clustering (Ng et al., 2002) one can choose the bandwidth σ that gives the least distorted clusters). This is extremely costly with large datasets. As we will show, we can run LLL with very few landmarks so that the *shape* of the model selection curve (especially its minimum) is preserved well. This way we can identify the optimal meta-parameters much faster and then solve the spectral problem (possibly using more landmarks). Second, LLL solves the out-of-sample problem for the spectral problem (4.1) (which projects only the training points) by providing a natural, explicit mapping to project new points, which does not exist in the original

spectral problem. Finally, we observe that the gain of LLL is much bigger when the number of eigenvectors d is large that makes it very attractive as a preprocessing step for classification to other machine learning tasks (e.g. see fig. 4.8 where we used LLL for Laplacian Eigenmaps as a preprocessing before 1-nearest neighbor classification).
Apart from the use in dimensionality reduction, we will also show the application of the method to a problems where spectral clustering can be used, including clustering of handwritten digits dataset as well as image and motion segmentation. It gives much faster yet still accurate solution with respect to the exact spectral clustering and compares favorably to the Nyström extension. For the large-scale experiment we applied the algorithm for a motion segmentation problem using spatio-temporal affinities with $N = 787\,200$ variables. It took the algorithm just under 10 minutes to get meaningful solution, while for Nyström we were not able to get good results at all, because the subset needed to approximate the problem is too big.

4.2 Related work

The most widespread method to find an approximate, fast solution of the spectral problem is the Nyström method (Williams and Seeger, 2001; Bengio et al., 2004b; Drineas and Mahoney, 2005; Talwalkar et al., 2008). It approximates the eigendecomposition of a large positive semidefinite matrix using the eigendecomposition of a much smaller matrix of landmarks. It can be seen as an out-of-sample extension where we first solve for the landmarks separately from the non-landmark points, and then use it to project the rest of the points. Since, during the projection of the landmarks, the Nyström method does not use the data from the non-landmark points, which is available from the beginning, it can result in large approximation errors if the number of landmarks is low.
It is possible to redefine the affinities between landmarks so that they use information from all points, for example by using a commute distance (the expected time it takes a random walk to travel from the first to the second node and back). For example, van der Maaten and Hinton (2008) use the affinity graph between all the points in the data set and define random walk on the graph that starts at the landmark point and finishes at one of the other landmark locations. Then the similarity between two given landmarks is proportional to the probability that random walk started in the first landmark will finish in the second. However, besides the fact that this solves a different spectral problem,

computing these distances is costly, it provides no out-of-sample mapping, and commute distances have been shown to be problematic with large datasets in high dimensions (von Luxburg et al., 2010). As we will show, in LLL the affinities between landmarks use naturally the information in non-landmarks without us having to define new affinities.

Other landmark-based methods can be seen as forms of a Nyström approach. De Silva and Tenenbaum (de Silva and Tenenbaum, 2004) suggested to run the metric MDS algorithm on a subset of the data, while the rest of the points can be located through a distance-based triangulation process. The same idea can be applied to a graph of geodesic distances (instead of Euclidean ones) which leads to the Landmark Isomap algorithm (de Silva and Tenenbaum, 2003). This algorithm is able to give better results due to its ability to deal with nonlinear manifolds. It has been showed (Bengio et al., 2004b; Platt, 2005) that those approaches are no more than a Nyström approximation combined with classical MDS or Isomap.

The idea of representing points by linear coding as in eq. (4.2) has been used in many different domains of machine learning, such as image classification (Gao et al., 2010; Wang et al., 2010), manifold learning (Roweis and Saul, 2000; Weinberger et al., 2005; Yu et al., 2009), supervised (Ladický and Torr, 2011) and semi-superwised (Liu et al., 2010) learning. In addition to linearity, many of above algorithms try to find local, sparse representations of the data, so that points are reconstructed using only nearby landmarks. An early work is the LLE method for manifold learning (Roweis and Saul, 2000), which computes the matrix $\mathbf{Z}$ that best reconstructs each data point from a set of nearby points. Variations exist, such as using multiple local weight vectors in constructing $\mathbf{Z}$ in the MLLE algorithm (Zhang and Wang, 2007). Weinberger et al. (2005) also used representation of the points using a small landmark subset in order to approximate the Maximum Variance Unfolding algorithm. However, these works use local linear mappings to define a new spectral problem, while LLL uses them to approximate an existing one. Zhou et al. (2009); van Gemert et al. (2008); Liu et al. (2011) evaluate linear approximation matrix $\mathbf{Z}$ using a sort of kernel regression. However, as pointed out by Liu et al. (2010), kernel-defined weights are sensitive to the parameters, such as kernel bandwidth, and also lack meaningful interpretation.

The AnchorGraph algorithm (Liu et al., 2010) uses the local coding in the graph Laplacian regularization term of a semi-supervised learning problem and also in the construction of the affinities. The problem that they solve is very different from (4.1). Chen and

Cai (2011) uses the AnchorGraph idea to speed-up the spectral clustering technique in a similar way it is done in LLL. However, they do not show how the rest of the points (non-landmarks) are clustered. Apart from that, the last two approaches do not generalize beyond Laplacian regularizer and the spectral clustering respectively compared to the more general approach that we propose here. Landmark SDE (Weinberger et al., 2005) proposes to reconstruct kernel matrix using much smaller matrix of inner products between the landmarks only. This problem is also different to ours.

Two approaches exist to construct out-of-sample mappings for spectral problems such as Laplacian eigenmaps: Bengio et al. (2004a) apply the Nyström formula using the affinity kernel that defined the problem. Carreira-Perpiñán and Lu (2007) augment the spectral problem with the test point and solve it subject to not changing the points already embedded. The resulting formula turns out to take the form of a Nadaraya-Watson estimator jointly constructed on the feature vectors and embedding projections. For both of these formulas the approximation error decreases as the number of landmarks L increases and becomes zero when $L = N$. In LLL, the out-of-sample mapping is a natural subproduct of assuming each low-dimensional point to be a local linear mapping of the landmark projections associated with it.

Random projection algorithm (Halko et al., 2011; Boutsidis et al., 2011) constructs an approximation by projecting the data onto a low-dimensional random subspace, computing the eigendecomposition in that space and then re-projecting them back to the original space.

Fergus et al. (2009) proposes a method that is able to calculate clusters for semi-supervised learning with complexity that is independent from the number of points. The method estimates density around the manifold by generalizing the eigenvectors to the eigenfunction on the whole space, thus, creating the density estimation.

4.3 Solving Spectral Problems with Locally Linear Landmarks

The fundamental assumption in LLL is that the local dependence of points on landmarks that occurs in high-dimensional space, eq. (4.2), is preserved in the low-dimensional space:

$$\mathbf{X} \approx \widetilde{\mathbf{X}}\mathbf{Z}. \tag{4.3}$$

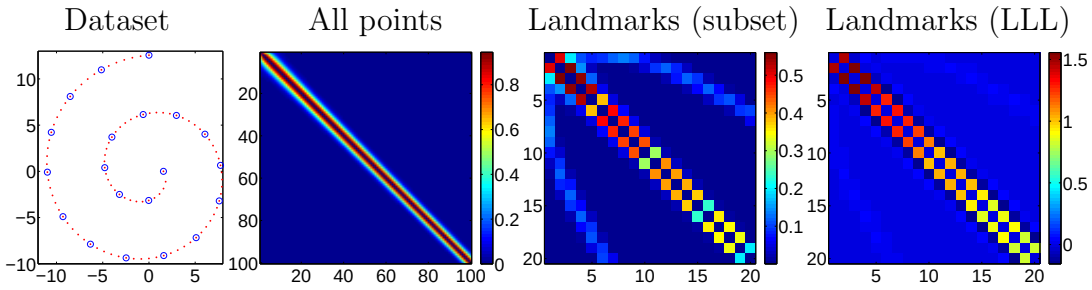

Figure 4.1: Affinity matrices for landmarks from the spiral dataset. *From left to right*: 100 points along the spiral (in red) with 20 landmarks selected uniformly (in blue); the affinity matrix $\mathbf{W}$ used by LE constructed using all the points; the affinity matrix $\mathbf{W}$ built using just landmarks; learned affinity matrix $\mathbf{C}$ of LLL using the whole dataset.

This assumption is related to the manifold assumption that the data lie approximately on low-dimensional manifold that spectral methods are seeking.

Substituting this into the spectral problem (4.1) gives the following reduced spectral problem (on dL parameters):

$$\min_{\widetilde{\mathbf{X}}} \operatorname{tr}\left(\widetilde{\mathbf{X}}\widetilde{\mathbf{A}}\widetilde{\mathbf{X}}^T\right) \text{ s.t. } \widetilde{\mathbf{X}}\widetilde{\mathbf{B}}\widetilde{\mathbf{X}}^T = \mathbf{I}, \tag{4.4}$$

where the matrices

$$\widetilde{\mathbf{A}} = \mathbf{Z}\mathbf{A}\mathbf{Z}^T, \qquad \widetilde{\mathbf{B}} = \mathbf{Z}\mathbf{B}\mathbf{Z}^T. \tag{4.5}$$

are of size $L \times L$. The solution for the reduced problem is given by $\widetilde{\mathbf{X}} = \widetilde{\mathbf{U}}_d^T \widetilde{\mathbf{B}}^{-\frac{1}{2}}$, where $\widetilde{\mathbf{U}}_d$ are d trailing eigenvectors of the matrix $\widetilde{\mathbf{C}} = \widetilde{\mathbf{B}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{B}}^{-\frac{1}{2}}$. After the solution for the landmarks is found, the values of $\mathbf{X}$ can be recovered by applying the formula (4.3) once again. Algorithm 3 shows the flow of the method.

We can see the reduced problem (4.4) as a spectral problem for just the landmark points using a similarity matrix $\widetilde{\mathbf{A}}$ that incorporates information from the whole dataset. For example, in the Laplacian Eigenmaps the spectral problem (see Section 4.6.1) $\mathbf{A}$ is given by a matrix $\mathbf{W}$ of affinities (typically Gaussian). Using LLL we can dramatically improve the quality of $\mathbf{W}$ over that of constructing $\mathbf{W}$ using the landmarks only by including additional information from the whole dataset. From (4.5) individual elements of the reduced affinities are $\widetilde{a_{ij}} = \sum_{n,m=1}^{N} z_{in} a_{nm} z_{jm}$. The points i and j in the reduced affinities are connected through a path $i \longleftrightarrow n \longleftrightarrow m \longleftrightarrow j$. Even if the landmarks are far apart, they sill can be connected along the manifold.

Algorithm 3 Locally Linear Landmarks

Given a spectral problem $\min_{\mathbf{X}} \operatorname{tr}\left(\mathbf{X}\mathbf{A}\mathbf{X}^T\right)$ s.t. $\mathbf{X}\mathbf{B}\mathbf{X}^T = \mathbf{I}$ for a dataset $\mathbf{Y} \in \mathbb{R}^{D\times N}$.

Pick L landmarks $\widetilde{\mathbf{Y}} \in \mathbb{R}^{D\times L}$ at random from $\mathbf{Y}$.

Compute reconstruction matrix $\mathbf{Z} \in \mathbb{R}^{L\times N}$ for each point wrt its nearest $\mathbf{K}_Z = d+1$ landmarks.

Compute the reduced affinity matrices $\widetilde{\mathbf{A}} = \mathbf{Z}\mathbf{A}\mathbf{Z}^T$ and $\widetilde{\mathbf{B}} = \mathbf{Z}\mathbf{B}\mathbf{Z}^T$.

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

Predict non-landmarks with out-of-sample mapping $\mathbf{X} = \widetilde{\mathbf{X}}\mathbf{Z}$.

Fig. 4.1 shows the affinity matrix constructed in the usual way for a spiral dataset in the full case (using all 100 points) and using 20 landmark points versus the affinity matrix learned using LLL. The latter one (right plot) is almost perfectly banded with uniform entries. This means the connectivity pattern proceeds along the spiral rather than across it, which happens when affinities are constructed directly on the landmarks that are quite distant from each other.

Out-of-sample extension. Given a new point $\mathbf{y}_0 \in \mathbb{R}^D$ that is not a part of the original dataset, we find its projection on the low-dimensional space by computing a new projection vector $\mathbf{z}_0$ for that point using K_Z landmarks around $\mathbf{y}_0$. The embedding of $\mathbf{y}_0$ is found from a linear combination of the landmark projections $\mathbf{x}_0 = \widetilde{\mathbf{X}}\mathbf{z}_0$. The cost of the out-of-sample is $\mathcal{O}(DK_Z^2 + Ld)$, which is linear for all the parameters except for K_Z, which is usually low.

Construction of the projection matrix Z. Let us define the landmarks as a set $\widetilde{\mathbf{Y}} = (\widetilde{\mathbf{y}}_1, \ldots, \widetilde{\mathbf{y}}_L) \in \mathbb{R}^{D\times L}$ of L points in the same space as the high-dimensional input $\mathbf{Y}$. The particular choice of the landmark location will be discussed later in section 4.4. Now each datapoint $\mathbf{y}_n$ can be expressed as a linear combination of nearby landmark points: $\mathbf{y}_n = \sum_{k=1}^{L} \widetilde{\mathbf{y}}_k z_{nk}$ where $\mathbf{z}_n$ is a local projection vector for the point $\mathbf{y}_n$. There are multiple ways to make this projection local. One can consider choosing only few landmarks closest to $\mathbf{y}_n$ or ϵ-balls centered around $\mathbf{y}_n$. Moreover, the choice of landmarks can be different for every n. In our experiments, we keep only K_Z landmarks that are closest to $\mathbf{y}_n$ and use the same K_Z for all the points. Therefore, the projection

matrix $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N) \in \mathbb{R}^{L \times N}$ has only K_Z nonzero elements for every column. This matrix intuitively corresponds to the proximity of the points in the dataset to the nearby landmarks and it should be invariant to rotation, rescaling and translation. The invariance to rotation and rescaling is given by the linearity of the reconstructing matrix $\widetilde{\mathbf{Y}}\mathbf{Z}$ with respect to $\widetilde{\mathbf{Y}}$, whereas translation invariance must be enforced by constraining columns of $\mathbf{Z}$ to sum to one. This leads to the following optimization problem:

$$\min_{\mathbf{Z}} \|\mathbf{Y} - \widetilde{\mathbf{Y}}\mathbf{Z}\|^2, \text{ s.t. } \mathbf{1}^T\mathbf{Z} = \mathbf{1}^T. \tag{4.6}$$

Following the approach proposed in LLE algorithm we introduce point-wise Gram matrix $\mathbf{G} \in \mathbb{R}^{L \times L}$ with elements

$$\mathbf{G}_{ij} = (\mathbf{y}_n - \widetilde{\mathbf{y}}_i)^T(\mathbf{y}_n - \widetilde{\mathbf{y}}_j) \tag{4.7}$$

for every $n = 1, \ldots, N$. Now, the solution to the problem (4.6) is found by solving a linear system

$$\sum_{k=1}^{L} \mathbf{G}_{jk} z_{nk} = \mathbf{1} \tag{4.8}$$

and then rescaling the weights so they sum to one.

Computational Complexity. Using this algorithm, we can drastically reduce the computational cost of the eigendecomposition in (4.1). However, we also need to perform some extra computations to evaluate $\mathbf{Z}$, compute auxiliary matrices (4.5) and perform the final multiplication (4.3) to recover the full embedding.

The computation of $\mathbf{Z}$ consists of computing point-wise Gram matrix $\mathbf{G}$ and solving the linear system. $\mathbf{G}$ is sparse and has only K_Z nonzero elements in each row, so it takes $\mathcal{O}(NDK_Z^2)$ to compute it. The linear system also should be solved just for K_Z unknowns, so it takes $\mathcal{O}(NK_Z^3)$. Among the two, the computation of $\mathbf{G}$ matrix dominates because as we will show below $K_Z < D$. Notice that this step is independent from the number of landmarks L. The cost of computing $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{B}}$ in case of dense matrices is $\mathcal{O}(K_Z N^2)$. In case of sparse input, that computation is reduced to $\mathcal{O}(K_Z Nc)$, where c is some constant that depends on the sparsity of matrices $\mathbf{A}$ and $\mathbf{B}$ as well as on the particular location of nonzero elements in $\mathbf{Z}$. Computing $\widetilde{\mathbf{C}}$ and performing eigendecomposition both take $\mathcal{O}(L^3)$ and recovering the final embedding takes $\mathcal{O}(NLd)$. Overall, the complexity of LLL is equal to $\mathcal{O}\big(K_Z N^2 + N(Ld + DK_Z^2) + L^3)\big)$ in case of dense input and $\mathcal{O}\big(N(K_Z c + Ld + DK_Z^2) + L^3)\big)$ in case of sparse input, which is asymptotically much faster than the cost of eigendecomposition given that $L \ll N$.

The computational cost of the out-of-sample is $\mathcal{O}(DK_Z^2)$ to find a projection vector $\mathbf{z}_0$ and $\mathcal{O}(Ld)$ for a reprojection in the low-dimensional space. Overall $\mathcal{O}(DK_Z^2 + Ld)$ which is linear for all the parameters, except for K_Z which is usually quite low.

4.4 Choice of Parameters

Number of landmarks L. The users should use as many landmarks as they can afford computationally, because the more landmarks the better the approximation. As L increases, the results look more and more similar to the solution of the original spectral problem, which is recovered when $L = N$.

Number of landmarks K_Z around each point. Each point should be a local linear reconstruction of nearby landmarks. Thus it is important that there are enough landmarks around each point so that its nearest landmarks are chosen along the manifold. These landmarks will have nonzero weights in the reconstruction, thus achieving locally and linearity. Non-local weights do not work unless the manifold is globally linear.
Using weights that are nonzero only for the nearest K_Z landmarks implies that the low-dimensional space is partitioned into regions where $\mathbf{X}$ is piecewise linear as a function of the corresponding subset of landmarks. Therefore, given that K_Z landmarks are in general position they span a linear manifold of dimension $K_Z - 1$. Therefore, we need, from one side, no more than $D+1$ landmarks, since $K_Z = D+1$ of them reconstruct any point in D dimensions perfectly. On the other size, using $K_Z > D+1$ makes the weights non-unique and we need to add regularization term to (4.7) to penalize the weight norm by adding a small positive amount to the diagonal of the linear system. However, the manifold learning assumption implies that the intrinsic dimensionality of the manifold is lower than D. For example, if the manifold is linear with dimension $\hat{d}$ then the number of landmark needed to reconstruct any point is $K_Z = \hat{d}+1$ by the same argument as above. However, if the manifold is nonlinear with local dimension $\hat{d}$, then $K_Z = \hat{d}+1$ landmarks reconstruct the point approximately (near its projection on the tangent plane). Thus, overall the number of landmarks around each point should be between $\hat{d}+1$ (which may have a certain reconstruction error, particularly if the landmarks are not in general position) and $D+1$ (which achieves perfect reconstruction). If the reconstruction is imperfect, we introduce an additional error on the embedding, by implicitly replacing each original data point with its projection on landmarks. Thus, K_Z is a user parameter

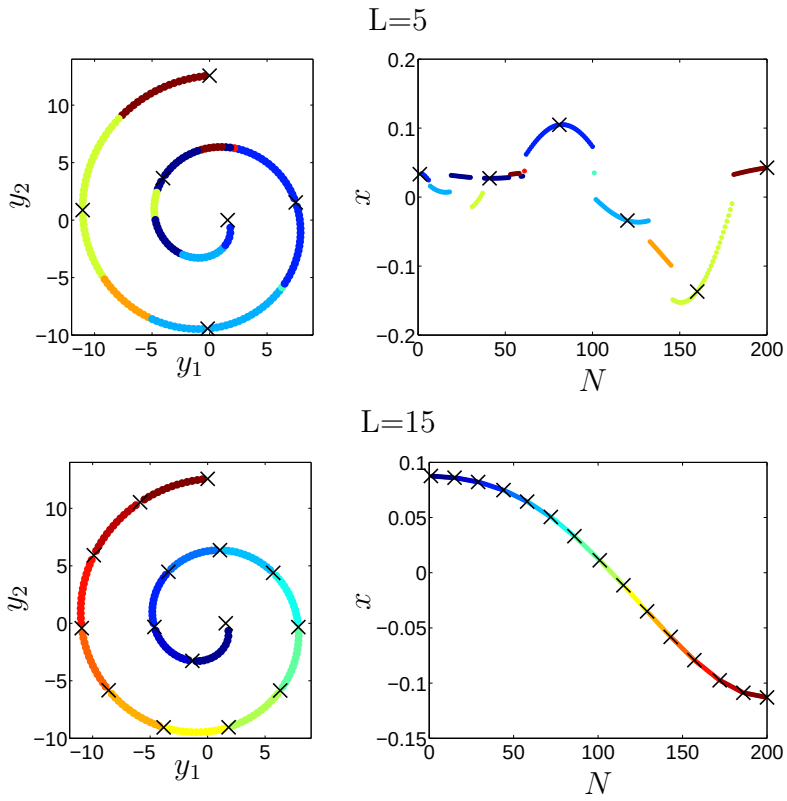

Figure 4.2: The effect of choosing too few landmarks. The color corresponds to two nearest landmarks assignment for each point. We show the original spiral dataset and 1D dimensionality reduction using LLL for LE for $L = 5$ landmarks (*top plot*) and $L = 15$ landmarks (*bottom plot*)

with values in $[\hat{d}+1, D+1]$: the larger K_Z the smaller the error with larger computational cost. In practice, K_Z can be estimated so a desired reconstruction error $||\mathbf{Y} - \widetilde{\mathbf{Y}}\mathbf{Z}||$ is achieved, but it should not be much bigger than $\hat{d} + 1$. Notice that $\hat{d}$ is this context is an intrinsic local dimensionality of the manifold and not the dimensionally of the low-dimensional output d that may or many not match $\hat{d}$.

To illustrate this in fig. 4.2 each of the points along the spiral is reconstructed using two landmarks that are closest to them. In the first two plots we show the results of using $L = 5$ landmarks only (note that coordinate $\mathbf{X}$ is on the vertical axes). As you can see, the number of landmarks is too small to capture the locality of the data, since the points in the branches of a spiral are constructed using landmarks from different branch (see e.g. green, brown and cyan). The resulting one dimensional curve is therefore consists of seemingly independent peaces. In the right two plots we show same dataset, but

this time using $L = 15$ landmarks. In this case there is enough landmarks to capture the locality of the data. Each point is reconstructed with landmarks along the same brunch and the resulted one-dimensional curve is continuos and looks much better than for $L = 5$.

Non-continuity of the result is due to the fact that some chosen landmarks are not located along the manifold, but rather on different "branches" of the spiral.

The location of landmarks. Kumar et al. (2012) provide a formal analysis of different types of sampling and show that, at least for Nyström approximation, uniform sampling works best. Our experiments confirm this as well. However, we cannot spend too much resources on the landmarks selection in order to introduce as little computational overhead as possible. Based on this, we investigated three general methods on how to compute the location of the landmarks.

First, we can always choose the landmarks at random from a set of existing points. This method requires almost no computational resources. However, the result can vary dramatically, especially when only a small number of landmarks is available. We can apply additional heuristic to make the landmark location as close to uniform as possible. First, we select $K+M$ landmarks at random, find M pairs of closest landmarks and then discard one landmark from each pair. This heuristic is useful also because the distances are already given to us from the adjacency matrix. Even in the case when the adjacency matrix is sparse, it is usually the largest distances that are missing. Thus, we can always identify closest landmarks to each other.

Second, we can select the landmarks from running clustering algorithm with L clusters and choose each landmark in the middle of the clusters. For instance, one can run k-means algorithms and set landmarks as the points that are closest to the centroids of the clusters. The problem of this approach is that the clustering is usually quite expensive and requires additional computation resources. Also, using this approach the landmarks can be identified just with centroids that generally do not coincide with the points from the dataset. From one point of view, it can be an advantage because it can potentially take into account outliers and regions with different densities. From another, it can be problematic in case of highly convoluted manifolds, where placing landmark in the middle of two "branches" can cause problems for the algorithm to unfold. In our experiments we avoid dealing with landmarks that are not part of the dataset.

Finally, the landmarks can be also selected using other heuristic so that they span the

manifold as uniformly as possible. It has been proposed (de Silva and Tenenbaum, 2004) to use MinMax algorithm which choses landmarks one by one in a way that maximizes the mutual distance between new landmark and an existing set of landmarks. However, the algorithm requires full graph of the mutual distances between the points available, which in case of large number of points N is unavailable.

4.5 Reusing Z for model and algorithm selection

One of the benefits of LLL is that the reconstruction matrix $\mathbf{Z}$ depends only on the topology of the data and not on the particular algorithm. Thus, constructing $\mathbf{Z}$ is independent from the algorithm we choose to run in the next step. There are two ways we can benefit reusing the same $\mathbf{Z}$ matrix.

Model selection. As a preprocessing step, spectral algorithms needs to have provided with carefully computed affinity matrix. This affinity matrix should be a good representation of the underlying data, which requires careful parameter selection. For example, for the Gaussian affinities, it is crucial to have good bandwidth parameter σ as well as the number of neighbors K_W that one wishes to preserve. There is no universal rule that allows a user to pick the best values for the affinity matrix. Unfortunately, in most cases there is no procedure to check the quality of the affinity matrix without running the algorithm. Ng et al. (2002) simply suggest to chose those parameters by trying several values up until some satisfactory results are achieved. Apart from that, couple of heuristic has been proposed that can improve the results of affinity matrix selection. Zelnik-Manor and Perona (2004) claim that the results can be improved by setting bandwidth individually per each point of the dataset and computing the Gaussian affinity between points $\mathbf{y}_n$ and $\mathbf{y}_m$ as $w_{nm} = \exp(-\frac{\|\mathbf{y}_n - \mathbf{y}_m\|^2}{\sigma_n \sigma_m})$. The paper suggests to use distance to k nearest neighbor to set individual σ_n, for $n = 1, \ldots, N$. Entropic affinities, described in chapter 2 compute individual bandwidth for each point, such that the perplexity, or the effective number of neighbors, is equal to some user-defined parameter K. While each of those methods above can result in a good affinity matrix, they all depend on some parameter(s) that need to be tuned (nearest neighbor, perplexity etc.). This can potentially lead to multiple restarts of the whole algorithms, which is expensive. To reduce this cost, we can leverage the fast approximations of LLL to perform the model selection. This has additional computational advantages, since the LLL can

reuse $\mathbf{Z}$ for all the model selection parameters. Moreover, in many of the algorithms we can additionally precompute one of the reduced affinity matrices $\widetilde{\mathbf{B}} = \mathbf{Z}\mathbf{Z}^T$ that depends only on $\mathbf{Z}$.

Because LLL uses complete affinity matrix, its parameters are also shared between exact method and LLL. Notice that it is not the case for the subset methods (e.g. Nyström), that use smaller affinity matrix which may require different parameter settings for the best performance. The algorithm that we propose is as follows: instead of running full exact spectral clustering several times for different parameter values, we can do this instead for LLL. Then, the best parameter for LLL will roughly correspond to the best parameter for the exact spectral clustering.

Once the optimal parameters are found, we can either stop there with the approximate solution, or run the exact algorithm with the parameters learned from the approximation. In the experimental section we will demonstrate empirically that the optimal parameters of LLL approximation are very close to the optimal parameters of the exact algorithm.

Algorithm selection. There exists many dimensionality reduction methods with different assumptions, optimization criteria, number of parameters, etc. For example, PCA is a linear method but does not require any parameters to tune. Comparing to it, LE requires user to specify additional parameters, such as bandwidth σ and sparsity K_W, however the results are usually better due to the non-linearity. A practitioner unfamiliar with the intricacies of the algorithms often needs to run multiple algorithms to choose a desired performance. In these cases, LLL approximation can be useful, since the same reconstruction matrix $\mathbf{Z}$ can be used for all the algorithms above. Similar to the model selection example above, once the desired algorithm and model parameters are found, one can either finish here with the approximate solution or run the exact algorithm just once with model parameters and the algorithm learned during the LLL step.

4.6 Case studies: LLL for spectral manifold learning

4.6.1 Laplacian Eigenmaps

A particular case of the spectral method for which we can apply LLL is Laplacian Eigenmaps (LE) (Belkin and Niyogi, 2003). The general embedding formulation is recovered using $\mathbf{A}$ as a graph Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$ defined on a symmetric affinity matrix $\mathbf{W}$ with degree matrix $\mathbf{D} = \text{diag}\left(\sum_{m=1}^{N} w_{nm}\right)$ and using $\mathbf{B}$ as that degree matrix $\mathbf{D}$. The objective function is thus

$$\min_{\mathbf{X}} \text{tr}\left(\mathbf{X}\mathbf{L}\mathbf{X}^T\right) \text{ s.t. } \mathbf{X}\mathbf{D}\mathbf{X}^T = \mathbf{I},\ \mathbf{X}\mathbf{D}\mathbf{1} = \mathbf{0}. \tag{4.9}$$

Note that adding the second constraint does not alter the general formulation of the spectral solution, but just remove the first eigenvector, which is constant and equal to $\mathbf{D}^{-\frac{1}{2}}\mathbf{1}$ with eigenvalue 1.

Using (4.5) the coefficients of the model become:

$$\widetilde{\mathbf{A}} = \mathbf{Z}\mathbf{L}\mathbf{Z}^T, \qquad \widetilde{\mathbf{B}} = \mathbf{Z}\mathbf{D}\mathbf{Z}^T. \tag{4.10}$$

Similarly to the case of the original LE, the second constraint is satisfied by discarding the first eigenvector. We can see this by noticing that $\widetilde{\mathbf{A}}\mathbf{1} = \mathbf{0}$ and looking at the eigendecomposition of $\widetilde{\mathbf{C}}$:

$$\widetilde{\mathbf{B}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{B}}^{-\frac{1}{2}}\widetilde{\mathbf{u}}_1 = \widetilde{\mathbf{B}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{x}}^T = \lambda_1\widetilde{\mathbf{u}}_1.$$

Therefore, the solution corresponding to the eigenvalue $\lambda_1 = 0$ is trivial.

4.6.2 Principal Component Analysis

Principal Component Analysis (PCA) is an important tool that has almost ubiquitous use in science. Generally, it has been used for preprocessing of the data. The original version of the algorithm maximizes the following objective function

$$\max_{\mathbf{M}} \text{tr}\left(\mathbf{M}^T\mathbf{C}\mathbf{M}\right), \text{ s.t. } \mathbf{M}^T\mathbf{M} = \mathbf{I} \tag{4.11}$$

where $\mathbf{C} = \overline{\mathbf{Y}}\overline{\mathbf{Y}}^T$ is data-covariance matrix and $\overline{\mathbf{y}}_i = \mathbf{y}_i - \frac{1}{N}\sum_{n=1}^{N}\mathbf{y}_n$ is the dataset $\mathbf{Y}$ shifted to get zero column mean. The solution $\mathbf{M}$ to (4.11) manifests itself also as

linear mapping and is given by an eigendecomposition of $\mathbf{C}$. The final low-dimensional embedding is equal to the projection of the original data on $\mathbf{M}$:

$$\mathbf{X} = \mathbf{M}\mathbf{Y}. \tag{4.12}$$

However, $\mathbf{C}$ is a dense $D \times D$ matrix and for many applications where D is large (e.g. image processing) this matrix is infeasible to store. On the other hand, the problem (4.11) can be reformulated as:

$$\max_{\mathbf{P}} \operatorname{tr}\left(\mathbf{P}\mathbf{K}\mathbf{P}^T\right), \text{ s.t. } \mathbf{P}\mathbf{P}^T = \mathbf{I} \tag{4.13}$$

Now the solution for $\mathbf{P}$ is given by the largest d eigenvectors of $N \times N$ matrix of inner products $\mathbf{K} = \overline{\mathbf{Y}}^T\overline{\mathbf{Y}}$ and the mapping $\mathbf{M}$ can be restored using $\mathbf{M} = \mathbf{\Lambda}^{-1/2}\mathbf{P}\mathbf{Y}^T$, where $\mathbf{\Lambda}$ is a diagonal matrix of d largest eigenvalues of matrix $\mathbf{K}$. Using this formulation instead of (4.11) requires the eigendecomposition of $N \times N$ matrix $\mathbf{K}$, which is more efficient when $N \ll D$ (e.g. when we need to reduce the dimensionality of a series of large images). However, in cases when N and D are large neither procedure is computationally feasible both from the point of storage (covariance and inner product matrices are dense) and runtime (eigendecomposition of dense matrix is $\mathcal{O}(N^3)$).

One way to deal with the complexity problem is to run the algorithm on a subset $\widetilde{\mathbf{Y}}$ and then use a linear mapping $\widetilde{\mathbf{M}}$ learned for a subset: $\mathbf{X} = \widetilde{\mathbf{M}}\mathbf{Y}$. The problem is that the mapping learned will not be representative for the whole dataset, since it will ignore the points that are not part of the subset.

Alternatively, we can exploit the LLL framework to speed up the computation of PCA as well. This may give better results than just using the linear mapping, because the reduced affinities exploit the structure of the whole dataset, rather then using a subset. The reduced affinities (4.5) become: $\widetilde{\mathbf{A}} = \mathbf{Z}\overline{\mathbf{Y}}^T\overline{\mathbf{Y}}\mathbf{Z}^T$, $\widetilde{\mathbf{B}} = \mathbf{Z}\mathbf{Z}^T$. Notice that LLL has one additional advantage: dense $N \times N$ matrix $\mathbf{K}$ does not need to be computed or stored. Instead we can precompute $L \times D$ matrix $\mathbf{Z}\overline{\mathbf{Y}}^T$. Due to the sparsity of $\mathbf{Z}$ the total complexity of computing the affinities is $\mathcal{O}(K_Z N D + L^2 D)$, which is linear for both N and D.

From the memory perspective, the main bottleneck is actually storing the actual data matrix $\mathbf{Y}$ of $D \times N$. However, the only place this matrix enters is in the computation of temporary matrix $\mathbf{Z}\overline{\mathbf{Y}}^T$, which can be done by loading $\mathbf{Y}$ into memory block by block.

4.6.3 Linear discriminant analysis

Linear discriminant analysis (LDA) is a supervised linear dimensionality reduction method that is used as a data preprocessing when the labels are available. The main idea behind the method is to preserve the largest variation of each cluster of the data (so that the clusters retain their structure), but in the way that also minimizes the variance between the classes (such that the clusters separate from each other).

Given a zero-mean dataset $\overline{\mathbf{Y}}$ splited between C classes with N_k points in a class k and $l_i \in (1, \dots, C)$ as a label of a point $\overline{\mathbf{y}}_i$ define between-class affinities $\mathbf{S}_b \in \mathbb{R}^{D \times D}$ and within-class affinities $\mathbf{S}_w \in \mathbb{R}^{D \times D}$ as:

$$\mathbf{S}_b = \sum_{k=1}^{C} N_k \boldsymbol{\mu}_k \boldsymbol{\mu}_k^T, \qquad \mathbf{S}_w = \sum_{k=1}^{C} \left(\sum_{i:l_i=k} (\overline{\mathbf{y}}_i - \boldsymbol{\mu}_k)(\overline{\mathbf{y}}_i^{(k)} - \boldsymbol{\mu}_k)^T \right), \tag{4.14}$$

where $\boldsymbol{\mu}_k$ is the mean of the points in class k.

The objective function of the LDA is to maximize the between-class affinities while minimizing the within-class affinities:

$$\max_{\mathbf{M}} \operatorname{tr} \left(\frac{\mathbf{M}^T \mathbf{S}_b \mathbf{M}}{\mathbf{M}^T \mathbf{S}_w \mathbf{M}} \right), \tag{4.15}$$

The linear mapping $\mathbf{M}$ can be found as leading d eigenvectors to the generalized eigenproblem $\mathbf{S}_b\mathbf{M} = \boldsymbol{\Lambda}\mathbf{S}_w\mathbf{M}$ and the final solution is found using the same projection (4.12) as in PCA.

The solution requires the eigendecomposition of $D \times D$ matrix, which can be expensive to compute in practice. Similar to PCA, the problem can be converted into $N \times N$ eigenproblem. Baudat and Anouar (2000) show that (4.15) can be equivalently expressed as

$$\max_{\mathbf{P}} \operatorname{tr} \left(\frac{\mathbf{P}\mathbf{H}\mathbf{P}^T}{\mathbf{P}\mathbf{T}\mathbf{P}^T} \right), \tag{4.16}$$

where the new affinities are defined as

$$\mathbf{H} = \overline{\mathbf{Y}}^T (\overline{\mathbf{Y}} \mathbf{Q} \overline{\mathbf{Y}}^T) \overline{\mathbf{Y}}, \qquad \mathbf{T} = \overline{\mathbf{Y}}^T (\overline{\mathbf{Y}\mathbf{Y}}^T) \overline{\mathbf{Y}}, \tag{4.17}$$

and $\mathbf{Q}_{ij} = \begin{cases} 1/N_k, & \text{if } l_i = l_j = k \\ 0, & \text{if } l_i \neq l_j \end{cases}$.

Now $\mathbf{P}$ is found as a solution to generalized eigenvalue problem $\mathbf{HP} = \boldsymbol{\Lambda}\mathbf{TP}$. From there, the mapping can be recovered using $\mathbf{M} = \mathbf{P}\mathbf{Y}^T$ and the overall low-dimensional solution is found using a linear projection (4.12).

Using LLL the reduced affinities become $\widetilde{\mathbf{A}} = \mathbf{Z}\overline{\mathbf{Y}}^T(\overline{\mathbf{Y}}\mathbf{Q}\overline{\mathbf{Y}}^T)\overline{\mathbf{Y}}\mathbf{Z}^T$, $\widetilde{\mathbf{B}} = \mathbf{Z}\overline{\mathbf{Y}}^T\overline{\mathbf{Y}}\overline{\mathbf{Y}}^T\overline{\mathbf{Y}}\mathbf{Z}^T$. Again, similar to PCA, we can precompute $\mathbf{Z}\overline{\mathbf{Y}}^T$ and thus avoid storing dense matrices $\mathbf{H}$ and $\mathbf{T}$. Computing the $\widetilde{\mathbf{B}}$ matrix from left to right and avoiding expensive computation of $\overline{\mathbf{Y}}\overline{\mathbf{Y}}^T$ the complexity of computing the affinities is linear in L, N and D: $\mathcal{O}(LND)$.
As an alternative, we can compute a solution for a subset $\widetilde{\mathbf{Y}}$ using exact LDA and then project the rest of the points using a learned mapping as $\mathbf{X} = \widetilde{\mathbf{M}}\mathbf{Y}$. However, the projection computed this way learns the mapping only using the subset of points. The labels of the rest of the points are simply ignored and do not affect the final solution. Nyström method does not help here either for the same reason. LLL, on the contrary, able to benefit from all the labels, while still working on a smaller $L \times L$ affinity matrix.

4.6.4 Kernel PCA and LDA

In kernel PCA (Schölkopf et al., 1998) and kernel LDA (Baudat and Anouar, 2000; Mika et al., 1999), the inner product matrix $\mathbf{K} = \overline{\mathbf{Y}}^T\overline{\mathbf{Y}}$ is replaced by a positive-definite kernel, that, from the Mercer theorem, corresponds to an inner product between feature vectors in some high-dimensional space. The feature vectors and the mapping are not computed explicitly and enter the algorithm only through the inner product matrix $\mathbf{K}$. The method is still linear in the feature space, but due to a nonlinear transformation to that space, it is able to recover more general nonlinear embeddings.
Removing the mean from the data $\mathbf{Y}$ works only with linear kernel. In case of an arbitrary positive definite kernel, Schölkopf et al. (1998) follows a more general procedure of centralizing the matrix $\mathbf{K}$ before the eigendecomposition:

$$\overline{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N\mathbf{K} - \mathbf{K}\mathbf{1}_N + \mathbf{1}_N\mathbf{K}\mathbf{1}_N, \tag{4.18}$$

where $(\mathbf{1}_N)_{ij} = 1/N$.
The application of LLL is straightforward: the reduced affinities have the form $\widetilde{\mathbf{A}} = \mathbf{Z}\overline{\mathbf{K}}\mathbf{Z}^T$, $\widetilde{\mathbf{B}} = \mathbf{Z}\mathbf{Z}^T$ and after solving the small subproblem (4.4) for $\widetilde{\mathbf{X}}$ the solution is recovered using $\mathbf{X} = \widetilde{\mathbf{X}}\mathbf{Z}$. Notice, that, different from PCA, LLL needs to have an $N \times N$ kernel matrix $\mathbf{K}$ as an input. However, for some kernels (e.g. Gaussian) this matrix is sparse and thus can be fit in the memory.

4.7 Case studies: LLL for Spectral Clustering

There exist several formulations of spectral clustering, depending on what affinities are used, whether the graph Laplacian is normalized, etc. Here we stick to the most common one, which approximates the normalized cut criterion (Shi and Malik, 2000). We define the spectral clustering as follows: given a (sparse) affinity matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ defined on a set of data points $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ of $D \times N$, we seek the solution for the following problem:

1. Find spectral embedding using K dimensions:

$$\min_{\mathbf{X}} \operatorname{tr}\left(\mathbf{X}\mathbf{L}\mathbf{X}^T\right), \text{ s.t. } \mathbf{X}\mathbf{D}\mathbf{X}^T = \mathbf{I},\ \mathbf{X}\mathbf{D}\mathbf{1} = \mathbf{0} \tag{4.19}$$

   where $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is a graph Laplacian defined for a degree matrix $\mathbf{D} = \operatorname{diag}\left(\sum_{m=1}^{N} w_{nm}\right)$ and $\mathbf{X} \in \mathbb{R}^{K \times N}$ is a projection of the data into a K dimensional space.

2. Obtain the final clustering by running k-means on $\mathbf{X}$, using the normalized projections $\hat{\mathbf{X}}_{ij} = \mathbf{X}_{ij} / (\sum_j \mathbf{X}_{ij}^2)^{1/2}$.

Similar to the LE, plugging (4.5) to the spectral embedding problem (4.19) gives

$$\min_{\widetilde{\mathbf{X}}} \operatorname{tr}\left(\widetilde{\mathbf{X}}\widetilde{\mathbf{L}}\widetilde{\mathbf{X}}^T\right) \text{ s.t. } \widetilde{\mathbf{X}}\widetilde{\mathbf{D}}\widetilde{\mathbf{X}}^T = \mathbf{I}, \tag{4.20}$$

with $L \times L$ matrices

$$\widetilde{\mathbf{A}} = \mathbf{Z}\mathbf{L}\mathbf{Z}^T, \qquad \widetilde{\mathbf{B}} = \mathbf{Z}\mathbf{D}\mathbf{Z}^T. \tag{4.21}$$

The solution is found by eigendecomposition of a small $L \times L$ matrix $\widetilde{\mathbf{C}} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{L}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$. Thus LLL basically solves a smaller embedding problem for landmarks using small reduced affinities that incorporate the structure of the whole original affinity. After landmarks are projected, the extension to the rest of the points can be found using reconstruction (4.3). Using the landmarks projection we can also accelerate the second step of the algorithm, which is k-means clustering.

4.7.1 Accelerating the k-means clustering step.

After LLL is finished we obtain the K-dimensional projections of the landmarks and the K-dimensional projections $\mathbf{x}_1, \ldots, \mathbf{x}_N$ of the non-landmark points (using (4.3)). We then

While it is possible to find an embedding in a dimension $d \neq K$, Ng et al. (2002) showed that, in the ideal case (i.e. zero values in the affinity matrix for pairs of points from different clusters), K dimensions exactly capture the variation between K clusters in the data.

have to run k-means on $\mathbf{x}_1, \ldots, \mathbf{x}_N$ to obtain the final K clusters. There, the algorithm iteratively reassigns the labels for each point, such that the sum of the distances from the points with the same label to the centroid of that label is minimized. This objective function is non-convex and, to avoid local optima and get better solution, a common strategy is to use multiple restarts with different random initializations (either of points' assignment or of centroids) and pick the result with the lowest error. The cost of this algorithm is $\mathcal{O}(mNK^2)$, where m is the number of repetitions.

We can speed up the k-means step by capitalizing on the fact that the landmarks themselves would provide a reasonable clustering, and there are far fewer landmarks than points. Instead of running k-means for the whole dataset m times, we run it m just on the landmarks, pick the best solution and feed its centroids as an initialization to k-means on the entire data. Thus, k-means is run just once on the full problem, but with an initialization that is much better than random. The complexity of this method is $\mathcal{O}(mLK^2 + NK^2)$. When $L \ll N$ the speed-up over the $\mathcal{O}(mNK^2)$ cost of the naive k-means step is essentially proportional to m. A second benefit is that, since a k-means run on the landmarks takes far less time than a single k-means iteration on the entire data, we can afford to run many restarts and thus increase the likelihood of finding a good local optimum.

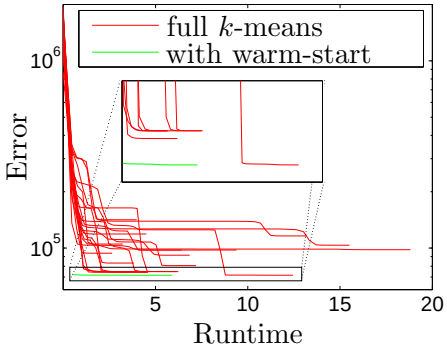

Figure 4.3: 20 runs of standard k-means compared with a warm-started k-means.

This warm-start initialization helps to get good locations of centroids fast, but also quite close to the final solution. Fig. 4.3 shows the decrease of k-means objective function for the clustering $N = 787\,200$ points to 6 classes using LLL (see motion segmentation example in the experimental section below). Red curves correspond to randomly initialized restarts of full k-means and green curve correspond to a single full k-mean initialized from the best of 20 different restarts of k-means of landmarks. Overall it took 0.5 seconds to run k-means on landmarks and 5.3 seconds for a single run of a full k-means. In comparison, 20 restarts of k-means took 91.3 seconds. Notice also, that although the results of both algorithms are the same, on the inset we can see that only one out of 20

runs of k-means converges to that solution. This means that using smaller number of restarts m we could end up with some worse solution.

4.7.2 Algorithm Analysis

Comparing to the motivation of methods that use LLL for manifold learning, the desired properties of spectral clustering in the projection space are different. For clustering, we do not need to preserve the manifold structure, but rather separate the points into clusters. To understand the performance of LLL, let us consider the following ideal clustering situation with spectral clustering. The eigenvectors are piecewise constant with same values for points in each cluster. Indeed, a clustering solution is also a piecewise constant function. The approximate solution (from LLL) for the landmarks' eigenvectors are also piecewise constant, corresponding to the same clusters. The LLL out-of-sample formula (4.3) is a linear combination of the nearest landmarks' eigenvector value. Then we have two cases: (1) if the nearest landmarks to the test point are in the same cluster, their eigenvectors values ("cluster labels") are equal and (4.3) will predict that same eigenvector value, which is correct. This will happen with test points in the "interior" of each cluster, i.e. "nearly everywhere" if the cluster boundaries are narrow. (2) if the nearest landmarks to the test point are not all in the same cluster, their eigenvector values ("cluster labels") are not all equal and the formula will predict an "average" of those values, giving more weight to the landmarks that are closer to the test point. The resulting predicted eigenvector value will be smoothed out, since it averages different labels. This will happen with test points in the "boundary" between different clusters. How narrow are the cluster boundaries depends on the number of landmarks. The more landmarks, the narrower the boundaries, because this is the region where points have nearest neighboring landmarks from different clusters. Thus, the more landmarks, the smaller the smoothing effect on the boundaries.

In fig. 4.4 we show the example of two cases above. The dataset on top plots consist of two clusters along the line. Exact spectral clustering solves this problem given sufficiently narrow bandwidth of the kernel, such that the points on the side of one cluster won't be affected by the points on the side of the other. For LLL, there is one more additional constraint. If the test point is reconstructed by landmarks from different clusters (dark red points in the left plot) the target eigenvector will be smoothed out. Good assignment (as the one on the right), gives much better approximation.

We might expect the LLL assumption (i.e. a piecewise linear model) to match perfectly the piecewise constant clustering model except at discontinuities (cluster boundaries). Note that these points are difficult for spectral clustering in the first place, i.e. the exact eigenvector value at them will be less "pure" than at points in the interior of a cluster.

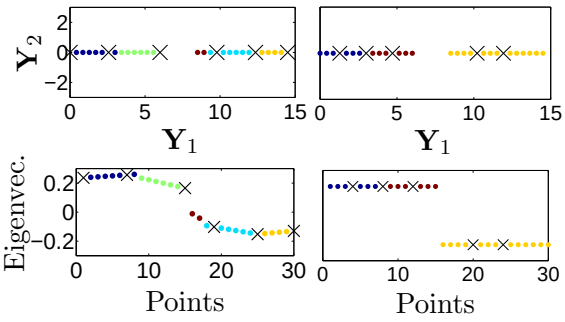

Figure 4.4: *Top:* dataset $\mathbf{Y}$ with two different landmark assignments. *Bottom:* the approximation of the eigenvector by LLL. The color corresponds to two nearest landmark assignment for each point.

LLL might have troubles with small clusters, because they will receive fewer landmarks. However, spectral clustering tends not to produce relatively small clusters, because it approximates the Normalized Cut objective function (Shi and Malik, 2000), which discourages small clusters so this is likely not much of a problem.

4.8 Experimental Evaluation

4.8.1 Laplacian Eigenmaps

We compared LLL for LE to three natural baselines. (1) "Exact LE" runs LE on the full dataset and gives the optimal embedding by definition, but the runtime is large. (2) "LE ($\mathbf{Z}$)" runs LE only on a set of landmark points and then projects non-landmark points using the projection matrix $\mathbf{Z}$, which gives a locally linear (but globally nonlinear) out-of-sample mapping. (3) "LE (Nys.)" runs LE only on a set of landmark points and uses the Nyström out-of-sample formula. The latter two Landmark LE baselines give faster performance, but the embedding quality can be worse because non-landmark points are completely ignored in solving the spectral problem. For all our experiments we used Matlab's `eigs` function to compute the partial eigendecomposition of a sparse matrix.

Role of the number of landmarks. We used 60 000 MNIST digits with sparsity $K_W = 200$ and bandwidth $\sigma = 200$ to build the affinity matrix and reduced the dimensionality to $d = 50$. For LLL, we set $K_Z = 50$ and increased the number of landmarks

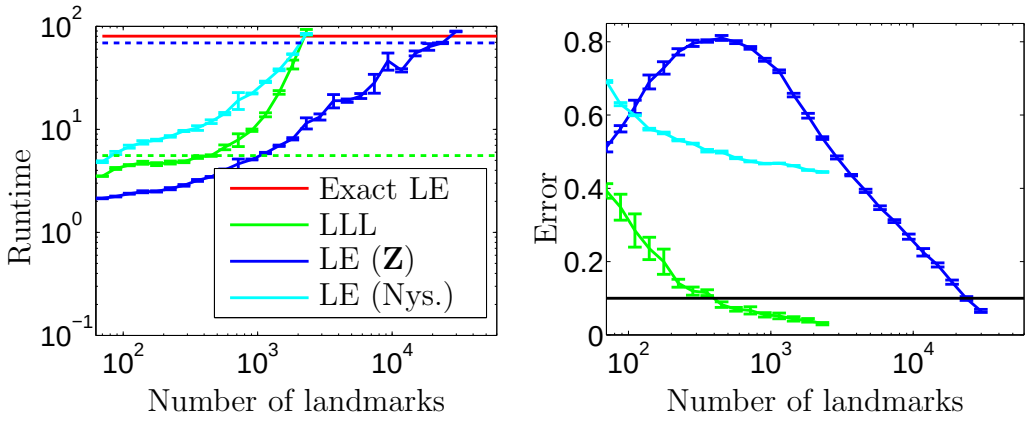

Figure 4.5: The performance of LLL (in green), Landmark LE with **Z** as an out-of-sample (in blue) and Landmark LE with Nyström as an out-of-sample (in cyan). *Left:* runtime as the number of landmarks changes. The green and blue dashed lines correspond to the runtime that gives 10% error with respect to Exact LE for LLL and Landmark LE using **Z** respectively. *Right:* The error with respect to Exact LE. The black line corresponds to 10% error. Note the log scale in most of the axes.

logarithmically from 50 to 60 000. We chose landmarks at random and repeated the experiment 5 times for different random initialization to see the sensibility of the results to the random choice of the landmarks. To quantify the error with respect to Exact LE we first used Procrustes alignment (Cox and Cox, 1994, ch. 5) to align the embeddings of the methods and then computed the relative error between aligned embeddings.

In Fig. 4.5 we show the error as well as the overall runtime for different algorithms as the number of landmarks increases. Our first indicator of performance is to see which algorithm can achieve the error of 10% faster. LLL needed 451 landmarks and 5.5 seconds (shown by a dashed green line in the left plot). This is 14 times faster compared to Exact LE which takes 80 seconds. Landmark LE with **Z** as an out-of-sample achieves the same error with 23 636 landmarks and the runtime of 69 seconds (1.15 speedup, blue dashed line in the right plot). Landmark LE with Nyström is not able to achieve the error smaller than 50% with any number of landmarks. Notice that the deviation from the mean for 5 runs of randomly chosen landmarks is rather small, suggesting that the algorithm is robust to different locations of landmarks.In Fig. 4.6 we show the embedding of Exact LE and the embedding of LLL with 451 randomly selected landmarks. The embedding of LLL is very similar to the one of Exact LE, but the runtime is 15 times faster (5.5

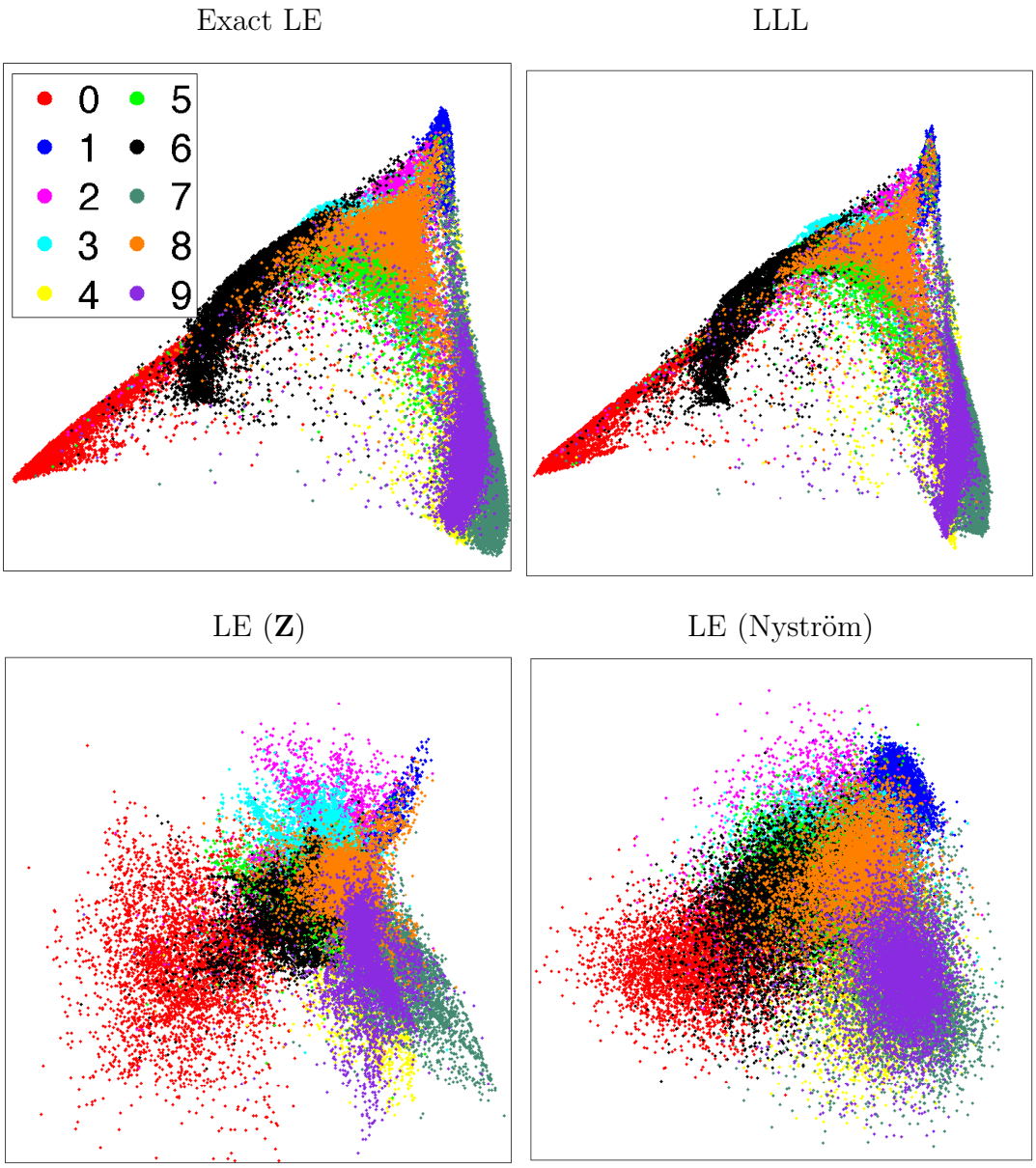

Figure 4.6: The embedding of 60 000 MNIST digits using first two dimensions. *From left to right::* Exact LE (t = 80 s.), LLL (t = 5.5 s., 451 landmarks), Landmark LE with $\mathbf{Z}$ as out-of-sample (t = 5.5 s., 1 144 landmarks), and LE with Nyström as out-of-sample (t = 5.5 s., 88 landmarks).

seconds compared to 80 seconds). Using bigger number of landmarks only decreases the error further and for 3 000 landmarks, where the runtime of LLL matches the runtime of Exact LE the mean error among 5 runs drops to 3%. Landmark LE with $\mathbf{Z}$ as an out-of-sample achieved the same error only by using 23 636 landmarks and the runtime of 69 seconds (1.15 speedup, blue dashed line in the right plot). Landmark LE with Nyström is not able to achieve the error smaller than 50% for any number of landmarks. Notice that the deviation from the mean for 5 runs of randomly chosen landmarks is rather small, suggesting that the algorithm is robust to different locations of landmarks.

Model selection. We evaluated the use of LLL to predict the parameters of the affinity matrix. We used 4 000 points from the swiss roll dataset and run the methods varying different parameters of the algorithm. We run LLL and Landmark LE 5 times using different random initializations to show the general behavior of the algorithm. Experimentally we discovered that the best results are obtained with the bandwidth $\sigma = 1.6$, the number of landmarks L no less than 300 and the sparsity level K_W around 150. We then fixed two out of these three parameters and changed the third one to see how the error curve changes. We show our results in Fig. 4.7. First, for different σ values the error curve of Exact LE is much more similar to the one from LLL, but LLL is able to achieve this results approximately 18 times faster (top plot). Compared to that the Landmark LE definitely needs more landmarks in order to show similar behavior. Second, the number of landmarks needed to achieve the same error as Exact LE is much lower for LLL than for Landmark LE. Using 300 landmarks the error of LLL is about 3% and it is also 18 times faster than Exact LE. Landmark LE never able to achieve 10% error for any set of landmark up to a 1 000. Third, changing the sparsity level parameter K_W the error curve of LLL is again very similar to the one from Exact LE and very different between Exact LE and Landmark LE. The speedup of LLL compared to Exact LE varies between 2 for small values of K_W to 40 for large K_W. Notice that although LLL is not able to match Exact LE for all the possible values of σ and K_W the minimum values of LLL and Exact LE match each other for both σ and K_W. This suggests a simple procedure for a user to set the parameters of Exact LE: use cheaper LLL algorithm to obtain the minimum error for different values of σ and K_W and then initialize Exact LE with those values.

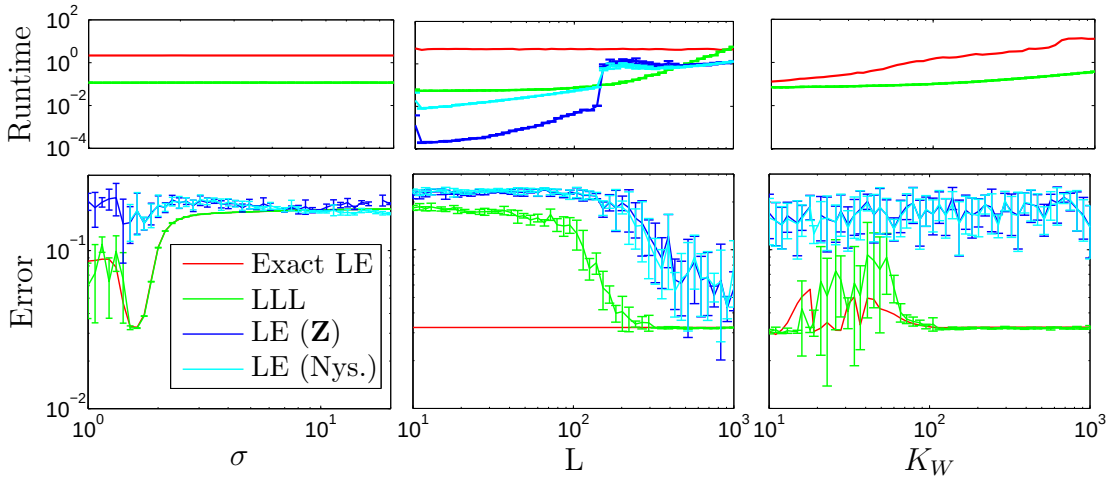

Figure 4.7: The quality of the embedding with respect to the ground truth for different values of bandwidth σ, number of landmarks L, and sparsity level K_W. The dataset is 4 000 points from swiss roll. *Bottom row.* From left to right: vary σ for fixed $L = 300$, $K_W = 150$; vary L for fixed $\sigma = 1.6$, $K_W = 150$; vary K_W for fixed $L = 300$, $\sigma = 1.6$. The error of LLL and Exact LE has similar values for a same parameters set. *Top row:* The runtime for different values of the parameters.

Classification. Here our goal was to find a good set of parameters to achieve low 1-nearest neighbor classification error for full 70 000 MNIST digits dataset. We first split the dataset into three independent sets: 50 000 digits as a training set, 10 000 digits as a test set and 10 000 digits for an out-of-sample. We then projected training and testing sets (overall 60 000 points) to 500 dimensions using LLL with 1 000 landmarks selected using k-means algorithm with $K_Z = 50$. We did it number of times for different values of K_W from 1 to 200 and σ from 4.6 to 1000. Notice that $\mathbf{Z}$ matrix is independent from the affinities and depends only on the choice of landmark points, so we are able to save 30 seconds of runtime for each point by precomputing that matrix and using it for all variation of parameters. Given the location of the embedding points $\widetilde{X}$, we also computed out-of-sample projection matrix $\mathbf{Z}_{oos}$ to find the embedding of the out-of-sample set as well. We used the fact that the solution of LE, as any spectral problem, is nested into each other and computed the 1-nearest neighbor classification for different number of dimensions separately and reported the smallest error. We did it for both the test and the out-of-sample sets. Figure 4.8 shows the results. The smallest error is achieved for very small values of K_W. There is also little discrepancy between the error for test and out-of-sample sets, which signify the quality of our out-of-sample projection technique.

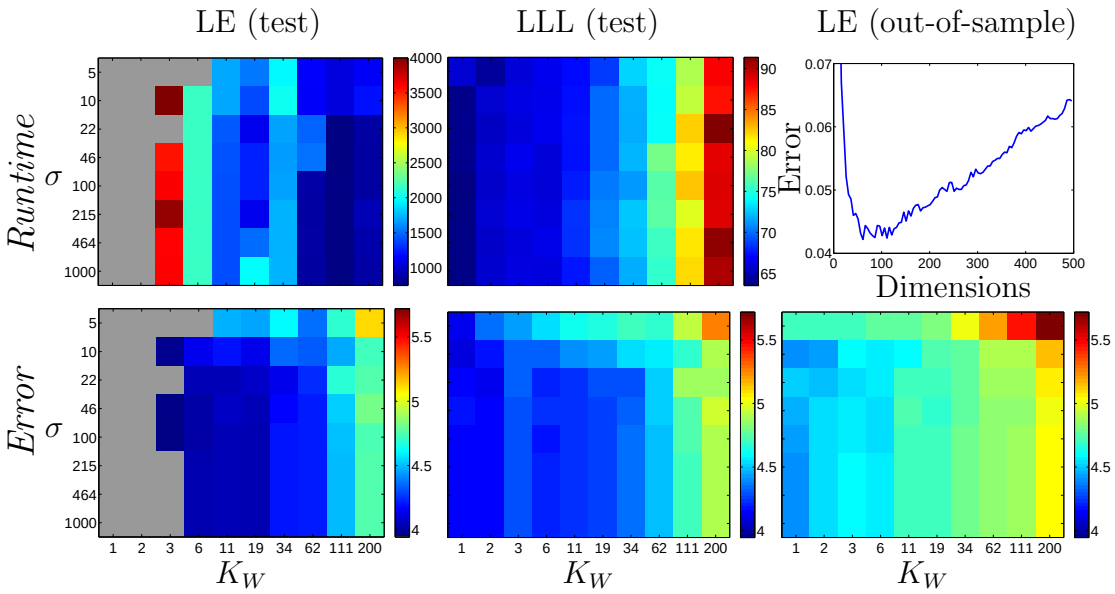

Figure 4.8: 1-nearest neighbor classification error for different values of σ and K_W of MNIST digits after applying LLL algorithm. See the text for a complete description of the setup. Gray areas corresponds to unconverted optimization of `eigs` routine. Top two plots show runtime for Exact LE and LLL. The color corresponds to the runtime is seconds. Bottom three plots show the error of Exact LE, LLL and out-of-sample set respectively. The color corresponds to classification error in percents. The runtime of out-of-sample is constant and equal to 30 seconds for all values of K_W and σ. The top right plot shows 1-nearest neighbor classification error for different dimensions for the test subset with $\sigma = 10$ and $K_W = 1$.

In the right top corner we show the error variation as we change the dimensionality. The results are shown for $\sigma = 10$ and $K_W = 1$, but the curve is very similar for other sets of parameters as well. Notice that the runtime of LLL is less than two minutes for the embedding of as many as 60 000 MNIST points.

We also tried to repeat the same experiment for Exact LE to compare the results with LLL, but we found a lot of complications. First of all, it turns out that for small values of K_W the graph Laplacian is not connected and MATLAB `eigs` routine does not converge (at least not all 500 requested eigenvalues). For the bigger values of K_W the algorithm converges, but requires a lot of iterations, which increase the runtime dramatically to almost 4 000 seconds. Notice, that exactly for those values, both Exact LE and LLL give the smallest error (in fact smallest error for LLL is achieved for $K_W = 1$, for which

Exact LE didn't even converge). Increasing K_W improves the connectivity of the graph Laplacian, but nevertheless the runtime didn't decrease much lower than 1 000 seconds, which means that LLL is 15–40× faster depending on the particular set of parameters. Finally, the general pattern and values of error is almost the same for Exact LE, LLL and out-of-sample set. The error is gradually increasing from the bottom left corner to the the upper right for all three cases.

Large-scale experiment. We used infinite MNIST dataset (see Appendix A for description) and reduced the dimensionality to $d = 2$ with 10 000 randomly selected landmarks and $K_Z = 5$ nearest landmarks. For LLL It took the algorithm 4.2 minutes to compute the projection matrix $\mathbf{Z}$ and 14 minutes to compute the embedding. We also run LE ($\mathbf{Z}$) on the same 10 000 landmarks. The resulting embedding is available in Fig. 4.9. For LLL embedding notice that zeros, sixes and ones are separated from the rest of the digits, and nines, fours and sevens form their own group (all those digits contain in them straight vertical line). For LE ($\mathbf{Z}$) embeddings the result is not that good. Only ones and a group containing sevens and nines can be separated. The rest of the points are trapped in the center of the figure.

4.8.2 PCA

For PCA, the natural comparison point is to run a PCA on a subset of points and then apply the learned linear mapping to the rest of points.
We used $N = 20\,000$ images from CIFAR dataset and apply PCA to find 10 largest principal component. We used random location of landmarks and changed the number of them logarithmically from 10 to N. We repeated the experiments 5 times to accommodate for randomness. For all number of landmarks we used $K_Z = L$. Fig. 4.10 we show the results. While LLL has much smaller error than the subset for any given number of landmarks, it is worse with respect to the runtime. First, this can be explained by the rapid decrease of the error for the subset. The linear mapping is easier to learn than the nonlinear and even a subset of 1 000 landmarks (5% of the data) the subset already gives a small error. Second, the subset does not involve any expensive additional operations and thus can afford to use much more landmarks for the same runtime (notice that the runtime for a subset almost does not grow for 10 to 1 000 landmarks).

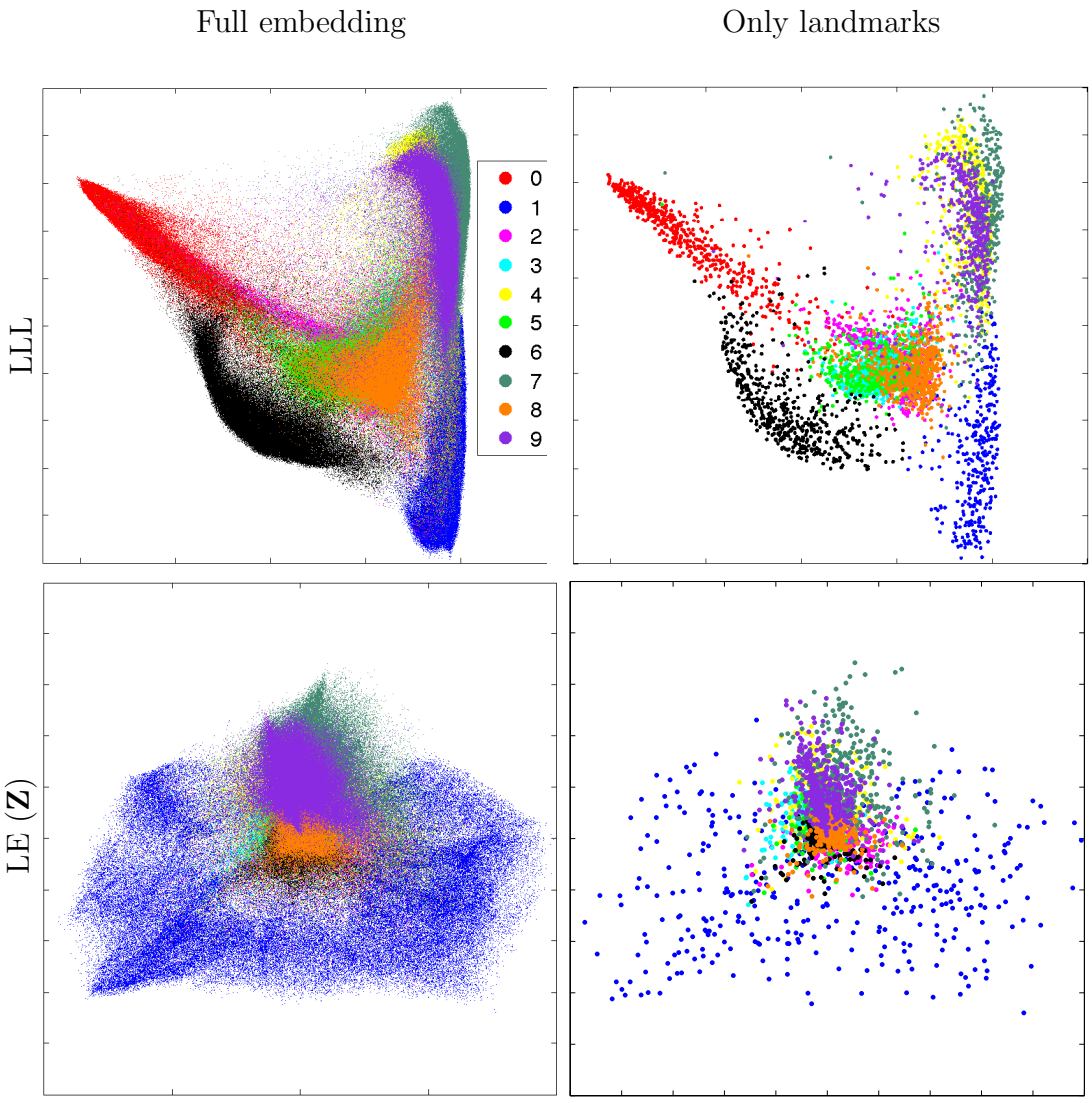

Figure 4.9: Embedding of 1 020 000 points from infiniteMNIST dataset using LLL with 10 000 landmarks. For LLL it took just 4.2 minutes to compute the projection matrix $\mathbf{Z}$ and 14 minutes to find the embedding. Shown the results of embedding of full dataset $\mathbf{X}$ and of the landmarks $\widetilde{\mathbf{X}}$ for LLL (*top row*) and LE ($\mathbf{Z}$) (*bottom row*).

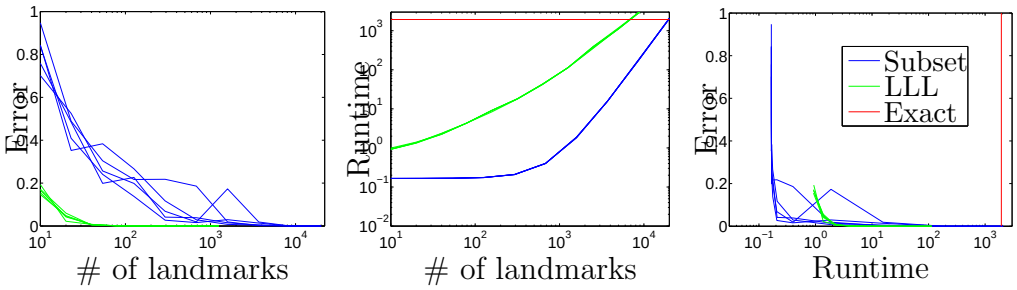

Figure 4.10: Results of PCA applied to the CIFAR dataset. *Left and central plot:* the error and the runtime change as number of landmarks grow. *Right plot:* error decrease as runtime grows. Solid red line corresponds to the exact run.

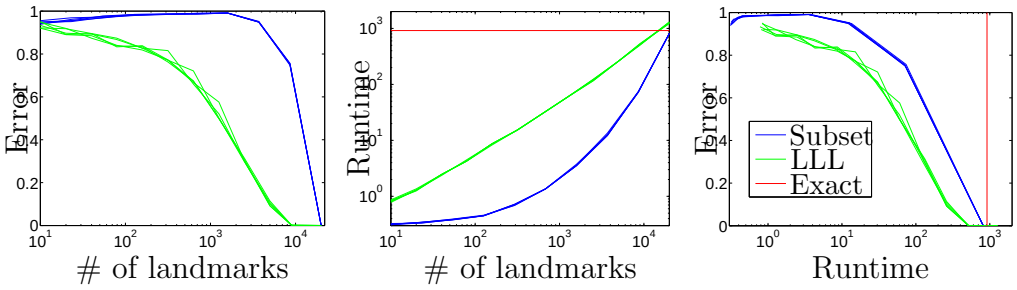

Figure 4.11: Results of LDA applied to the CIFAR dataset. *Left and central plot:* the error and the runtime change as number of landmarks grow. *Right plot:* error decrease as runtime grows. Solid red line corresponds to the exact run.

4.8.3 LDA

For the experiment we used the same subset of the CIFAR dataset and the same settings as in the PCA experiment, however this time we used the label information. In fig. 4.11 we show the results. Clearly, this time the results are very different from PCA. LLL has lower error against subset both for the same number of landmarks and, more importantly, for the same runtime. The reason for this is that for the subset the linear mapping used to project non-landmark points does not take into account the labels of those points. In comparison to this, reduced affinities constructed by LLL use the information from all the points.

4.8.4 Spectral Clustering

The solution of the spectral embedding (both exact and approximate) gives the matrix $\mathbf{X}$ containing the N K-dimensional projections (on which k-means will be run). Since the k-means solution is invariant to translations, rotations and scaling of $\mathbf{x}_1, \ldots, \mathbf{x}_N$ we can define a *projection error* between an approximate solution $\widetilde{\mathbf{X}}$ and the exact solution $\mathbf{X}$ by first finding the optimal Procrustes alignment between them, and then computing the Frobenius norm, normalized by the norm of the exact one: $\left\| procr(\widetilde{\mathbf{X}}) - \mathbf{X} \right\|_F / \left\| \mathbf{X} \right\|_F$, expressed as percentage. Moreover, as a general performance measure, we can compute the *clustering error* with respect to the exact spectral clustering. For this, we align the clusters using the Hungarian algorithm (Burkard et al., 2009) and report the fraction of misclassified points.

Number of landmarks. In the first experiment, similar to LE, we computed the runtime and the error for a different number of landmarks. We used random location for landmarks and average over 5 different runs. For LLL, we set the number of nearest landmarks to $K_Z = K + 1$.

In fig. 4.12 we show the results of LLL and Nyström approximations for 128×128 `cameraman` grayscale image ($N = 16384, D = 3$). We used sparse Gaussian affinities constructed using a sliding window over the pixels with a side $r = 10$ (1681 nearest neighbors for each point) with bandwidth $\sigma = 20$. First of all, when the number of landmarks is small, the subset used in Nyström has isolated components and thus is not able to project all the points. The out-of-sample points connect to the subets only for 200+ landmarks. As number of landmarks grows, the error improves little-by-little, but very slowly. We also observed that the location of landmarks plays important role for this algorithm, since sometimes same number of landmarks can give very different errors, depending on their location (e.g. there are few runs in lower left plot with low error that are clearly separated from the others). In comparison, LLL gives results for any number of landmarks, with both projection and clustring error consistently decreasing as the number of landmarks grows. Notice that the error-bars averaging the runs are pretty tight, which means that the algorithm is robust to different random initializations.

In fig. 4.13 we show the final clustering and the normalized eigenvectors ordered based

Another way to do it would be to compute the angle between the subspaces defined by the eigenvectors. We observed that it gives very similar errors to the one we use

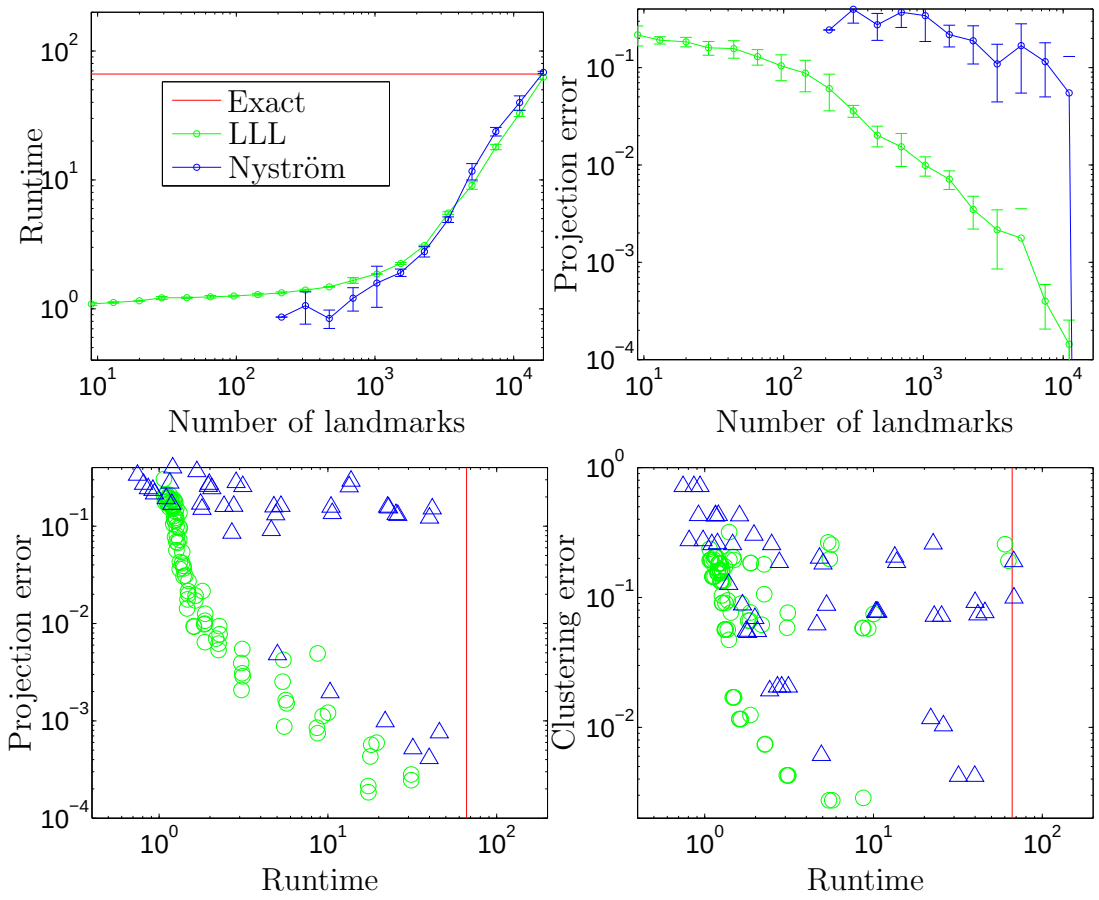

Figure 4.12: Image segmentation of 128×128 `cameraman`. *Top plot:* runtime and projection error as the number of landmarks grow. *Bottom plot:* projection error and clustering error with respect to exact spectral clustering as runtime grows. LLL consistently outperforms Nyström method for any number of landmarks.

on eigenvalues. Top row shows the exact algorithm, then we show those runs LLL and Nyström that were use fewest landmarks to achieve 10% and 1% clustering error. Comparing the number of those runs, to achieve 10% LLL needed 7× fewer landmarks than Nyström with 1.7× speed-up. For 1% accuracy, LLL needed 2.2× fewer landmarks and 2.15× speed-up. With respect to the exact algorithm, clustering performance of LLL is visually identical for both clustering and eigenvectors, but with LLL being almost 30× faster.

Model selection. In fig. 4.14 we show the model selection search for the parameters for simple Gaussian affinities for 10 000 digits from MNIST. Different from the exper-

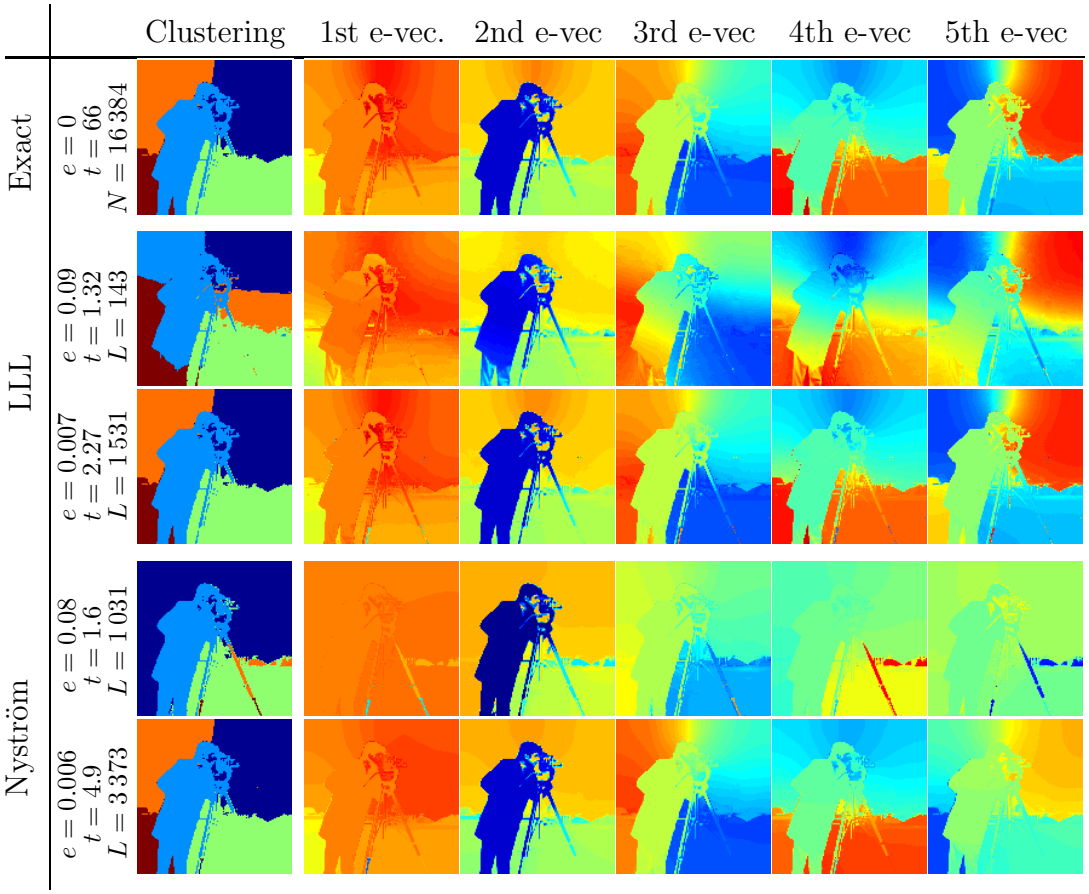

Figure 4.13: Clustering results and leading eigenvectors for exact spectral clustering, LLL and Nyström. For each row, left columns give a clustering error e, the runtime t and the number of landmarks L that were used to produce the results.

iment with LE (see fig. 4.8) in this experiment we can actually directly compare the classification error of the k-means clustering output with respect to the ground truth classes of MNIST letters, by predicting the majority class in each cluster, without needing additional classification algorithms like k-nearest neighbor classifiers. To compare, we run exact spectral clustering and LLL approximation with 1000 landmarks using 20 different values for σ and K_W chosen on a logarithmic grid from 1 to 10 000. Left two plots correspond to the classification error for the exact and LLL spectral clustering respectively. Notice that the error has similar pattern for both exact and approximate clustering. Also, for $K_W = 1$ exact clustering fails, because the affinity matrix decouples into many disconnected components which gives very high error. LLL reduced affinity matrix brings those components together again and gives meaningful error. Right two

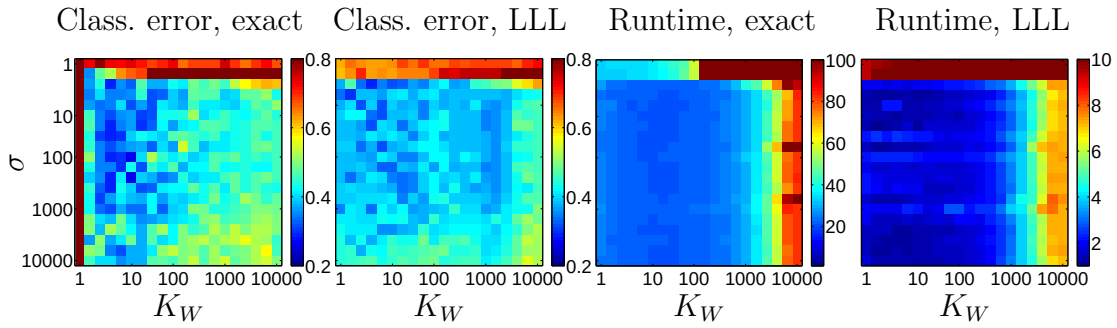

Figure 4.14: The result of model selection of the Gaussian affinities using exact spectral clustering and LLL approximation with different values of bandwidth σ and number of nearest neighbors K_W. The first two plots show the classification error with respect to ground truth classes and last two the runtime. Notice that the error pattern in very similar for exact and LLL, but the runtime is an order of magnitude smaller.

plots show the runtime of exact spectral clustering and LLL respectively with LLL being more than 10× faster than the exact one (notice different ranges of the color bar).

Non-Gaussian affinity matrix. One of the benefits of LLL comes from the fact that we can use it with any affinity matrix and with any explicit or implicit kernel that generates it. In contrary, Nyström requires well defined data-independent kernel function that generates the affinities, and, to our knowledge, was applied only to the Gaussian kernel (Talwalkar et al., 2008; Fowlkes et al., 2004; Bengio et al., 2004a). Not all affinities are Gaussian, however. For example, Carreira-Perpiñán and Zemel (2005) define affinity as an ensemble of multiple affinities constructed using minimum spanning tree on a neighborhood graph perturbed with noise. Constructed this way, the affinities tend to be more robust to shortcuts between distant parts of the manifold. In fig. 4.15 we show the results of the image segmentation for the 512×512 grayscale image of the house using LLL for spectral clustering with 3 000 landmarks and $K_Z = 5$. First, we used a graph ensemble of 10 affinities constructed with MST with every point perturbed using uniform random jitter with standard deviation $0.4\bar{d}$, where $\bar{d}$ is a mean distance to the available neighbors of that point. Second, we also show the results of traditional Gaussian affinities (with bandwidth $\sigma = 5$). Both affinities take as an input the neighborhood graph defined with sliding window over the spatial coordinates of side 10. Notice that the graph ensemble gives overall better results than Gaussian affinities.

Figure 4.15: Image segmentation of 512×512 house image (shown on the left) into four clusters using graph ensemble affinities from (Carreira-Perpiñán and Zemel, 2005) (center plot) and Gaussian affinities (right plot). Notice that the former segments the shadow in a separate cluster.

In particular, sky, windows and shadow of the house is better clustered using the former affinities.

Motion segmentation. Apart from the image segmentation we can also apply spectral clustering for motion segmentation in videos. We used a dataset of 41 video frames of a person walking around a room. Each frame is represented as 120×160 RGB image. Following Shi and Malik (1998), we can represent this dataset in both spatial and temporal domains, where each point has six coordinates: two representing its location on the plane, one for the number of the frame and three for color intensity. Overall, the dataset has $N = 787\,200$ points in $D = 6$ dimensions. We can define a neighborhood graph for this dataset by connecting points along both spatial and temporal domain. For spatial information, we used a sliding window with a side $r_s = 30$. For temporal, we connected each pixel with the pixel at the same location and its adjacent pixels of the previous and the next frames. Overall it gives a neighborhood graph with 963 neighbors for each point. We then build a Gaussian affinities with bandwidth $\sigma = 15$ for each pair of neighbors. Altogether, building this affinity matrix takes 6.8 minutes.

For LLL, it took 3 minutes of runtime for 5 000 random landmarks with one minute spent on computing the entries of matrix $\mathbf{Z}$ and one minute on computing the reduces affinities (4.5). Thus, we can find a good solution in half a the time needed to compute the affinity entries. Also, as we showed in the experiment with k-means in sec. 4.7,

Available at http://cmp.felk.cvut.cz/multicam/Demos/Students/BorrasJoan/

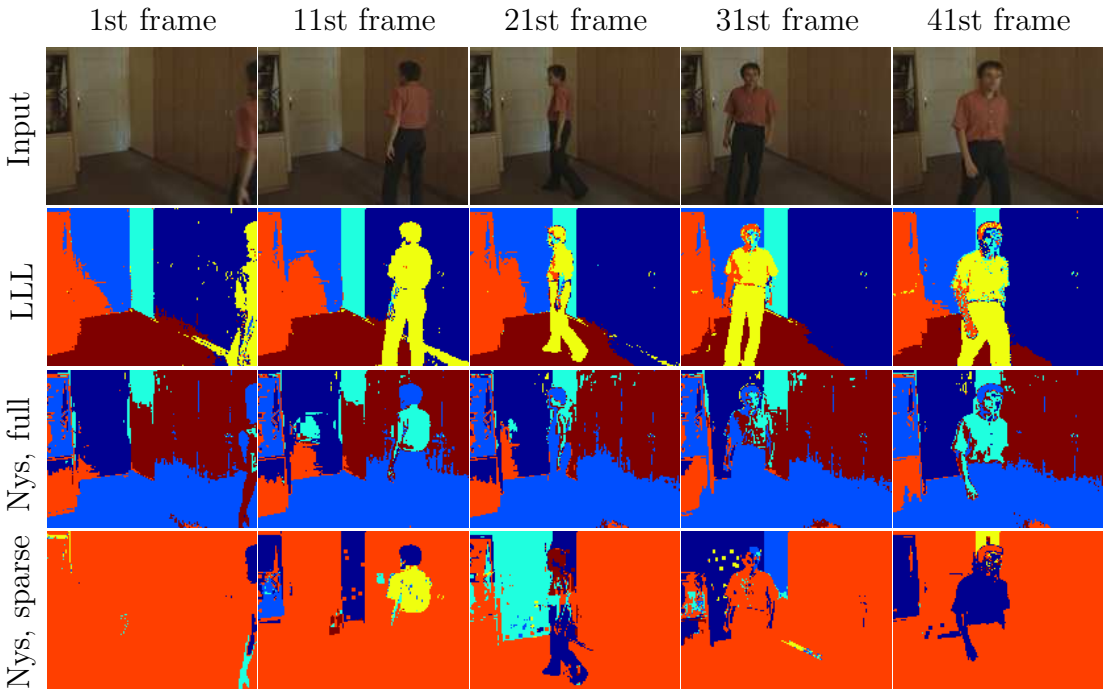

Figure 4.16: Spatio-temporal segmentation of 41 frame video with resolution 120×160. Top row shows 4 frames from the original video. Each subsequent row shows the clustering of those frames using LLL and Nyström methods that approximate full or sparse affinities respectively.

k-means with 20 restarts would take 91 seconds by itself which is comparable to the runtime of LLL. We also tried using larger number of landmarks, but the results did not improve dramatically.

For Nyström we first used sparse affinities, but with so few non-zeros, it is almost impossible to cover the entire dataset even with thousands of landmarks. We end up trying 10 000 landmarks before running out of memory. However, it still produced 35 singleton points that we assign as "missing" and which we did not included in the clustering. We also applied Nyström that approximates full affinities by computing the Gaussian kernel between all the point in the subset. However, this requires computing eigendecomposition of full $L \times L$ matrix as well as evaluating a Nyström out-of-sample kernel for $N - L$ non-landmark points, which are both quite costly. For $L = 3\,000$ landmarks these two steps already take more time than the runtime of the LLL, so we used this as comparison point.

In fig. 4.16 we show several frames form the original movie as well as their segmentation

results for LLL and Nyström. Notice that the cluster assignment for LLL corresponds to the meaningful objects (the floor, the person or the wall) and does not change from one frame to another. Comparing to this, Nyström shows inferior performance. The version approximating the full affinities does separate some meaningful information, such as cluster assignments for the person's torso, but generally the clusters does not represent any feature of the space. Nystöm that uses sparse affinities does not show almost any meaningful segmentation.

4.9 Discussion

LLL algorithm uses all the available information from the affinity matrix to construct reduced affinities that propagate . Nyström is restricted only to the subset of the affinities, which is wasteful, particularly if collecting the data is costly. The reason why spectral clustering works well is the neighborhood graph, which propagates label information over points within a cluster, according to the edges between points. Using a small subset of the vertices and the edges between those vertices coarsens the graph too drastically. From this point of view, the reduced graph in LLL can be seen as a contraction of the original graph that preserves more information about it.

A further speedup not explored here is the use of multiprocessors. The basic operations required are: (1) the computation of the sparse affinity matrix (itself possibly involving constructing a nearest-neighbor graph and computing the affinity values); (2) the computation of the weights Z; (3) the construction of the reduced spectral problem (involving matrix products); (4) the computation of the landmark eigenvectors; (5) the out-of-sample extension of the eigenvectors to the full data; (6) the random k-means restarts on the landmarks; and (7) the final k-means on the entire data. Most of these steps are easily parallelizable, in particular the more expensive ones that involve the entire, $\mathcal{O}(N)$, dataset.

4.10 Conclusion

Spectral methods for manifold learning and clustering often give good solutions to problems involving nonlinear manifold or complex clusters, and are in widespread use. However, scaling them up to large datasets (large N) and nontrivial numbers of eigenvectors

(large d) requires approximations. The Locally Linear Landmarks method proposes a reduced formulation of the original spectral problem that optimizes only over a small set of landmarks, while retaining structure of the whole data. The algorithm is well defined theoretically and has better performance than the Nyström method, allowing users to scale up applications to much bigger sizes. LLL also defines a natural out-of-sample extension that is cheaper and better than the Nyström method.

We show that the algorithm can be applicable to a wide range of spectral problems: manifold learning, such as Laplacian Eigenmaps, PCA and kPCA; clustering, such as Spectral Clustering; and supervised dimensionality reduction, such as LDA and kLDA. There exist other methods not included in this version, such as LLL for metric learning (Kulis, 2012), ISOMAP (Tenenbaum et al., 2000), MDS (Cox and Cox, 1994), MVU (Weinberger and Saul, 2004) and LLE (Saul and Roweis, 2003). In most of the studied cases (with exception of PCA) the algorithm shows robustly 10-20$\times$ speed up with small approximation error.

The basic framework of LLL, where we replace the low-dimensional projections by a fixed linear function of only a few of the projections, is applicable to any spectral method. However, the best choice of the linear function is an interesting topic of future research.

Chapter 5

Linear-time Training using N-Body approximations

5.1 Introduction

Although in chapter 3 we were able to significantly reduce the number of iterations of NLE algorithms, each iteration is still quadratic on the number of points N, and this does not scale to large datasets. In fact, no matter how fast we would make out optimization routine, it would just decrease the number of iterations needed for convergence, while each iteration is still going to be quadratic. This is prohibitively expensive for datasets with more than few hundred thousand points. Stochastic gradient descent is not helpful, because each step would only update a small subset of the $\mathcal{O}(N)$ parameters, becoming a form of alternating optimization. In this chapter we are going to try to break the quadratic cost of NLE iterations by approximating the gradient with N-body methods, in particular fast multipole methods (FMM; Greengard and Rokhlin, 1987). N-body problems arise when the exact computation involves the interaction between all pairs of points in the dataset. They are of particular importance in particle simulations in biology and astrophysics. Generally, there are two ways to speed up N-body problems: using a tree structure (e.g. Barnes and Hut, 1986) or using a FMM expansion, and they approximate the computations in $\mathcal{O}(N \log N)$ and $\mathcal{O}(N)$ time, respectively. FMMs also

This chapter is an extended version of Vladymyrov and Carreira-Perpiñán (2014).

have known error bounds (Baxter and Roussos, 2002), while the Barnes-Hut algorithm does not (Salmon and Warren, 1994). Unfortunately, both types of methods scale poorly with the latent-space dimensionality d. However, they work well for $d \leq 3$, which makes them suitable for visualization applications, and we focus on that here.

In section 5.2 we are going to review two categories of N-Body methods: the ones based on tree structure (most notably Barnes-Hut method) and the ones based on FMM. We also going to review few papers that have already tried to use N-Body methods for the fast training of NLE. Then, in section 5.2.3 we will show how we can approximate the computation of the objective function and the gradient of NLE using N-Body methods. In section 5.4 we will show an initial attempt to evaluate the quality of the approximation. We evaluate the role of noisy gradients and propose the use of increasing schedules for the accuracy parameter of N-body methods in order to speed up the optimization. Section 5.5 will show the embedding results of MNIST dataset and show how FMM compare with Barnes-Hut approximation and the exact evaluation. Finally, for large-scale experiment we show that FMM approximation is able to find an embedding of million-point infiniteMNIST dataset in three hours' runtime. We will finish the chapter with some conclusive remarks.

5.2 Review of N-Body Methods

All fast computation methods for N-body problems produce approximate, rather than exact, values for sums of $\mathcal{O}(N^2)$ interactions. They are generally based on tree structures, such as the $\mathcal{O}(N \log N)$ Barnes-Hut (BH) method; or on series expansions, such as the $\mathcal{O}(N)$ fast multipole method (FMM) and fast Gauss transform (FGT), which besides have bounds for the approximation error.

5.2.1 Tree-based Methods

Here, we build a tree structure around the points $\mathbf{X}$, such as *kd*-trees, ball-trees or range-trees (Friedman et al., 1977; Samet, 2006), and we query tree nodes rather than individual points. Each node of the tree represents a subset of the data contained in a d-dimensional cell, usually a box aligned with the coordinate axes. The root node represents the whole dataset and each new level partitions the space into subsets (e.g. in the middle of the largest-variance dimension) until there is only one point left in

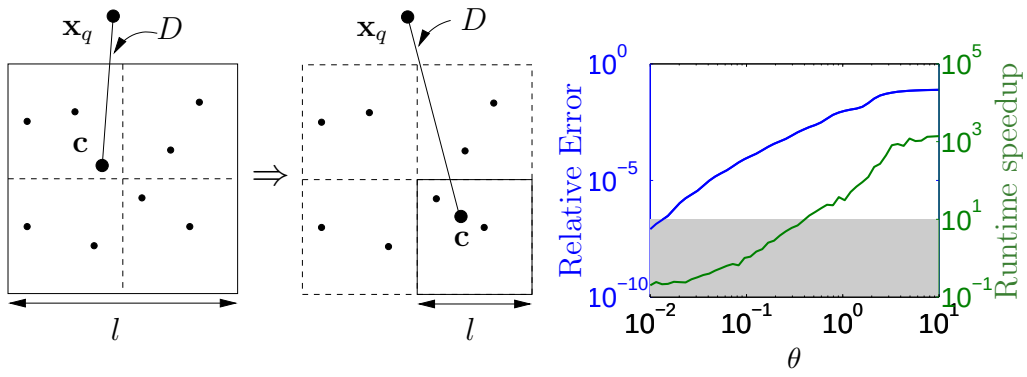

Figure 5.1: *Left*: for $l/D > \theta$, the cell is subdivided into smaller subcells. Otherwise, the interaction is computed approximately. *Right*: speedup and relative error for different values of θ. The gray area corresponds to the region with no speedup. Notice the log/log plot.

each leaf node. The tree can then be used to locate points within a given distance of a query point without exhaustive search on the entire dataset. For faster, approximate calculations, we replace many point-point interactions with point-node interactions, by pruning nodes too far away or by subsuming all points in a small cell into one interaction. In machine learning, this idea has been used to speed up various nonparametric models, such as regression with locally weighted polynomials (Moore et al., 1997) or Gaussian processes (Shen et al., 2006). Dual-trees (Gray and Moore, 2001) yield further speedups by building trees for both target and query points, which allows node-node interactions besides point-node ones.

We focus here on the Barnes and Hut (1986) (BH) method. This first constructs a quadtree in 2D (octree in 3D) around the set of target points. Then, for every query point $\mathbf{x}_q$, it traverses the tree down from the root until the cell can be considered approximately as a single point because it is sufficiently small and far from $\mathbf{x}_q$, as follows. For a cell of size l, let D be the distance between the cell's center of mass $\mathbf{c}$ and $\mathbf{x}_q$ (see fig. 5.1 left). If the fraction l/D is smaller than a user-defined parameter θ, then all the interactions between $\mathbf{x}_q$ and the points inside that cell are approximated by a single interaction with $\mathbf{c}$. If the fraction is bigger than θ, the algorithm continues to explore the children of the node. If we reach a leaf, the interaction is computed exactly, since it contains only one point, otherwise an approximation error is incurred. As a function of N, the construction of the tree costs $\mathcal{O}(N \log N)$ and for each of the N query points, the interaction is computed in expected $\mathcal{O}(\log N)$ time. Thus, the overall cost reduces from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

The user parameter θ controls the trade-off between the accuracy of the solution and the runtime speedup. Increasing θ means we approximate cells that are bigger or closer to the query point. This reduces the runtime because we prune the tree earlier, but also increases the approximation error. Fig. 5.1(right) shows the relative error and the speedup compared to the exact computation for different values of θ. Good speedups with small relative error occur for $\theta \in [0.5, 2]$, roughly, but this region does vary with each problem.

Tree-based algorithms have some limitations. Most crucially, the tree size grows exponentially with the dimension d, thus limiting their use to problems with low dimensionality. Second, the approximation quality declines when the interaction scale (e.g. the Gaussian kernel bandwidth) is too big or too small. The hierarchical fast Gauss transform (Lee et al., 2006) somewhat alleviates the second problem by combining dual trees with fast multipole methods, but it still does not work well when $d > 3$. Finally, it is hard to estimate the approximation error, which in fact can be unbounded (Salmon and Warren, 1994).

5.2.2 Fast Multipole Methods

Fast multipole methods (FMM) were initially used in astrophysics to compute gravitational interactions between many particles (Greengard and Rokhlin, 1987) and have since enabled large particle simulations in many areas. The idea of FMMs is to do a series expansion of the interactions locally around every point such that the point pair decouples in each term of the series. Truncating the series reduces the cost from quadratic to linear. The fast Gauss transform (FGT; Greengard and Strain, 1991) applies this to compute sums of Gaussian interactions

$$Q(\mathbf{x}_n) = \sum_{m=1}^{N} q_m \exp(-\left\|(\mathbf{x}_n - \mathbf{x}_m)/\sigma\right\|^2) \tag{5.1}$$

for a set of points $\mathbf{x}_n,\ n = 1, \ldots, N$ and a bandwidth σ. It has been applied to accelerate problems such as kernel density estimation (Raykar and Duraiswami, 2006a) and matrix inversion and eigendecomposition (de Freitas et al., 2006) in machine learning.

In the FGT, we start by normalizing the points to lie in the unit hypercube and dividing the space into boxes of side $\sqrt{2}\sigma r$, where $r < 1/\sqrt{2}$ is a user parameter. Defining $h_n(t) = e^{-t^2} H_n(t)$ as Hermite functions with Hermite polynomials $H_n(t)$, there exist three different ways to compute the approximation to (5.1) (cf. fig. 5.2):

- We can use a Hermite expansion for each box $\mathcal{B}$ (with center $\mathbf{s}_{\mathcal{B}}$) and evaluate it for all target points $\mathbf{t}$:

$$Q(\mathbf{t}) = \sum_{\mathcal{B}} \sum_{\boldsymbol{\alpha}<p} A_{\boldsymbol{\alpha}}^{\mathcal{B}} h_{\boldsymbol{\alpha}}\Big(\frac{\mathbf{t}-\mathbf{s}_{\mathcal{B}}}{\sigma}\Big) + \epsilon_H(p), \tag{5.2}$$

  where $A_{\boldsymbol{\alpha}}^{\mathcal{B}} = \frac{1}{\boldsymbol{\alpha}!}\sum_{\mathbf{s}_j \in \mathcal{B}} q_j (\frac{\mathbf{s}_j - \mathbf{s}_{\mathcal{B}}}{\sigma})^{\boldsymbol{\alpha}}$ and $\epsilon_H(p)$ is a known error term, defined below.

- For every source point in a box $\mathcal{B}$ we can form a Taylor series of the target points $\mathbf{t}$ in the nearby boxes $\mathcal{C}$ with corresponding centers $\mathbf{t}_{\mathcal{C}}$:

$$Q(\mathbf{t}) = \sum_{\boldsymbol{\beta}<p} \left(\sum_{\mathcal{B}} C_{\boldsymbol{\beta}}^{\mathcal{B}\mathcal{C}} \right) \Big(\frac{\mathbf{t}-\mathbf{t}_{\mathcal{C}}}{\sigma}\Big)^{\boldsymbol{\beta}} + \epsilon_T(p), \tag{5.3}$$

  where $C_{\boldsymbol{\beta}}^{\mathcal{B}\mathcal{C}} = \frac{1}{\boldsymbol{\beta}!}\sum_{\mathbf{s}_j \in \mathcal{B}} q_j h_{\boldsymbol{\beta}}(\frac{\mathbf{s}_j - \mathbf{t}_{\mathcal{C}}}{\sigma})$ and $\epsilon_T(p)$ is a known error term, defined below.

- We can further approximate (5.2) by expanding $h_{\boldsymbol{\alpha}}(\mathbf{t})$ as a Taylor series to get

$$Q(\mathbf{t}) = \sum_{\boldsymbol{\beta}<p} \left(\sum_{\mathcal{B}} \widehat{C}_{\boldsymbol{\beta}}^{\mathcal{B}\mathcal{C}} \right) \Big(\frac{\mathbf{t}-\mathbf{t}_{\mathcal{C}}}{\sigma}\Big)^{\boldsymbol{\beta}} + \epsilon_{TH}(p), \tag{5.4}$$

  where $\widehat{C}_{\boldsymbol{\beta}}^{\mathcal{B}\mathcal{C}} = \frac{1}{\boldsymbol{\beta}!}\sum_{\boldsymbol{\alpha}<p} A_{\boldsymbol{\alpha}}^{\mathcal{B}} (-1)^{|\boldsymbol{\alpha}|} h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}(\frac{\mathbf{s}_{\mathcal{B}} - \mathbf{t}_{\mathcal{C}}}{\sigma})$ and $\epsilon_{TH}(p)$ is a known error term, defined below.

The approximation errors are given in Baxter and Roussos (2002) with an extended derivation from Raykar (2006):

$$\epsilon_H(p) \leq \frac{\sum_{m=1}^{N} q_m}{(1-r)^d} \sum_{k=0}^{d-1} \binom{d}{k} (1-r^p)^k \left(\frac{r^p}{\sqrt{p!}}\right)^{d-k}, \tag{5.5}$$

$$\epsilon_T(p) \leq \frac{\sum_{m=1}^{N} q_m}{(1-r)^d} \sum_{k=0}^{d-1} \binom{d}{k} (1-r^p)^k \left(\frac{r^p}{\sqrt{p!}}\right)^{d-k}, \tag{5.6}$$

$$\epsilon_{TH}(p) \leq \epsilon_T(p) + \frac{\sum_{m=1}^{N} q_m}{(1-\sqrt{2}r)^{2d}} \left(\sum_{k=0}^{d-1} \binom{d}{k} (1-(\sqrt{2}r)^p)^k \left(\frac{(\sqrt{2}r)^p}{\sqrt{p!}}\right)^{d-k} \right)^2. \tag{5.7}$$

Wan and Karniadakis (2006) give tighter bounds.

Computing multiindex sums over $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ scales as $\mathcal{O}(p^d N)$, which is the bottleneck computation. The rest of the computations add a linear cost to the algorithm and depend of the number of boxes included around every source box $\mathcal{B}$.

We use multi-index notation: $\boldsymbol{\alpha} \geq 0 \Rightarrow \alpha_1, \ldots, \alpha_d \geq 0$; $\boldsymbol{\alpha}! = \alpha_1! \cdots \alpha_d!$; $\mathbf{t}^{\boldsymbol{\alpha}} = t_1^{\alpha_1} \cdots t_d^{\alpha_d}$ for $\boldsymbol{\alpha} \in \mathbb{N}^d,\ \mathbf{t} \in \mathbb{R}^d$.

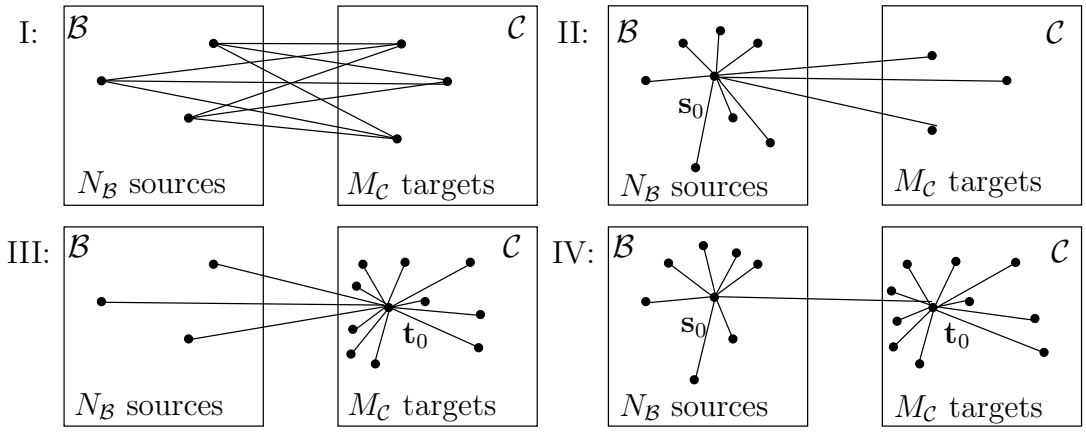

Figure 5.2: Different FGT approximations. I: exact interaction (5.1) (few points in both boxes); II: expansion around $\mathbf{s}_0$ (many source points); III: expansion around $\mathbf{t}_0$ (many target points); IV: expansion around $\mathbf{s}_0$ and then Taylor expansion to the Hermite functions (many points in both boxes).

The choice of the method depends on the number of source points $N_\mathcal{B}$ in a given box $\mathcal{B}$ and the number of target points $M_\mathcal{C}$ in a given box $\mathcal{C}$. The user provides two cutoff parameters $\bar{N}_\mathcal{B}$ and $\bar{M}_\mathcal{C}$. Now, one of the following can occur:

- If $N_\mathcal{B} < \bar{N}_\mathcal{B}$ and $M_\mathcal{C} < \bar{M}_\mathcal{C}$ use exact evaluation (5.1).
- If $N_\mathcal{B} < \bar{N}_\mathcal{B}$ and $M_\mathcal{C} \geq \bar{M}_\mathcal{C}$ use Hermite expansion (5.2).
- If $N_\mathcal{B} \geq \bar{N}_\mathcal{B}$ and $M_\mathcal{C} < \bar{M}_\mathcal{C}$ use Taylor expansion (5.3).
- If $N_\mathcal{B} \geq \bar{N}_\mathcal{B}$ and $M_\mathcal{C} \geq \bar{M}_\mathcal{C}$ use Hermite expansion followed by a Taylor expansion (5.4).

To gain additional speedup we can use the fast decay of the Gaussian and compute the interaction to target points that are located no further than K boxes away from the box with the source point. However, note that the FMM is still $\mathcal{O}(N)$ with heavy-tailed kernels such as the gravitational interaction.

The main drawback of FMMs and the FGT is that they are limited to small dimensions d (due to the p^d cost). The improved FGT (Yang et al., 2003) uses clustering and other techniques to grid the data into data-dependent regions, and a modified Taylor expansion so the cost is $\mathcal{O}(d^p N)$. This allows for somewhat larger dimensions, but the issue still remains, and the IFGT needs careful setting of various parameters (Raykar and Duraiswami, 2006b), or otherwise the overhead is so large that computing the exact

interaction is actually cheaper. In this chapter, we focus on $d \leq 3$ and the plain FGT with parameters $r = 1/2$, $\bar{N}_{\mathcal{B}} = \bar{M}_{\mathcal{C}} = 5$, $K = 4$, so that the quality of the approximation is controlled using just the order of the expansion p.

FMMs do have important advantages over BH: their cost is lower ($\mathcal{O}(N)$ vs $\mathcal{O}(N \log N)$), they work well on a wide range of kernel bandwidths, and they have known bounds for the approximation error as a function of p.

While in this chapter we concentrate on the Gaussian kernel (and the FGT), it is possible to use FMMs for virtually any kernel, for example the "kernel-independent" FMM (Ying et al., 2004; Fong and Darve, 2009) needs only numerical values of the kernel.

5.2.3 Related Work

N-body problems arise in the graph drawing literature, where the goal is to visualize in an aesthetically pleasing way edges and vertices of a given graph, which is typically unweighted and sparse (Battista et al., 1999). This is similar to dimensionality reduction given an affinity (or adjacency) matrix. One of the most successful algorithms for graph drawing are force-directed methods (Battista et al., 1999; Fruchterman and Reingold, 1991), which try to balance attractive and repulsive forces on the graph vertices in a similar formulation to that of NLEs (eq. (3.1)). Each iteration of the force-directed method requires the computation of interactions between every pair of points, which is $\mathcal{O}(N^2)$ for a graph with N vertices. Fast, approximate graph drawing is done with the BH algorithm (Quigley and Eades, 2000; Hu, 2005) in $\mathcal{O}(N \log N)$ runtime. Recently, the BH algorithm has been used to speed up the training of NLEs (van der Maaten, 2013; Yang et al., 2013) in a similar way to the work in graph drawing. The use of dual trees and FMMs to speed up gradient descent training of stochastic neighbor embedding (SNE) was also proposed by de Freitas et al. (2006), as a particular case of their work on N-body methods for matrix inversion and eigendecomposition problems in machine learning. Our work provides a more thorough study of N-body methods and the FGT for NLEs and demonstrates it in million-point datasets.

5.3 Applying N-body Methods to Embeddings

For NLEs the N-body problem appears in the computation of the objective function and the gradient, where the interactions between all point pairs must be evaluated. In

particular, the objective function of NLE (3.1) involves two N-body problems, one for each of the attractive and repulsive terms. The computation of the attractive term can be mitigated by the nature of the matrix $\mathbf{W}$: in most practical applications it is sparse and thus can be computed in linear time. The repulsive term is not sparse and involves an N-body problem as a sum of kernel similarities between all point pairs. For the gradient, the first term involves the graph Laplacian $\mathbf{L}$, which has the same sparsity pattern as $\mathbf{W}$ and can be computed efficiently. The second term involves the graph Laplacian $\widetilde{\mathbf{L}} = \widetilde{\mathbf{D}} - \widetilde{\mathbf{W}}$, which depends on $\mathbf{X}$ through a kernel in $\widetilde{\mathbf{W}}$. Let us define the following kernel interactions:

$$S(\mathbf{x}_n) = \sum_{m=1}^{N} K(\|\mathbf{x}_n - \mathbf{x}_m\|^2), \qquad S^x(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{x}_m K(\|\mathbf{x}_n - \mathbf{x}_m\|^2). \tag{5.8}$$

Now we can rewrite the objective function and the gradient of EE and s-SNE as follows:

$$E(\mathbf{X}) = \sum_{n,m=1}^{N} w_{nm} \|\mathbf{x}_n - \mathbf{x}_m\|^2 + \lambda \sum_{m=1}^{N} f(S(\mathbf{x}_m)),$$
$$\frac{\partial E}{\partial \mathbf{X}} = 4\mathbf{X}\mathbf{L} - 4\lambda Z(\mathbf{X})\mathbf{X}\operatorname{diag}(S(\mathbf{X})) + 4\lambda Z(\mathbf{X})S^x(\mathbf{x}_n),$$

where $f(x) = \log x$, $Z(\mathbf{X}) = 1/\sum_{n=1}^{N} S(\mathbf{x}_n)$ for s-SNE and $f(x) = x$, $Z(\mathbf{X}) = 1$ for EE. Given $S(\mathbf{x}_n)$ and $S^x(\mathbf{x}_n)$ both the objective function and the gradient can be computed in linear time.

The BH method can be applied to compute approximately the kernel interactions (5.8). We get

$$S(\mathbf{x}_n) \approx \sum_{m=1}^{\hat{N}} N_m K\big(\|\mathbf{c}_m - \mathbf{x}_n\|^2\big), \qquad S^x(\mathbf{x}_n) \approx \sum_{m=1}^{\hat{N}} N_m \mathbf{c}_m K\big(\|\mathbf{c}_m - \mathbf{x}_n\|^2\big),$$

where N_m and $\mathbf{c}_m$ for $m = 1, \dots, \hat{N}$ are the number of points and the centers of mass of the cells, respectively, for which we need to compute the interaction. For the weighted kernel interaction $S^x(\mathbf{x}_n)$ we require an additional approximation of each weight $\mathbf{x}_m$, due to its dependence on m. Fortunately, when we compute the approximation between the cell and the query point, the cell size is small (compared to the distance to the query point) and thus can be approximated by its center of mass.

For the FGT, $S(\mathbf{x}_n)$ can be obtained by taking $\sigma = 1$ and $q_n = 1$ in (5.1) for all $n = 1, \dots, N$. $S^x(\mathbf{x}_n)$ is recovered by taking $\sigma = 1$ and $q_n = x_{kn}$ and computing the formula d times for $k = 1, \dots, d$.

For t-SNE we cannot apply the FGT, because the former uses the t-Student kernel. However, a FMM approximation could be derived with a suitable series expansion, or with a kernel-independent FMM method (section 5.2).

Out-of-Sample Mapping The N-body approximation can also be used to obtain a fast out-of-sample mapping. Carreira-Perpiñán and Lu (2007); Carreira-Perpiñán (2010) compute the projection of a new test point $\mathbf{y}$ by keeping the projection of the training points $\mathbf{X}$ fixed and minimizing the objective function of the NLE wrt the unknown projection $\mathbf{x}$ (the mapping of a new $\mathbf{x}$ point to $\mathbf{y}$-space is defined analogously). For example, for EE:

$$\min_{\mathbf{x}} E'(\mathbf{x}, \mathbf{y}) = 2 \sum_{n=1}^{N} \big(w(\mathbf{y}, \mathbf{y}_n) \left\| \mathbf{x} - \mathbf{x}_n \right\|^2 + \lambda \exp\big(- \left\| \mathbf{x} - \mathbf{x}_n \right\|^2 \big) \big). \tag{5.9}$$

For M new test points the formula above can be approximated in $\mathcal{O}(M + N)$ using N-body methods (iterating all M minimizations synchronously), instead of $\mathcal{O}(NM)$ with the exact computation.

Optimization Strategy Since exact values of the objective function and gradient are not available during the optimization, it makes sense not to use a line search (it might be possible to use line searches with the FGT because it does give us an interval for the true value). This also saves time, since the line search would require repeated evaluations of the objective function. So the only N-body problem we need to solve per iteration is the gradient.

Our problem has similarities with stochastic gradient descent, for which a convergence theory exists (Spall, 2003, ch. 4.3), which leads to Robbins-Monro schedules that decrease the step size over iterations in a specific way. However, NLE training is different in that the number of parameters is proportional to the number of training points and the characteristics of the "noise" in the gradient (the approximation error) are not well understood. As far as we know, no convergence theory exists for NLEs. We provide an initial study of the role of this noise in section 5.4.

In pilot runs, we found that schedules that decrease the step size over iterations can improve the performance, but they are difficult to use in a robust way over different problems. Thus, for our experiments we use a constant step size η, chosen sufficiently small, which is simpler.

5.4 Analysis of the Effect of Approximate Gradients in the Optimization

The parameters that quantify the trade-off between the accuracy and the speedup are θ for BH and p for FGT. A higher value of p (or lower of θ) increases the accuracy, but so does the runtime. Clearly, the speed at which the optimization progresses and whether it converges depend crucially on these accuracy parameters. Here, we try to gain some understanding of this by considering the iterate updates as noisy, where the "noise" comes from the approximation error incurred and has a variance that grows with p. In order to solve the mathematical derivations, we will assume zero-mean Gaussian noise, which implies that the error is not systematic, as one might intuitively expect. This will allow us to derive some expressions that seem to hold in experiments, at least qualitatively.

At iteration k during the optimization of an objective function $E(\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^n$, if using exact gradient evaluations, we would move from the previous iterate $\mathbf{x}_{k-1}$ to the current one $\mathbf{x}_k$ without error (for example, for a gradient descent step, $\mathbf{x}_k = \mathbf{x}_{k-1} - \eta \nabla f(\mathbf{x}_{k-1})$). However, if using an inexact gradient, we would move to $\mathbf{x}_k + \boldsymbol{\epsilon}_k$, incurring an error $\boldsymbol{\epsilon}_k$. In our case, $\boldsymbol{\epsilon}_k$ is caused by using an approximate method and is a deterministic function of $\mathbf{x}_{k-1}$ and the method parameters (θ for Barnes-Hut, p for fast multipole methods, etc.). Let us model $\boldsymbol{\epsilon}_k$ as a zero-mean Gaussian with variance ξ^2 in each dimension. The fundamental assumption is that, although $\boldsymbol{\epsilon}_k$ is deterministic at each iterate, over a sequence of iterates we expect it not to have a preferred direction (i.e., no systematic error). The value of ξ corresponds to the accuracy level of the method, where $\xi = 0$ means no error ($\theta = 0$, $p \to \infty$). In practice, ξ will be quite small. Then we have the following result.

Theorem 5.4.1. *Let $E(\mathbf{x})$ be a real function with $\mathbf{x} \in \mathbb{R}^n$. Call $\Delta E(\mathbf{x})$ and $\delta E(\mathbf{x})$ the absolute and relative error, respectively, incurred at point $\mathbf{x} \in \mathbb{R}^d$ upon a perturbation of $\mathbf{x}$ that follows a Gaussian noise model $\mathcal{N}(\mathbf{0}, \xi^2 \mathbf{I})$. Call $\mu_\Delta(\mathbf{x}) = \langle \Delta E(\mathbf{x}) \rangle$, $v_\Delta(\mathbf{x}) = \left\langle (\Delta E(\mathbf{x}) - \langle \Delta E(\mathbf{x}) \rangle)^2 \right\rangle$, $\mu_\delta(\mathbf{x}) = \langle \delta E(\mathbf{x}) \rangle$ and $v_\delta(\mathbf{x}) = \left\langle (\delta E(\mathbf{x}) - \langle \delta E(\mathbf{x}) \rangle)^2 \right\rangle$ the expected errors and their variances under the noise model. Assume E has derivatives up to order four that are continuous and have finite expectations under the noise model. Call $\mathbf{g}(\mathbf{x}) = \nabla E(\mathbf{x})$ and $\mathbf{H}(\mathbf{x}) = \nabla^2 E(\mathbf{x})$ the gradient and Hessian at that point, respectively, and $\mathbf{J_H}(\mathbf{x})$ the $d \times d$ Jacobian matrix of the Hessian diagonal elements, i.e.,*

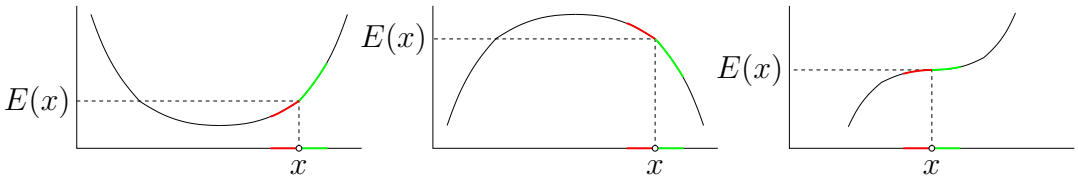

Figure 5.3: The result of a Gaussian perturbation to a point x on the function $E(x)$ in 1D. With positive mean curvature (left), the perturbation is equally likely to move x to the left or to the right, but points to the right have a larger, positive error in E, while points to the left have a smaller, negative error in E, and the net effect is that the perturbed E value is larger than $E(x)$ on average. With negative mean curvature (middle), the opposite is true. With zero mean curvature (right), the perturbed E value is zero to first order.

$(\mathbf{J_H}(\mathbf{x}))_{ij} = \partial h_{ii}/\partial x_j = \partial^3 E(\mathbf{x})/\partial x_i^2 \partial x_j$. *Then, the expected errors and their variances satisfy,* $\forall \mathbf{x} \in \mathbb{R}^d$*:*

$$\mu_\Delta = \frac{1}{2}\xi^2 \operatorname{tr}(\mathbf{H}(\mathbf{x})) + \mathcal{O}(\xi^4) \tag{5.10}$$

$$v_\Delta(\mathbf{x}) = \xi^2 \|\mathbf{g}(\mathbf{x})\|^2 + \xi^4 \left(\frac{1}{2}\|\mathbf{H}(\mathbf{x})\|_F^2 + \mathbf{1}^T \mathbf{J_H}(\mathbf{x})\mathbf{g}(\mathbf{x})\right) + \mathcal{O}(\xi^6) \tag{5.11}$$

$$\mu_\delta = \mu_\Delta / E(\mathbf{x}) \qquad v_\delta = v_\Delta / E(\mathbf{x})^2. \tag{5.12}$$

If $\|\mathbf{H}(\mathbf{x})\|_2 \leq M \ \forall \mathbf{x} \in \mathbb{R}^d$ *for some* $M > 0$*, then* $\forall \mathbf{x} \in \mathbb{R}^n$*:*

$$|\mu_\Delta| \leq \frac{1}{2}\xi^2 dM. \tag{5.13}$$

The proof to this theorem is given in Vladymyrov and Carreira-Perpiñán (2014).

Note that the mean error in eq. (5.10) depends on the point $\mathbf{x}$, i.e., the iterate where we apply the approximate step, through the trace of the Hessian at that point (and the accuracy level ξ, which we assume fixed by the user). It does not depend on the gradient itself, because the linear term is an odd function that integrates to zero. The formula for the mean error is accurate when ξ is small, which means the accuracy in the gradient evaluation is high. If the function E is quadratic, the formulas are exact. The bound for the mean error in eq. (5.13) is valid at any iterate (i.e., it does not depend on $\mathbf{x}$), but will typically be too coarse, and it also loses the information about the sign of the error. The formula for the mean error in eq. (5.10) has a simple geometric interpretation (see fig. 5.3): while a Gaussian perturbation is symmetric in $\mathbf{x}$-space, the value of $E(\mathbf{x} + \boldsymbol{\epsilon})$ is not because of the curvature in E, so the average of the E-error is not zero.

The behavior of the variance of the absolute error during the optimization can be characterized as follows. The variance is, to first order, proportional to the squared gradient, so we expect large variations in the error in early stages of the optimization. Near a minimizer, $\mathbf{g}(\mathbf{x}) \approx \mathbf{0}$ and so the coefficient of variation of the absolute error is

$$\frac{\sqrt{v_\Delta(\mathbf{x})}}{\mu_\Delta(\mathbf{x})} \approx \sqrt{2}\frac{\|\mathbf{H}(\mathbf{x})\|_F}{\operatorname{tr}(\mathbf{H}(\mathbf{x}))} = \sqrt{2}\frac{\|\boldsymbol{\lambda}(\mathbf{x})\|_2}{\|\boldsymbol{\lambda}(\mathbf{x})\|_1} \in \left[\sqrt{\frac{2}{d}}, \sqrt{2}\right]$$

since $\|\mathbf{x}\|_1/\sqrt{d} \leq \|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1 \ \forall \mathbf{x} \in \mathbb{R}^d$, $\operatorname{tr}(\mathbf{H}(\mathbf{x})) = \sum_{i=1}^d \lambda_i$ and $\|\mathbf{H}(\mathbf{x})\|_F^2 = \sum_{i=1}^d \lambda_i^2$, where $\boldsymbol{\lambda}(\mathbf{x}) = (\lambda_1, \ldots, \lambda_d)^T \geq \mathbf{0}$ are the eigenvalues of $\mathbf{H}(\mathbf{x})$. Thus, the coefficient of variation of the absolute error is independent of the accuracy level ξ and dependent only on the curvature. The ends of the interval above occur when all the eigenvalues are equal ($\sqrt{2/d}$) or at most one eigenvalue is nonzero ($\sqrt{2}$). In practice, if n is large we are likely to have many nonzero eigenvalues and thus be closer to the $\sqrt{2/d}$ end, so the coefficient of variation will be very small. Hence, near a minimizer we expect to see absolute errors with a near-constant value of $\mu_\Delta(\mathbf{x}) = \frac{1}{2}\xi^2 \operatorname{tr}(\mathbf{H}(\mathbf{x}))$.

This allows us to characterize the behavior of the optimization near the minimizer. Assume that, if using exact gradients, we converge linearly with rate $0 < r < 1$, e.g. $E_{k+1} = rE_k$ where E_k is the exact value of the objective function $E(\mathbf{x}_k)$ at iterate k (and we assume w.l.o.g. that $E_k \to E^* = 0$). Using the approximate gradients, the sequence of objective function values is instead $e_{k+1} \approx re_k + \mu_\Delta(\mathbf{x}_k) \approx re_k + \mu$, where $\mu = \frac{1}{2}\xi^2 \operatorname{tr}(\mathbf{H}(\mathbf{x}^*))$ and $\mathbf{x}^*$ is the minimizer. We assume a high enough accuracy $\xi \ll 1$ so that $\mu \ll 1$ and convergence actually occurs. Then, we have that $e_k \to e^* = \frac{\mu}{1-r}$ linearly with rate r. Indeed, for the limit we have $e_{k+1} = e^* = re^* + \mu \Rightarrow e^* = \frac{\mu}{1-r}$. For the rate, we have:

$$\frac{|e_{k+1} - e^*|}{|e_k - e^*|} = \frac{re_k + \mu - \frac{\mu}{1-r}}{e_k - \frac{\mu}{1-r}} = r.$$

This means that, when using approximate gradients, the sequence of objective function values (e_k) will seem to converge, but will do so to a value e^* that is larger than the optimal one E^*, and proportional to ξ^2. The iterates $\mathbf{x}_k$ will, of course, not converge but oscillate around $\mathbf{x}^*$. Fig. 5.4 illustrates this.

Fig. 5.5 shows the effect of different settings of the accuracy. We run EE (with $\lambda = 10^{-4}$) using gradient descent with FMM approximation for 4 000 points from the Swiss roll dataset. We fixed the step size to $\eta = 0.3$. First, we run the optimization for 100 iterations only (left two plots) and tried four different accuracy schedules: keep the accuracy at $p = 3$, at $p = 10$, or decrease it every 10 iterations from $p = 10$ to

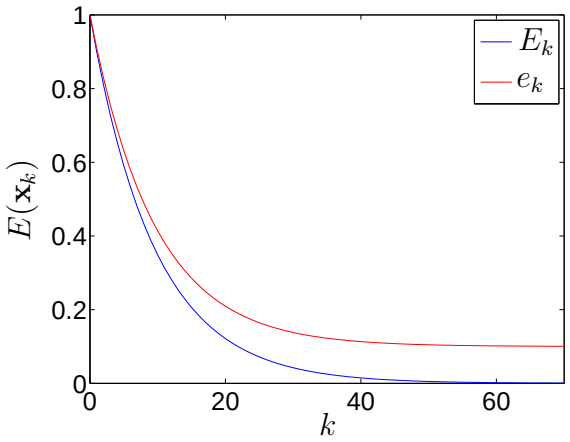

Figure 5.4: Illustration of the objective function sequences E_k and e_k for the optimization using exact and inexact gradients, respectively. We use $r = 0.9$, $\mu = 0.01$ and an initial $E_0 = e_0 = 1$. E_k converges to $E^* = 0$ while e_k converges to $e^* = 0.1$.

$p = 1$, or increase it from $p = 1$ to $p = 10$, respectively. Increasing the accuracy gives almost the same decrease per iteration as the approximation with $p = 10$ terms, however the runtime in the former case is faster. Both using a crude approximation ($p = 3$) and decreasing the accuracy does not achieve the same decrease in the objective function. Second, on the right plot, we used the same dataset, but now run it 10 times for 500 iterations with different $p = 1, \ldots, 10$ (blue lines on the plot). After each approximate step we also evaluate the exact gradient to see the difference between exact and approximate steps (black dashed lines). First, as the gradient approximation improves, the objective function decrease is greater. Second, the exact steps are always better than the approximate ones, which agrees with theorem 5.4.1. Third, the error between the exact and the approximate step becomes smaller as the approximation improves. Eventually, it becomes identical to the exact run of the method (red line).

5.5 Experiments

In all experiments, we reduce dimension to $d = 2$. First, we show that the performance of the methods matches the theoretical complexity. Fig. 5.6 shows the error and runtime of the exact method compared to those of BH and FGT as the number of points grows. We approximated the $S(\mathbf{x}_n)$ sum for uniformly distributed $\mathbf{x}_n \in \mathbb{R}^2$. The theory estimates

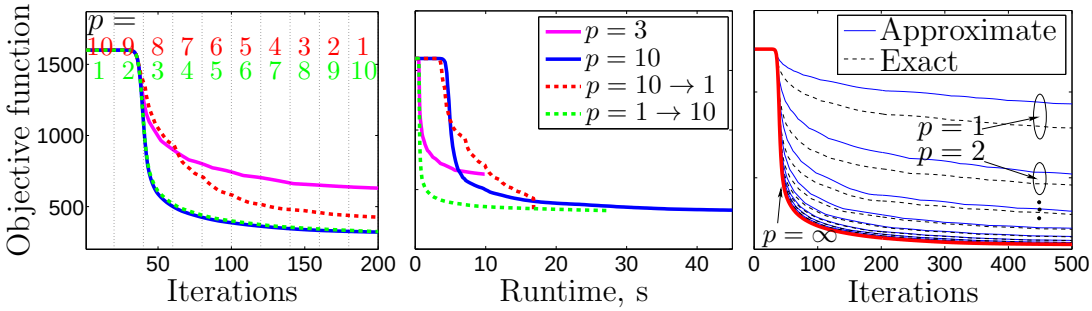

Figure 5.5: Minimization of 4 000 points from the swiss roll dataset using EE with gradient descent with different accuracy parameters. *Left two plots*: the number of iterations is limited for 200 iterations. The green and red numbers specify the p value change for dashed red and green curves. *Right plot*: we run FGT for $p = 1, \ldots, 10$ (blue lines) and run one exact step after each iteration of the FGT (black lines). Compare with the exact run (red line).

that the logarithm of the runtime t should be $\mathcal{O}(2 \log N)$ for exact methods, $\mathcal{O}(\log N + \log\log N)$ for BH and $\mathcal{O}(\log N)$ for FGT. Thus, in the log/log plot, the exact method and FGT should appear linearly with slopes 2 and 1 respectively and BH should appear almost linear. Indeed, the slope of the exact method is 2.02, the slope of FMM is 0.89 ± 0.08 (averaging over different p values) and the slope of BH is 1.17 ± 0.06 (averaging over θ), which as expected is slightly bigger than linear.

We compared the performance of the exact algorithms to FGT and BH for the EE algorithm (with $\lambda = 10^{-4}$) using gradient descent (GD), fixed point iteration (FP; Carreira-Perpiñán, 2010) and L-BFGS algorithms. For BH, we used our own C++ implementation; for FGT, our code was based on the implementation available at `www.cs.ubc.ca/~awll/nbody_methods.html`. We used fixed step sizes in the line search: $\eta = 0.1$ for GD, $\eta = 0.05$ for FP and $\eta = 0.01$ for L-BFGS. We tried several values and chose the ones that gave greatest steady decrease of the objective function, without frequent increases in the objective function. For the accuracy schedule, for BH we started with $\theta = 2$ and logarithmically decreased it to $\theta = 0.1$ for the first 100 iterations. For FGT, we started with $p = 1$ term in the local expansion and logarithmically increased it to $p = 10$ terms after the first 100 iterations. We kept the last approximation parameter fixed for subsequent iterations.

In the first experiment we used 60 000 digits from the MNIST handwritten dataset (fig. 5.7). We use a sparse affinity graph with 200 nearest neighbors for each point. We use entropic affinities with a perplexity $K = 50$. If we consider the decrease per iteration

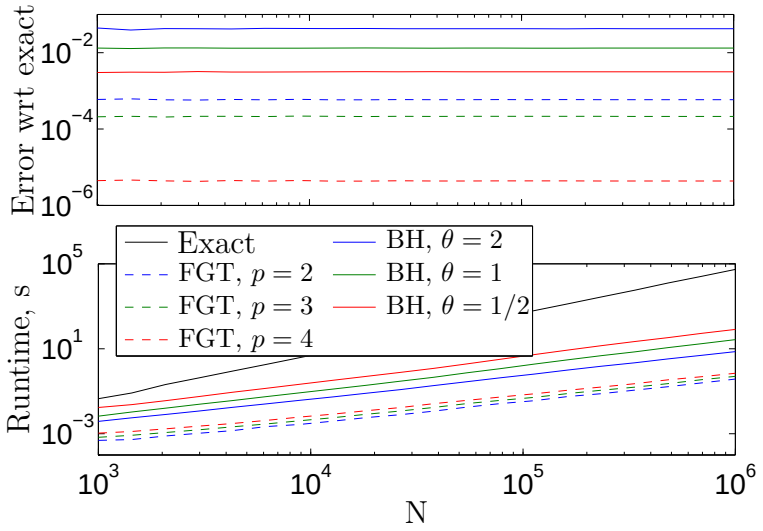

Figure 5.6: Error with respect to the exact computation (*top*) and runtime vs. the number of points (*bottom*).

disregarding the runtime (left plot), the methods go down in groups of three: one for GD, FP and L-BFGS respectively. This means the decrease per iteration is almost the same for the exact methods compared to the approximations, suggesting that the optimization follows a similar path. However, taking the runtime into account (right plot), we see a clear separation of FGT (green) from BH (blue) and the exact computation (red). Overall, BH is about 100× faster and FGT is about 400× faster than the exact method. Note the objective function values shown in the plot are not needed in the optimization and are computed exactly offline.

We used 1 020 000 points from infiniteMNIST dataset. For each digit the entropic affinities were constructed from the set of neighbors of the original digit and their deformations using perplexity 10. We run the optimization for 11 hours using GD, FP and L-BFGS for EE with FGT and BH approximations. Fig. 5.8 shows the objective function decrease per iteration and per second of runtime. Similarly to the previous experiment, BH and FGT show similar decrease per iteration (right plot), but FGT is much faster in terms of runtime (left plot). On average, we observe FGT being 5–7 times faster than BH. Below, we show the embedding of the digits after 3 hours of L-BFGS optimization using FGT and BH. The former looks much better than the latter, showing clearly the separation between digits. We also tried the exact computation on this dataset, but after 8 hours of optimization the algorithm only reached the second iteration.

We also generated 60 000 test digits and used the FGT approximation of the out-of-

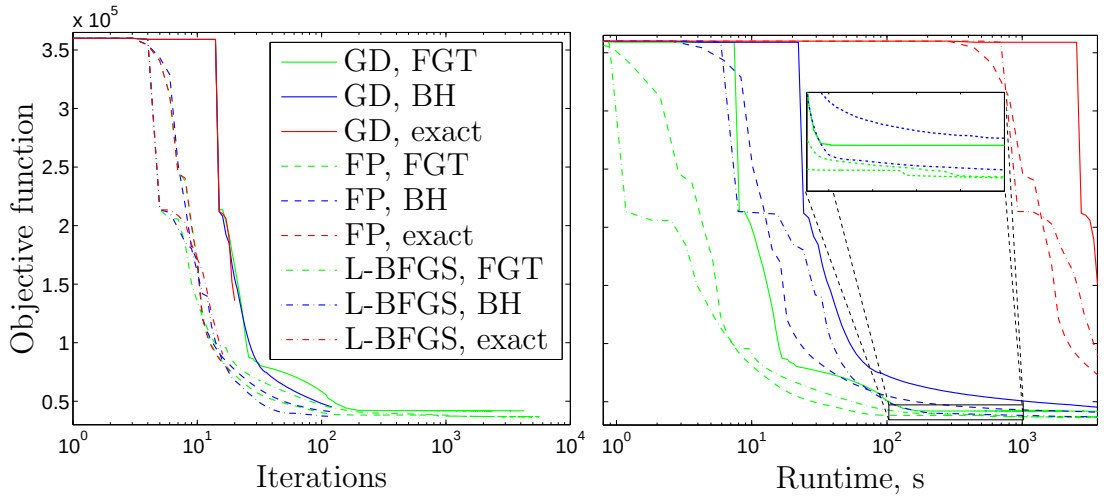

Figure 5.7: Speedup of the EE algorithm using BH and FGT for 60 000 MNIST digits using gradient descent (GD), fixed point iteration (FP) and L-BFGS. Learning curves as a function of the number of iterations (left) and runtime (right). The optimization follows almost the same path for the exact method and both approximations, however BH and FGT are about 100× and 400× faster respectively. Note the log plot in the X axis and the inset showing the BH and FGT curves.

sample mapping (5.9). We used the result of L-BFGS after 3 hours of optimization as the training data and initialized each test point to the training point that is closest to it. The resulting embedding is shown in fig. 5.9. We obtained the embedding of the test points in just 11 minutes and the embedding agrees with the structure of the training dataset.

5.6 Discussion

An interesting way to create a tree-based approximation that would run in linear time is to use a Barnes-Hut algorithm, but with a tree that has a maximum height. This way the complexity of building and querying the tree is linear, but the error is higher, since we don't query the leaved all the way to the bottom and for some of them the condition $l/D \leq \theta$ will be violated. This means that this new algorithm will most likely perform worse than FMM, that, for some dataset (see fig. 5.6), has both smaller computational error and faster runtime.

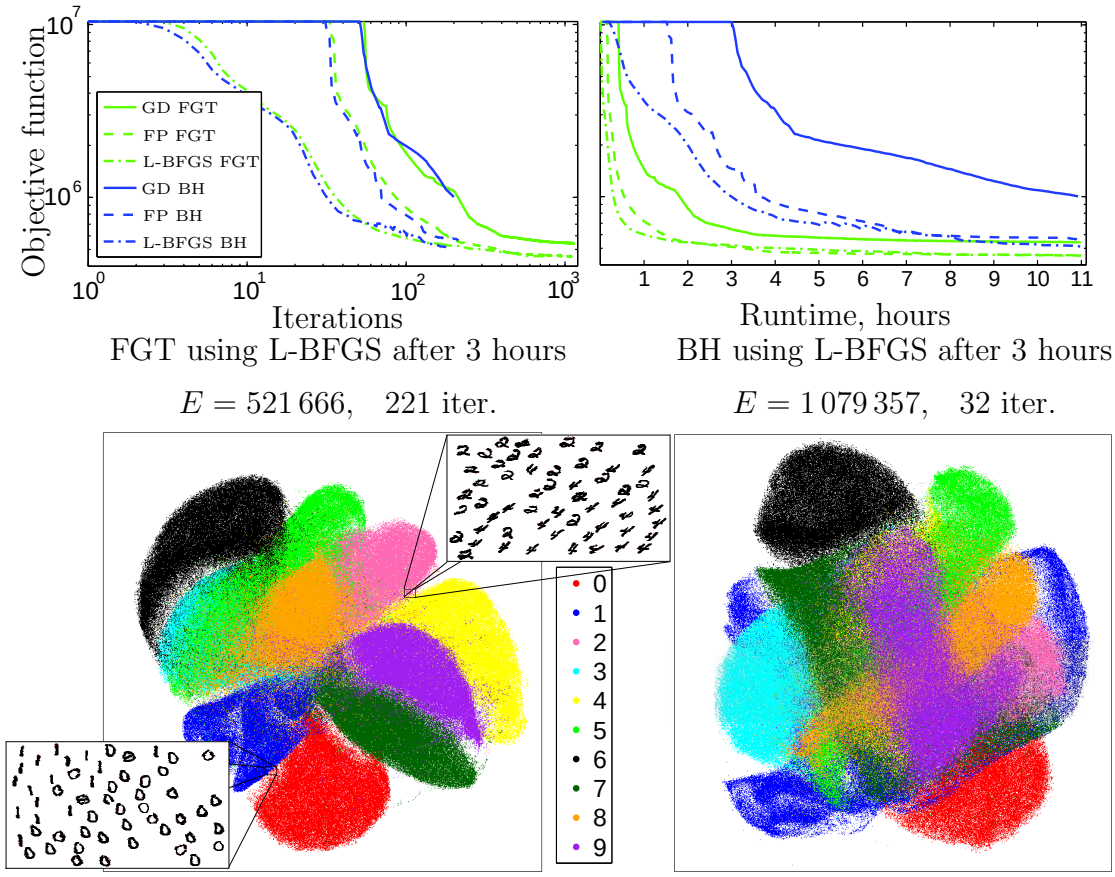

Figure 5.8: Embeddings of 1 020 000 digits from the infiniteMNIST dataset using the elastic embedding algorithm with FGT and BH, optimized with gradient descent (GD), fixed-point iteration (FP) and L-BFGS. *Top*: objective value change with respect to the number of iterations and runtime. *Bottom*: embedding of FGT and BH with L-BFGS after 3 hours of optimization. The inset shows that, in addition to separating digits, the embedding has also learned their orientation.

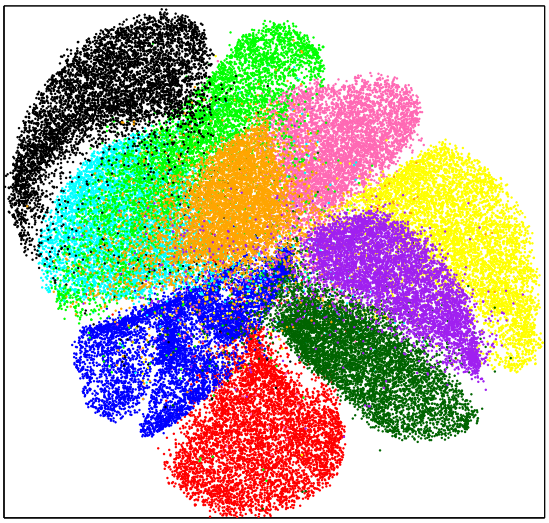

Figure 5.9: Out-of-sample projection of 60 000 digits using the embedding of 1 020 000 digits from the infiniteMNIST plotted on the bottom left of fig. 5.8.

5.7 Conclusion

We have shown that fast multipole methods, specifically the fast Gauss transform, are able to make the iterations of nonlinear embedding methods linear in the number of training points, thus attacking the main computational bottleneck of NLEs. This allows existing optimization methods to scale up to large datasets. In our case, we can achieve reasonable embeddings in hours for datasets of millions of points. We have also shown the FGT to be considerably better than the Barnes-Hut algorithm in this setting. Based on theoretical and experimental considerations, we show that starting at low accuracy and increasing it gradually further speeds up the optimization.

We think there is much room to design better algorithms that combine specific search directions, optimization techniques and N-body methods with specific NLE models. Another important direction for future research is to characterize the convergence of NLE optimization with inexact gradients obtained from N-body methods.

Chapter 6

Conclusions and Future Work Directions

6.1 Contributions

The main objective of this thesis was to extend the application of nonlinear dimensionality reduction methods to large-scale real world dataset. It has already been known that certain graph-based dimensionality reduction methods defined on a decently affinity matrix are able to give good results in showing the structure of complicated high-dimensional datasets. However, many things have prevented the applicability of these algorithms to large datasets.

As a start, for the affinity matrix, single bandwidth or multiple bandwidths set with a rule of thumb, often do not give satisfactory results since they don't take into account the whole distribution of distances in the dataset. Instead, in chapter 2 we proposed to use entropic affinities that were originally introduced by Hinton and Roweis (2003). Through the equation of entropy they connect the bandwidth, which is a spatial parameter, to the perplexity (effective number of neighbors) which is information-theoretical statistics and is much simpler to set. We analyze the properties of that equation and characterize its behavior, by showing that it is a well-defined function and giving explicit bounds for its implicitly defined value. Based on these properties, we have studied different algorithms for the computational problems involved: root-finding and ordering points for best initialization. One of the best and simplest choices is a Newton-based iteration, robustified with bisection steps, using a tree- or density-based order. This achieves

just above one iteration per data point on average, which is the optimally achievable performance.

However, even for a good affinity matrix, the applicability of existing nonlinear embedding algorithms were limited, due to slow and complicated optimization. In chapter 3 we showed that many of the methods (such as SNE, s-SNE, t-SNE and EE) can be described with a simple, yet quite general framework. We have uncovered the relation with spectral methods and the role of graph Laplacians in the gradient and Hessian, and derived several partial-Hessian optimization strategies. A thorough empirical evaluation shows that among several competitive strategies one emerges as particularly simple, generic and scalable, based on the Cholesky factors of the (sparsified) attractive Laplacian. This adds a negligible overhead to the computation of the gradient and objective function but improves existing algorithms by 1–2 orders of magnitude.

Next, in chapter 4 we addressed the computational bottleneck of the spectral methods, which is the expensive eigendecomposition of $N \times N$ matrix. Instead, we propose a reduced formulation of the original spectral problem that optimizes only over a small set of landmarks while retaining structure of the whole data. The algorithm is well defined theoretically and has better performance than more conventional Nyström method, allowing users to scale up applications to much bigger sizes. LLL also defines a natural out-of-sample extension that is cheaper and better than the Nyström method. We showed an extensive evaluation of LLL applied to two specific spectral methods algorithms: Laplacian eigenmaps and spectral clustering, we were able to achieve 10×–20× speed up with small approximation error.

Finally, in 5 we came back to the problem of nonlinear embedding methods and propose an approximation that is able to reduce the computational complexity of each iteration of the methods from quadratic to linear on the number of points in the dataset. The approximation is controlled by an accuracy parameter and we also provide a theoretical and empirical evidence that starting at low accuracy and increasing it gradually further speeds up the optimization.

All in all, the algorithms proposed in this thesis are able to accelerate most of the steps in the process of finding the solution of many best nonlinear dimensionally reduction algorithms such as spectral methods and nonlinear embedding methods. For all the algorithms we proposed, we were able to show one or two orders of magnitude acceleration, comparing to the methods proposed before us. Combining the methods together we can

get better affinity matrix fast (using entropic affinities), reduce drastically the number of iteration needed for convergence (using spectral direction optimization) and make those iterations fast (using FMM approximation). This will allow a practitioner to analyze a large-scale datasets consisting in millions of points on a modern size workstation in just a few hours.

6.2 Future Directions

Apart from noting some of the specific future direction in the end of each chapter, here we propose couple more general direction for future research that are applicable generally to the whole dissertation.

Nearest-neighbor graph. All the algorithms described in this thesis assume that the neighborhood graph is given to us. Finding it is an important problem by itself that we don't deal with in this dissertation.

In some of the applications, the neighborhood graph may be trivially given to us from the problem definition. For example, in the application of image or video segmentation in Chapter 4 we can define a neighborhood graph for this dataset by connecting points inside some predefined sliding window. Because the pixels of the image do not have drastically different intensity in their neighborhood (except for the edges maybe) this graph would be close to the exact one.

For other, more general, cases the exact construction of the k nearest neighbor graph would require the computation of all the distances between all N points, which takes $\mathcal{O}(N^2)$, and then taking k closest one for each point. The last operation naively costs $\mathcal{O}(N^2 \log N)$, but it might be accelerated by using partial quicksort to $\mathcal{O}(N(N+k \log k))$. However, as we see, if computed exactly the cost is still quadratic on the number of points. There exist many different algorithms that compute the nearest neighbor graph approximately, such as cover trees (Beygelzimer et al., 2006), divide and conquer approaches Chen et al. (2009), techniques based on locally sensitive hashing (Indyk, 2000; Liu et al., 2005) etc.

However, either the the computational cost or the approximation error of many of such approximation algorithms depends crucially on the dimensionality of the data (Shakhnarovich et al., 2006; He et al., 2012).

It remains an interesting question of how those methods affect for entropic affinities and the results of nonlinear dimensionality reduction in general.

Parallelization. One of the future direction is to adopt the methods for a use in a distributed system. Indeed, in a presence of really large data, even storing a sparse affinity matrix can be infeasible. Therefore distributing storage and computation is a natural extension for all the methods that we have described in this thesis. However, not all of the algorithms are readily available for use in parallel settings. The algorithm should the able to process data in batch and computations of those batches, at least some parts of the algorithm should be independent.

For the Entropic Affinities, adopting distributed computation requires only few modification to the algorithm. In particular, except for the initialization, every computation of β (including the bounds and using a root-finding algorithm) is independent from all the others and can be done separately. As for the initialization, we can split the points for each chunk according to the order of the points.

For LLL, we have converted the main bottleneck from the eigendecomposition to the matrix multiplication. The latter operation is much easier to parallelize and gives much better speed up (Chen et al., 2011). In addition, the construction of $\mathbf{Z}$ matrix consists of (1) computing the K_Z nearest landmarks for every point and (2) solving a linear system (4.8). Both of these steps are independent from one another given a location of (small) number of landmarks. Thus, the computation of $\mathbf{Z}$ is embarrassingly parallelizable.

There have been some work done on parallelizing the force-directed algorithms of graph drawing for the use on GPU (Frishman and Tal, 2007; Hachul and Jünger, 2005). Those methods use space partitioning to split the space into an independent regions that they solve for in parallel. The objective function force-directed algorithms have very similar form to the NLE methods and it would be an interesting direction to try to adopt the techniques presented in those papers.

Finally, we can also parallelize the N-Body computation. As we see from the form of the equations (5.1),(5.2),(5.3),(5.4) each of the expansions inside the source boxes can be computed individually. In addition, the interaction between the boxes (the outer sum) also decouples spatially due to the fast decay of the kernel. There exist few papers (Gumerov and Duraiswami, 2008; Stock and Gharakhani, 2008; Godiyal et al., 2009).that use these and other argument to apply FMM acceleration on GPU.

For the N-Body approximation it is important to understand more thoroughly how the

gradient approximation affects the change in the objective function. We think there is much room to design better algorithms that combine specific search directions, optimization techniques and N-body methods with specific NLE models. Another important direction for future research is to characterize the convergence of NLE optimization with inexact gradients obtained from N-body methods.

Appendix A

Datasets

A.1 COIL-20

COIL-20 dataset (Nene et al., 1996) contains images of rotation sequences of 20 physical objects every 5 degrees, each a grayscale image of 128×128 pixels, total $N = 720$ points in $D = 16\,384$ dimensions. The dataset is useful for dimensionality reduction because, on one side, it is very high-dimensional and, on the other, intrinsically, the images of each objects follow a simple circular trajectory from 0 to 360 degrees. Thus, while we don't know the exact form of the embedding, we know that good embedding should preserve this trajectory for every object and also keep each object separately from each other. In fig. A.1 we show an example of the rotation of two objects from that dataset.

A.2 MNIST handwritten digits dataset

This dataset (LeCun et al., 1998) contains 60 000 images of different handwritten digits. Each datapoint is a 28×28 grayscale image of a normalized and centered digit represented by a 784-dimensional vector. The dataset is useful, because it intrinsically has 10 different clusters and the 2D embedding is easy to interpret. In addition, the large number of digits in this dataset makes it non-trivial to many algorithm to find a solution. On the top row of fig. A.2 we show few examples of handwritten digits from MNIST dataset.

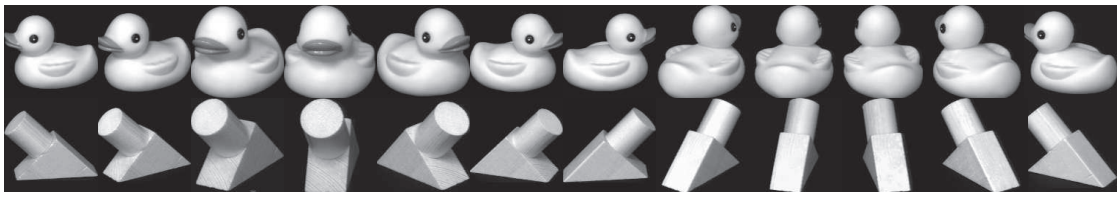

Figure A.1: Example of two objects from COIL-20 dataset and 12 out 72 images per object that show the rotational structure of the dataset.

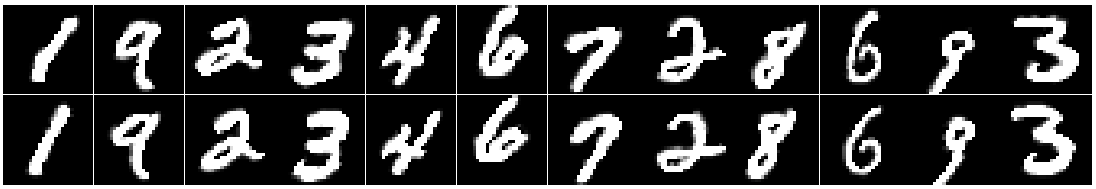

Figure A.2: Example of the elastic deformation to MNIST digits in the infiniteMNIST dataset. *Top*: original digit, *bottom:* one of the 16 deformations applied to each digit.

A.3 infiniteMNIST

For our large-scale experiments, sometimes even 60 000 points from the original MNIST is not enough. Therefore we have also used infiniteMNIST dataset (Loosli et al., 2007) where a new handwritten digits is generated using elastic deformations to the original MNIST dataset. For our purposes, we modified each digit from the original MNIST dataset 16 different times, resulting in 960 000 new digits. Together with the original dataset, this add up to a 1 020 000 digits. On the bottom of fig. A.2 we show one of 16 deformations that we have applied to the MNIST digit shown on the top row.

Bibliography

J. Barnes and P. Hut. A hierarchical $\mathcal{O}(N \log N)$ force-calculation algorithm. *Nature*, 324(4), 1986.

G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

G. Baudat and F. Anouar. Generalized discriminant analysis using a kernel approach. *Neural Computation*, 12(10):2385–2404, Oct. 2000.

B. J. C. Baxter and G. Roussos. A new error estimate of the fast Gauss transform. *SIAM J. Sci. Comput.*, 24(1), 257–259 2002.

O. Bchir and H. Frigui. Fuzzy relational kernel clustering with local scaling parameter learning. In *Proc. of the 2006 Machine Learning for Signal Processing (MLSP10)*, pages 289–294, Kittilä, Finland, Aug. 29 –Sept. 1 2010.

M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, June 2003.

M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, Nov. 2006.

Y. Bengio, O. Delalleau, N. Le Roux, J.-F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Computation*, 16 (10):2197–2219, Oct. 2004a.

Y. Bengio, J.-F. Paiement, P. Vincent, O. Delalleau, , N. Le Roux, and M. Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and spectral clustering.

In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 16. MIT Press, Cambridge, MA, 2004b.

A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In W. W. Cohen and A. Moore, editors, *Proc. of the 23rd Int. Conf. Machine Learning (ICML'06)*, pages 97–104, Pittsburgh, PA, June 25–29 2006.

I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Application*. Springer Series in Statistics. Springer-Verlag, Berlin, second edition, 2005.

C. Boutsidis, P. Drineas, and M. Magdon-Ismail. Near optimal column-based matrix reconstruction. In *Proc. of the 52th Annual Symposium on Foundations of Computer Science (FOCS 2011)*, pages 305–314, Palm Springs, RI, Oct. 23–25 2011.

R. P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, N.J., 1973.

R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. Other Titles in Applied Mathematics. SIAM Publ., revised reprint edition, 2009.

M. Á. Carreira-Perpiñán. Fast nonparametric clustering with Gaussian blurring mean-shift. In W. W. Cohen and A. Moore, editors, *Proc. of the 23rd Int. Conf. Machine Learning (ICML'06)*, pages 153–160, Pittsburgh, PA, June 25–29 2006.

M. Á. Carreira-Perpiñán. The elastic embedding algorithm for dimensionality reduction. In *Proc. of the 27th Int. Conf. Machine Learning (ICML 2010)*, Haifa, Israel, June 21–25 2010.

M. Á. Carreira-Perpiñán and Z. Lu. The Laplacian Eigenmaps Latent Variable Model. In M. Meilă and X. Shen, editors, *Proc. of the 11th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2007)*, pages 59–66, San Juan, Puerto Rico, Mar. 21–24 2007.

M. Á. Carreira-Perpiñán and R. S. Zemel. Proximity graphs for clustering and manifold learning. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 17, pages 225–232. MIT Press, Cambridge, MA, 2005.

J. Chen, H.-r. Fang, and Y. Saad. Fast approximate kNN graph construction for high dimensional data via recursive Lanczos bisection. *Journal of Machine Learning Research*, 10:1989–2012, Sept. 2009.

W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 33 (3):568–586, Mar. 2011.

X. Chen and D. Cai. Large scale spectral clustering with landmark-based representation. In *Proc. of the 25th National Conference on Artificial Intelligence (AAAI 2011)*, pages 313–318, San Francisco, CA, Aug. 7–11 2011.

J. Cook, I. Sutskever, A. Mnih, and G. Hinton. Visualizing similarity data with a mixture of maps. In M. Meilă and X. Shen, editors, *Proc. of the 11th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2007)*, San Juan, Puerto Rico, Mar. 21–24 2007.

T. F. Cox and M. A. A. Cox. *Multidimensional Scaling*. Chapman & Hall, London, New York, 1994.

N. de Freitas, Y. Wang, M. Mahdaviani, and D. Lang. Fast Krylov methods for n-body learning. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 18. MIT Press, Cambridge, MA, 2006.

V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 15, pages 721–728. MIT Press, Cambridge, MA, 2003.

V. de Silva and J. B. Tenenbaum. Sparse multidimensional scaling using landmark points. June 30 2004.

P. Drineas and M. W. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6: 2153–2175, Dec. 2005.

M. J. Er, S. Wu, J. Lu, and H. L. Toh. Face recognition with radial basis function (RBF) neural networks. *IEEE Trans. Neural Networks*, 13(3):697–710, May 2002.

R. Fergus, Y. Weiss, and A. Torralba. Semi-supervised learning in gigantic image collections. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 22, pages 522–530. MIT Press, Cambridge, MA, 2009.

W. Fong and E. Darve. The black-box fast multipole method. *J. Comp. Phys.*, 228(23): 8712–8725, 2009.

C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nyström method. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(2):214–225, Feb. 2004.

J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Mathematical Software*, 3(3):209–226, 1977.

Y. Frishman and A. Tal. Multi-level graph layout on the GPU. *IEEE Trans. Visualization and Computer Graphics*, 13(6):1310–1319, 2007.

T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, Nov. 1991.

W. Gander. On Halley's iteration method. *Amer. Math. Monthly*, 92(2):131–134, Feb. 1985.

S. Gao, I. W.-H. Tsang, L.-T. Chia, and P. Zhao. Local features are not lonely — Laplacian sparse coding for image classification. In *Proc. of the 2010 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'10)*, pages 3555–3561, San Francisco, CA, June 13–18 2010.

A. Globerson and S. T. Roweis. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems (NIPS)*, Jan 2006. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.7998&rep=rep1&type=pdf`.

A. Godiyal, J. Hoberock, M. Garland, and J. C. Hart. Rapid multipole graph drawing on the GPU. In D. Eppstein and E. R. Gansner, editors, *Proc. 17th Int. Symposium on Graph Drawing (GD 2009)*, pages 90–101, Chicago, IL, Sept. 22–25 2009.

J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems (NIPS)*, pages 513–520, 2004.

A. G. Gray and A. W. Moore. 'n-body' problems in statistical learning. In T. K. Leen, T. G. Diettrich, and V. Tresp, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pages 521–527. MIT Press, Cambridge, MA, 2001.

L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comp. Phys.*, 73(2):325–348, Dec. 1987.

L. Greengard and J. Strain. The fast Gauss transform. *SIAM J. Sci. Stat. Comput.*, 12 (1):79–94, Jan. 1991.

N. A. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *J. Comp. Phys.*, 227(18):8290–8313, 2008.

S. Hachul and M. Jünger. Large-graph layout with the fast multipole multilevel method. 2005.

N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

J. He, S. Kumar, and S.-F. Chang. On the difficulty of nearest neighbor search. In *Proc. of the 29th Int. Conf. Machine Learning (ICML 2012)*, Edinburgh, Scotland, June 26 – July 1 2012.

X. He and P. Niyogi. Locality preserving projections. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 16. MIT Press, Cambridge, MA, 2004.

G. Hinton and S. T. Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 15, pages 857–864. MIT Press, Cambridge, MA, 2003.

Y. Hu. Efficient and high-quality force-directed graph drawing. *The Mathematica Journal*, 10(1):37–71, 2005.

P. Indyk. Dimensionality reduction techniques for proximity problems. In *Proc. of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 371–378, San Francisco, CA, Jan. 9–11 2000.

S. Kaski and J. Peltonen. Informative discriminant analysis. In T. Fawcett and N. Mishra, editors, *Proc. of the 20th Int. Conf. Machine Learning (ICML'03)*, pages 329–336, Washington, DC, Aug. 21–24 2003.

B. Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5 (4):287—364, 2012.

S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the Nyström method. *Journal of Machine Learning Research*, 2012.

Ľ. Ladický and P. H. Torr. Locally linear support vector machines. In *Proc. of the 4th Int. Conf on Web Search and Data Mining (WSDM 2011)*, pages 985–992, Hong Kong, China, Feb. 9 – 12 2011.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov. 1998.

D. Lee, A. Gray, and A. Moore. Dual-tree fast Gauss transforms. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 18, pages 747–754. MIT Press, Cambridge, MA, 2006.

R. B. Lehoucq and D. C. Sorensen. Deflation techniques for an implicitly restarted Arnoldi iteration. *SIAM J. Matrix Anal. and Apps.*, 17(4):789–821, 1996.

T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 17, pages 825–832. MIT Press, Cambridge, MA, 2005.

W. Liu, J. He, and S.-F. Chang. Large graph construction for scalable semi-supervised learning. In *Proc. of the 27th Int. Conf. Machine Learning (ICML 2010)*, Haifa, Israel, June 21–25 2010.

W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proc. of the 4th Int. Conf on Web Search and Data Mining (WSDM 2011)*, pages 1–8, Hong Kong, China, Feb. 9 – 12 2011.

G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, Neural Information Processing, pages 301–320. MIT Press, 2007.

C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

A. Melman. Geometry and convergence of Euler's and Halley's methods. *SIAM Review*, 39(4):728–735, Dec. 1997.

R. Memisevic. Kernel information embeddings. In W. W. Cohen and A. Moore, editors, *Proc. of the 23rd Int. Conf. Machine Learning (ICML'06)*, pages 633–640, Pittsburgh, PA, June 25–29 2006.

R. Memisevic and G. Hinton. Improving dimensionality reduction with spectral gradient descent. *Neural Networks*, 18(5–6):702–710, June–July 2005.

S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K. Mullers. Fisher discriminant analysis with kernels. In Y. H. Hu and J. Larsen, editors, *Proc. of the 1999 IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing (NNSP'99)*, pages 41–48, Madison, WI, Aug. 23–25 1999.

A. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. In D. H. Fisher, editor, *Proc. of the 14th Int. Conf. Machine Learning (ICML'97)*, pages 236–244, Nashville, TN, July 6–12 1997.

S. A. Nene, S. K. Nayar, and H. Murase. Columbia object image library (COIL-20). Technical Report CUCS–005–96, Dept. of Computer Science, Columbia University, Feb. 1996.

A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Infor-*

mation Processing Systems (NIPS), volume 14, pages 849–856. MIT Press, Cambridge, MA, 2002.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

J. Peltonen and S. Kaski. Discriminative components of data. *IEEE Trans. Neural Networks*, 16(1):68–83, Jan. 5 2005.

J. Platt. FastMap, MetricMap, and landmark MDS are all Nyström algorithms. In R. G. Cowell and Z. Ghahramani, editors, *Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, pages 261–268, Barbados, Jan. 6–8 2005.

A. Quigley and P. Eades. FADE: Graph drawing, clustering, and visual abstraction. In J. Marks, editor, *Proc. 8th Int. Symposium on Graph Drawing (GD 2000)*, pages 197–210, Colonial Williamsburg, VA, Sept. 20–23 2000.

V. C. Raykar. The fast Gauss transform with all the proofs. Apr. 2006.

V. C. Raykar and R. Duraiswami. Fast optimal bandwidth selection for kernel density estimation. In *Proc. of the 2006 SIAM Int. Conf. Data Mining (SDM 2006)*, pages 524–528, Bethesda, MD, Apr. 20–22 2006a.

V. C. Raykar and R. Duraiswami. The improved fast Gauss transform with applications to machine learning. In L. Bottou, O. Chapelle, D. Decoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, Cambridge, MA, 2006b.

C. J. F. Ridders. A new algorithm for computing a single root of a real continuous function. *IEEE Trans. Circuits and Systems*, 26(11):979–980, Nov. 1979.

S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, Dec. 22 2000.

J. K. Salmon and M. S. Warren. Skeletons from the treecode closet. *J. Comp. Phys.*, 111(1):136–155, 1994.

H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

L. K. Saul and S. T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, June 2003.

L. K. Saul, K. Q. Weinberger, J. H. Ham, F. Sha, and D. D. Lee. Spectral methods for dimensionality reduction. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-Supervised Learning*, Adaptive Computation and Machine Learning Series, chapter 16, pages 293–308. MIT Press, 2006.

T. R. Scavo and J. B. Thoo. On the geometry of Halley's method. *Amer. Math. Monthly*, 102(5):417–426, May 1995.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, July 1998.

G. Shakhnarovich, P. Indyk, and T. Darrell, editors. *Nearest-Neighbor Methods in Learning and Vision*. Neural Information Processing. MIT Press, Cambridge, MA, 2006.

Y. Shen, A. Y. Ng, and M. Seeger. Fast Gaussian process regression using KD-trees. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 18. Citeseer, 2006.

J. Shi and J. Malik. Motion segmentation and tracking using normalized cuts. In *Proc. 6th Int. Conf. Computer Vision (ICCV'98)*, Bombay, India, Jan. 4–7 1998.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):888–905, Aug. 2000.

J. C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. John Wiley & Sons, 2003.

M. J. Stock and A. Gharakhani. Toward efficient GPU-accelerated n-body simulations. *46th AIAA Aerospace Sciences Meeting And Exhibit*, 608, 2008.

A. Talwalkar, S. Kumar, and H. Rowley. Large-scale manifold learning. In *Proc. of the 2008 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'08)*, Anchorage, AK, June 23–28 2008.

J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, Dec. 22 2000.

J. F. Traub. *Iterative Methods for the Solution of Equations*. Prentice-Hall, second edition, 1982.

L. J. van der Maaten and G. E. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, November 2008.

L. J. P. van der Maaten. Fast optimization for t-SNE. Workshop on Challenges in Data Visualization at NIPS 2010, 2010.

L. J. P. van der Maaten. Barnes-Hut-SNE. Jan. 15 2013.

J. C. van Gemert, J.-M. Geusebroek, C. J. Veenman, and A. W. Smeulders. Kernel codebooks for scene categorization. In D. Forsyth, P. Torr, and A. Zisserman, editors, *Proc. 10th European Conf. Computer Vision (ECCV'08)*, pages 696–709, Marseille, France, Oct. 13–16 2008.

J. Venna, J. Peltonen, K. Nybo, H. Aidos, and S. Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *Journal of Machine Learning Research*, 11:451–490, Feb. 2010.

M. Vladymyrov and M. Á. Carreira-Perpiñán. Partial-Hessian strategies for fast learning of nonlinear embeddings. In *Proc. of the 29th Int. Conf. Machine Learning (ICML 2012)*, pages 345–352, Edinburgh, Scotland, June 26 – July 1 2012.

M. Vladymyrov and M. Á. Carreira-Perpiñán. Entropic affinities: Properties and efficient numerical computation. In *Proc. of the 230h Int. Conf. Machine Learning (ICML 2013)*, pages 477–485, Atlanta, GA, 2013a.

M. Vladymyrov and M. Á. Carreira-Perpiñán. Locally linear landmarks for large-scale manifold learning. In H. Blockeel, K. Kersting, S. Nijssen, and F. Zelezný, editors, *Proc. of the 24nd European Conf. Machine Learning (ECML–13)*, Prague, Czech Republic, Sept. 23–27 2013b.

M. Vladymyrov and M. Á. Carreira-Perpiñán. Linear-time training of nonlinear low-dimensional embeddings. In *Proc. of the 17th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2014)*, pages 968–977, Reykjavik, Iceland, Apr. 22–25 2014.

U. von Luxburg, A. Radl, and M. Hein. Getting lost in space: Large sample analysis of the resistance distance. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 23, pages 2622–2630. MIT Press, Cambridge, MA, 2010.

X. Wan and G. E. Karniadakis. A sharp error estimate for the fast Gauss transform. *J. Comp. Phys.*, 219(1):7–12, 2006.

J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *Proc. of the 2010 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'10)*, pages 3360–3367, San Francisco, CA, June 13–18 2010.

K. Weinberger, B. Packer, and L. Saul. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In R. G. Cowell and Z. Ghahramani, editors, *Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, pages 381–388, Barbados, Jan. 6–8 2005.

K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proc. of the 2004 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'04)*, pages 988–995, Washington, DC, June 27 – July 2 2004.

K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *Int. J. Computer Vision*, 70(1):77–90, Oct. 2006.

K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, Feb. 2009. URL `http://www.cse.wustl.edu/~kilian/code/code.html`.

C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T. K. Leen, T. G. Diettrich, and V. Tresp, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pages 682–688. MIT Press, Cambridge, MA, 2001.

C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis. Improved fast Gauss transform and efficient kernel density estimation. In *Proc. 9th Int. Conf. Computer Vision (ICCV'03)*, pages 464–471, Nice, France, Oct. 14–17 2003.

Z. Yang, J. Peltonen, and S. Kaski. Scalable optimization for neighbor embedding for visualization. In *Proc. of the 230h Int. Conf. Machine Learning (ICML 2013)*, pages 127–135, Atlanta, GA, 2013.

L. Ying, G. Biros, and D. Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *J. Comp. Phys.*, 196(2):591–626, 2004.

K. Yu, T. Zhang, and Y. Gong. Nonlinear learning using local coordinate coding. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 22. MIT Press, Cambridge, MA, 2009.

L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1601–1608, 2004.

Z. Zhang and J. Wang. MLLE: Modified locally linear embedding using multiple weights. In B. Schölkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 19, pages 1593–1600. MIT Press, Cambridge, MA, 2007.

D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems (NIPS)*, pages 321–328, 2003.

X. Zhou, N. Cui, Z. Li, F. Liang, and T. S. Huang. Hierarchical Gaussianization for image classification. In *Proc. 12th Int. Conf. Computer Vision (ICCV'09)*, pages 1971–1977, Kyoto, Japan, Sept. 29 – Oct. 2 2009.