Linear-time Training of Nonlinear Low-Dimensional Embeddings

Max Vladymyrov **Miguel Á. Carreira-Perpiñán**
Electrical Engineering and Computer Science, School of Engineering, University of California, Merced

Abstract

Nonlinear embeddings such as stochastic neighbor embedding or the elastic embedding achieve better results than spectral methods but require an expensive, nonconvex optimization, where the objective function and gradient are quadratic on the sample size. We address this bottleneck 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.

Dimensionality reduction algorithms are 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. We focus on the class of *embedding algorithms based on pairwise affinities*. Here, a dataset consisting of N objects is represented by a weighted graph where each object is a vertex and weighted edges indicate similarity or distance between objects. *Nonlinear embeddings (NLE)*, such as stochastic neighbor embedding (SNE; Hinton and Roweis, 2003), t-SNE (van der Maaten and Hinton, 2008) or the elastic embedding (EE; Carreira-Perpiñán, 2010), are particularly desirable because they produce embeddings that are much better than those of linear (such as PCA) or spectral methods (such as Laplacian eigenmaps or LLE; Belkin and Niyogi, 2003; Roweis and Saul, 2000), especially when the high-dimensional data have a complex cluster and manifold structure. Also, given the weighted graph, the runtime of nonlinear (and spectral) embedding algorithms is independent of the input dimensionality, and so they can handle very high-dimensional objects, such as images.

Appearing in Proceedings of the 17^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33. Copyright 2014 by the authors.

The fundamental disadvantage of NLE is their difficult, slow optimization, which has prevented their widespread use (particularly in exploratory data analysis, where interactivity is important). Although recent advances in the optimization algorithms, such as the spectral direction of Vladymyrov and Carreira-Perpiñán (2012), have significantly reduced the number of iterations, each iteration is still quadratic on the number of points N, and this does not scale to large datasets. 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. As pointed out by Vladymyrov and Carreira-Perpiñán (2012), a convenient way to break the quadratic cost is to approximate 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 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.

The contributions of this paper are as follows. We review existing NLE methods and N-body methods that can be applicable to them. We then propose a linear-time algorithm based on FMMs and compare it to the Barnes-Hut approximation and the exact computation. Finally, 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. This enables us to handle million-point datasets in three hours' runtime.

1 Review: Embeddings and N-body Methods

1.1 Review of Nonlinear Embedding (NLE) Methods

Many NLE methods can be written in the following generic form (Vladymyrov and Carreira-Perpiñán, 2012). Given a symmetric nonnegative affinity[1] matrix $\mathbf{W}$ defined for a set of input high-dimensional data points $\mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_N)$, we find a $d \times N$ matrix of low-dimensional points $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ by minimizing the objective function

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}), \qquad \lambda \geq 0, \tag{1}$$

where E^+ is an "attractive" term that pulls points together in $\mathbf{X}$-space if they were close in the original $\mathbf{Y}$-space, and E^- is a "repulsive" term that drives all points apart from each other. Optimal embeddings balance both forces. Special NLE cases include EE[2]:

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

and s-SNE and t-SNE[3]:

$$E(\mathbf{X}, \lambda) = \textstyle\sum_{n,m=1}^N w_{nm} \log K(\left\| \mathbf{x}_n - \mathbf{x}_m \right\|^2) + \lambda \log \left(\textstyle\sum_{n,m=1}^N K(\left\| \mathbf{x}_n - \mathbf{x}_m \right\|^2) \right), \tag{3}$$

where K is a Gaussian or a Student's t kernel, respectively. The gradient of (1) is $\nabla E(\mathbf{X}) = 4\mathbf{X}(\mathbf{L} - \lambda\widetilde{\mathbf{L}})$ with graph Laplacians defined as:

$$\mathbf{L} = \operatorname{diag}\left(\textstyle\sum_{n=1}^N w_{nm}\right) - \mathbf{W} \qquad \widetilde{\mathbf{L}} = \operatorname{diag}\left(\textstyle\sum_{n=1}^N \widetilde{w}_{nm}\right) - \widetilde{\mathbf{W}}$$

where the weights $\widetilde{\mathbf{W}}$ depend on the embedding $\mathbf{X}$ and have elements $\widetilde{w}_{nm}$ given by:

$$\text{EE: } e^{-\left\| \mathbf{x}_n - \mathbf{x}_m \right\|^2} \qquad \text{s-SNE: } \frac{\exp(- \left\| \mathbf{x}_n - \mathbf{x}_m \right\|^2)}{\sum_{k,l=1}^N \exp(- \left\| \mathbf{x}_k - \mathbf{x}_l \right\|^2)}$$

$$t\text{-SNE: } \frac{(1 + \left\| \mathbf{x}_n - \mathbf{x}_m \right\|^2)^{-2}}{\sum_{k,l=1}^N (1 + \left\| \mathbf{x}_k - \mathbf{x}_l \right\|^2)^{-1}}.$$

1.2 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.

[1]Computing the affinities efficiently from the input data is an important problem that we do not consider here. At present, approximate nearest neighbor methods are one possible solution.

[2]EE additionally has negative affinities w_{nm}^- inside the repulsive term. In this paper we take them equal to 1.

[3]The original, equivalent formulation of s-SNE and t-SNE was given in terms of KL divergences and used $\lambda = 1$.

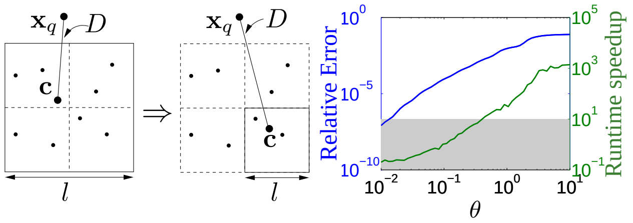

Figure 1: *Left*: for $l/D > \theta$, the cell is subdivided into smaller subcells. Otherwise, the interaction is computed approximately. *Right*: speedup and relative error for different values of θ. The gray area corresponds to the region with no speedup. Notice the log/log plot.

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 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. 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. 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).

Fast Multipole Methods (FMM) These 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

$$Q(\mathbf{x}_n) = \textstyle\sum_{m=1}^{N} q_m \exp(-\left\|(\mathbf{x}_n - \mathbf{x}_m)/\sigma\right\|^2) \quad (4)$$

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

In the FGT, we normalize the dataset to lie in the unit box $[0, 1]^d$ and subdivide this into smaller boxes parallel to the axes of side $\sqrt{2}\sigma r$ (for some $r \leq 1/2$). To compute the sum (4), we write each Gaussian interaction between a source point $\mathbf{s}$ and a target point $\mathbf{t}$ as a Hermite expansion around the center $\mathbf{s}_{\mathcal{B}}$ of the box $\mathcal{B}$ that $\mathbf{s}$ belongs to[4]:

$$e^{-\|(\mathbf{t}-\mathbf{s})/\sigma\|^2} = \sum_{\boldsymbol{\alpha} \geq 0} \frac{1}{\boldsymbol{\alpha}!}\, h_{\boldsymbol{\alpha}}\Big(\frac{\mathbf{s}-\mathbf{s}_{\mathcal{B}}}{\sigma}\Big)\Big(\frac{\mathbf{t}-\mathbf{s}_{\mathcal{B}}}{\sigma}\Big)^{\boldsymbol{\alpha}}, \quad (5)$$

where $h_n(t) = e^{-t^2} H_n(t)$ are Hermite functions with Hermite polynomials $H_n(t)$. The algorithm decouples the evaluation of the exponent into two separate computations: one between $\mathbf{s}$ and $\mathbf{s}_{\mathcal{B}}$, and another between $\mathbf{s}_{\mathcal{B}}$ and $\mathbf{t}$. Analogously, instead of a Hermite expansion around the center

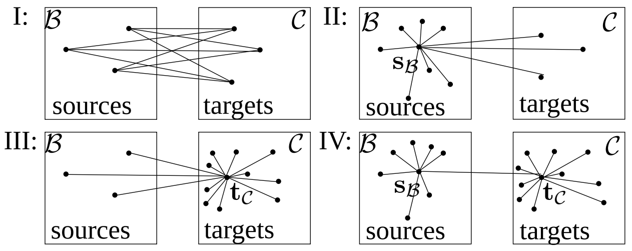

Figure 2: Different FGT approximations. I: exact interaction (4) (few points in both boxes); II: expansion around $\mathbf{s}_{\mathcal{B}}$ (many source points); III: expansion around $\mathbf{t}_{\mathcal{C}}$ (many target points); IV: expansion around $\mathbf{s}_{\mathcal{B}}$, then Taylor expansion to the Hermite functions (many points in both boxes).

$\mathbf{s}_{\mathcal{B}}$, we can use a Taylor expansion around the center $\mathbf{t}_{\mathcal{C}}$ of the box $\mathcal{C}$ the target $\mathbf{t}$ belongs to. Finally, we can further approximate the Hermite expansion by also expanding the term $h_{\boldsymbol{\alpha}}(t)$ in (5). The approximation comes from truncating the series (5) to terms of up to order p, which reduces the cost from $\mathcal{O}(N^2)$ to $\mathcal{O}(Np^d)$ (since there are p terms per dimension). Strain (1991) has also extended the FGT to the case of variable bandwidths for source and target points. Detailed approximation formulas appear in the supplementary material.

The choice of whether to use the direct evaluation or to approximate it with a series, and which series to use, depends on the number of points in a given box (see fig. 2). Greengard and Strain (1991) propose the following algorithm: if the number of points is smaller than a certain threshold M_0, the interaction is computed exactly between all the points in that box. Otherwise, it is computed using the centers $\mathbf{s}_{\mathcal{B}}$ and/or $\mathbf{t}_{\mathcal{C}}$. 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, 2007), or otherwise the overhead is so large that computing the exact interaction is actually cheaper. In this paper, we focus on $d \leq 3$ and the plain FGT with parameters $r = 1/2$, $M_0 = 5$, $K = 4$, 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.

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

While in this paper 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.

1.3 Work on Fast Training of Nonlinear Embeddings

Until recently, NLEs were usually trained with variations of gradient descent that were slow and limited the applicability of the methods to very small datasets. Fixed-point iteration algorithms (Carreira-Perpiñán, 2010) can improve this by an order of magnitude. The currently fastest algorithm is the spectral direction of Vladymyrov and Carreira-Perpiñán (2012), which uses a sparse positive definite Hessian approximation to "bend" the true gradient with the curvature of the spectral part of the objective, at a negligible overhead. This is 10–100$\times$ faster than previous methods and beats standard large-scale methods such as conjugate gradients and L-BFGS. However, each iteration of all these methods scales quadratically on N.

N-body problems also 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. (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.

2 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 (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:

$$S(\mathbf{x}_n) = \textstyle\sum_{m=1}^N K(\|\mathbf{x}_n - \mathbf{x}_m\|^2) \tag{6a}$$

$$S^x(\mathbf{x}_n) = \textstyle\sum_{m=1}^N \mathbf{x}_m K(\|\mathbf{x}_n - \mathbf{x}_m\|^2). \tag{6b}$$

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

$$E(\mathbf{X}) = \textstyle\sum_{n,m=1}^N w_{nm} \|\mathbf{x}_n - \mathbf{x}_m\|^2 + \lambda \sum_{m=1}^N f(S(\mathbf{x}_m))$$

$$\tfrac{\partial E}{\partial \mathbf{X}} = 4\mathbf{X}\mathbf{L} - 4\lambda Z(\mathbf{X})\mathbf{X} \operatorname{diag}(S(\mathbf{X})) + 4\lambda Z(\mathbf{X}) S^x(\mathbf{x}_n)$$

where $f(x) = \log x$, $Z(\mathbf{X}) = 1/\sum_{n=1}^N S(\mathbf{x}_n)$ for s-SNE and $f(x) = 1$, $Z(\mathbf{X}) = 1$ 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 (6). We get

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

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

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 (4) 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 1.2).

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:

$$\min_{\mathbf{x}} E'(\mathbf{x},\mathbf{y}) = 2\sum_{n=1}^{N} \big(w(\mathbf{y},\mathbf{y}_n) \left\|\mathbf{x}-\mathbf{x}_n\right\|^2 + \lambda \exp\big(- \left\|\mathbf{x}-\mathbf{x}_n\right\|^2 \big) \big). \quad (7)$$

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 3.

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, in this paper we use a constant step size η, chosen sufficiently small, which is simpler.

3 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 $\mathbf{x} \in \mathbb{R}^d$, 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. 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. 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 3.1. *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 $\mu_\Delta(\mathbf{x}) = \langle \Delta E(\mathbf{x})\rangle$, $v_\Delta(\mathbf{x}) = \big\langle (\Delta E(\mathbf{x}) - \langle \Delta E(\mathbf{x})\rangle)^2 \big\rangle$, $\mu_\delta(\mathbf{x}) = \langle \delta E(\mathbf{x})\rangle$ and $v_\delta(\mathbf{x}) = \big\langle (\delta E(\mathbf{x}) - \langle \delta E(\mathbf{x})\rangle)^2 \big\rangle$ 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 $\mathbf{J}_\mathbf{H}(\mathbf{x})$ the $d \times d$ Jacobian matrix of the Hessian diagonal elements, i.e., $(\mathbf{J}_\mathbf{H}(\mathbf{x}))_{ij} = \partial h_{ii}/\partial x_j = \partial^3 E(\mathbf{x})/\partial x_i^2 \partial x_j$. Then, the expected errors and their variances satisfy, $\forall \mathbf{x} \in \mathbb{R}^d$:*

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

$$v_\Delta(\mathbf{x}) = \xi^2 \left\|\mathbf{g}(\mathbf{x})\right\|^2 + \xi^4 \left(\tfrac{1}{2} \left\|\mathbf{H}(\mathbf{x})\right\|_F^2 + \mathbf{1}^T \mathbf{J}_\mathbf{H}(\mathbf{x})\mathbf{g}(\mathbf{x}) \right) + \mathcal{O}(\xi^6)$$

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

If $\left\|\mathbf{H}(\mathbf{x})\right\|_2 \le M\ \forall \mathbf{x} \in \mathbb{R}^d$ for some $M > 0$, then $\forall \mathbf{x} \in \mathbb{R}^d$ $|\mu_\Delta| \le \frac{1}{2}\xi^2 dM$.

The proof is given in the supplementary material. While this noise model is probably too simple to make quantitative predictions, it does give important qualitative predictions: (1) adding noise will be beneficial only where the mean curvature $\frac{1}{d}\operatorname{tr}\big(\nabla^2 E(\mathbf{x})\big)$ is negative; (2) when the mean curvature is positive, the lower the accuracy the worse the optimization; (3) $\mu_\Delta / \operatorname{tr}\big(\nabla^2 E(\mathbf{x})\big)$ should take an approximately constant value over iterates which is related to the accuracy level; and (4) $\Delta E(\mathbf{x})$ will vary widely at the beginning of the optimization and become approximately constant and equal to $\frac{1}{2}\xi^2 \operatorname{tr}(\mathbf{H}(\mathbf{x}))$ near a minimizer. This gives suggestions as to how to tune the accuracy (θ or p). Let us assume that the optimization algorithm decreases the objective function, at least on average. Thus, we expect that the early iterates will move through a region that may have negative or positive mean curvature, but eventually they will move through a region of positive mean curvature, as they approach a minimizer. A higher

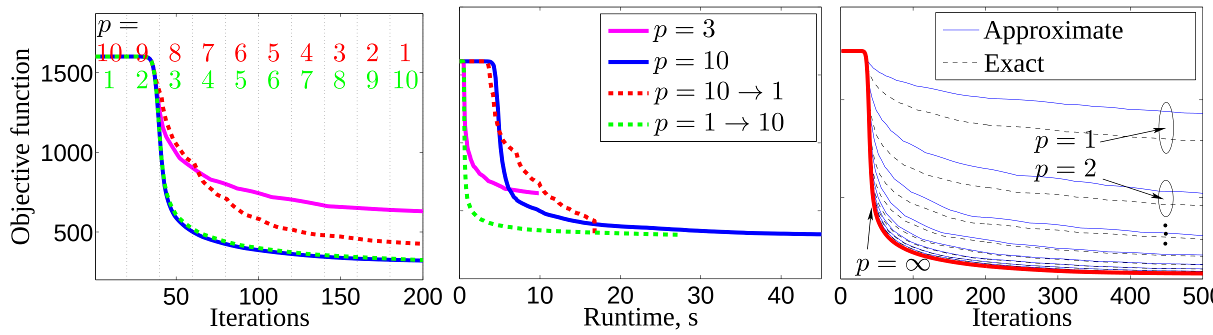

Figure 3: Minimization of $4\,000$ points from the Swissroll dataset using EE with gradient descent with different accuracy parameters. *Left two plots*: the number of iterations is limited for 200 iterations. *Right plot*: we run FGT for $p = 1, \dots, 10$ (blue lines) and run one exact step after each iteration of the FGT (black lines). Compare with the exact run (red line).

accuracy will be necessary in the later stages of the optimization. As for the early stages, we can be more specific by looking at the Hessian trace for some embedding models (see Vladymyrov and Carreira-Perpiñán, 2012 for the exact formulas):

- EE: $\operatorname{tr}\left(\nabla^2 E(\mathbf{x})\right) = 4d \operatorname{tr}(\mathbf{L})$, where $\mathbf{L}$ is the $N \times N$ graph Laplacian corresponding to the affinities in the high-dimensional space and d is the dimension of the low-dimensional space.

- s-SNE, t-SNE: $\operatorname{tr}\left(\nabla^2 E(\mathbf{x})\right) = 4d \operatorname{tr}(\mathbf{L}) - 16\lambda \left\|\mathbf{X}\mathbf{L}^q\right\|_F^2$, where $\mathbf{L}^q$ is a $N \times N$ graph Laplacian corresponding to the affinities learned in the low-dimensional space.

For the graph Laplacian in the input space, we have $\operatorname{tr}(\mathbf{L}) = \sum_{n \neq m}^N w_{nm}$, which is a positive constant. Thus, the mean curvature is always positive for EE, so we do not expect the noise to help anywhere. For s-SNE and t-SNE, the mean curvature can be negative if $\left\|\mathbf{X}\mathbf{L}^q\right\|_F^2$ is large enough, but this will likely not happen if, as is commonly done, one initializes $\mathbf{X}$ from small values. In summary, it seems unlikely that the mean curvature will be negative during the optimization, and therefore the inexact steps caused by the BH or FMM methods will reduce the objective less than exact steps on average. However, it is likely that the mean curvature will become more positive as the optimization progresses, which suggests starting with relatively low accuracy and increasing it progressively. It still may make sense to try to benefit from the noise whenever the mean curvature does become negative. Since the Hessian trace for s-SNE and t-SNE can be computed in linear time in the number of parameters Nd in the embedding $\mathbf{X}$, one could detect when it is negative and use very low accuracy in the gradient evaluations.

Practically, there are two more reasons why it is beneficial to start with low accuracy and increase it further on. First, it is cheaper to compute the low-accuracy value, so the runtime is smaller. Second, inexact gradient values may increase the value of the objective function at some iterations. Thus, using the accuracy as an inverse temperature may give our algorithm the advantages of simulated annealing: a low accuracy in the beginning facilitates some degree of wandering in parameter space, which may help to identify good optima. As we proceed in the optimization, the accuracy should be increased to reduce the wandering behavior and eventually converge.

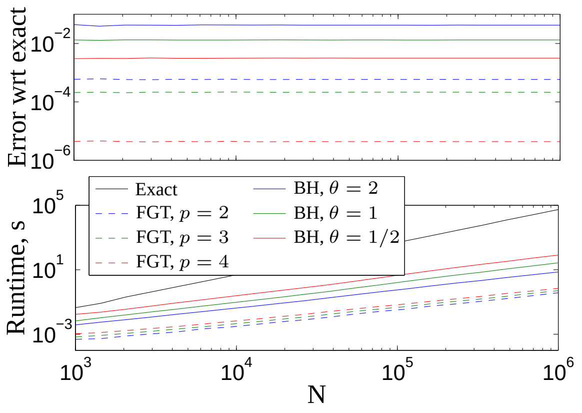

Figure 4: Error with respect to the exact computation (*top*) and runtime vs. the number of points (*bottom*).

Theorem 3.1 tries to be as independent as possible of the particular approximation method (FMM, BH, etc.) and NLE (SNE, t-SNE, EE, etc.). The FGT bounds of Baxter and Roussos (2002) and Wan and Karniadakis (2006) only apply to Gaussian sums with the FGT method and are *independent of the iterate* $\mathbf{x}$ (they only depend on the number of terms p, dimension of latent space d and box width r). Hence, these FMM bounds can be coarse, and do not distinguish between early and late stages of the optimization, so they do not help to design adaptive schedules for the accuracy level.

Fig. 3 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 $p = 1$, or increase it from $p = 1$ to $p = 10$, respectively. Increasing the accuracy gives almost

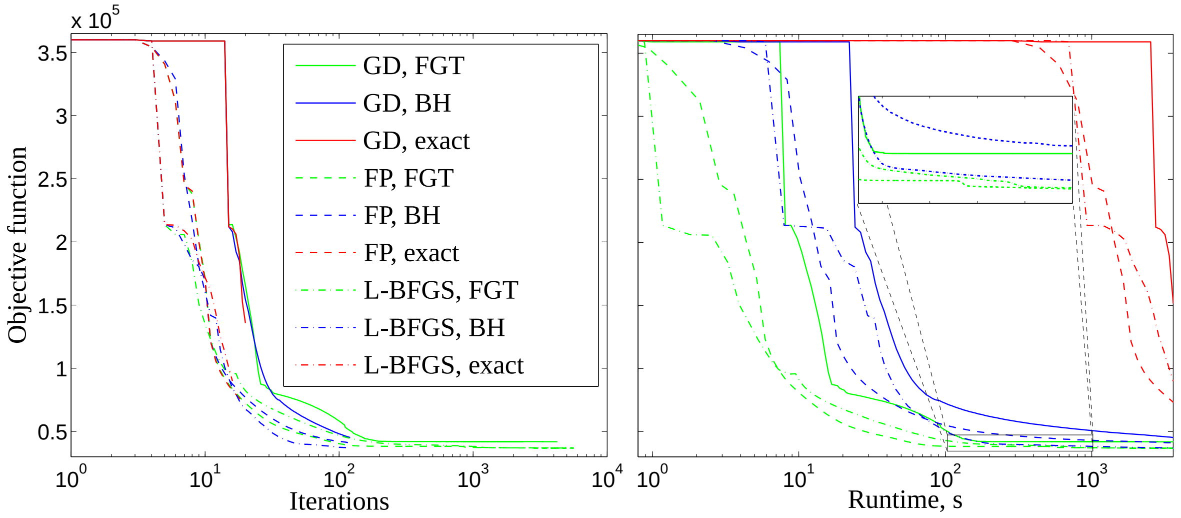

Figure 5: Speedup of the EE algorithm using BH and FGT for 60 000 MNIST digits using gradient descent (GD), fixed point iteration (FP) and L-BFGS. Learning curves as a function of the number of iterations (left) and runtime (right). The optimization follows almost the same path for the exact method and both approximations, however BH and FGT are about $100\times$ and $400\times$ faster respectively. Note the log plot in the X axis and the inset showing the BH and FGT curves.

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 $p = 1, \dots, 10$ (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 3.1. 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).

4 Experiments

In all experiments, we reduce dimension to $d = 2$. First, we show that the performance of the methods matches the theoretical complexity. Fig. 4 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 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). We use a sparse affinity graph with 200 nearest neighbors for each point. We use entropic affinities (Hinton and Roweis, 2003; Vladymyrov and Carreira-Perpiñán, 2013) with a perplexity (effective number of neighbors) of 50, that is, Gaussian affinities where the local bandwidth of each point is set so it defines a distribution over its neighbors having an entropy of $\log(50)$. If we consider the decrease per iteration 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 $100\times$ faster and FGT is about $400\times$ faster than the exact method. Note the

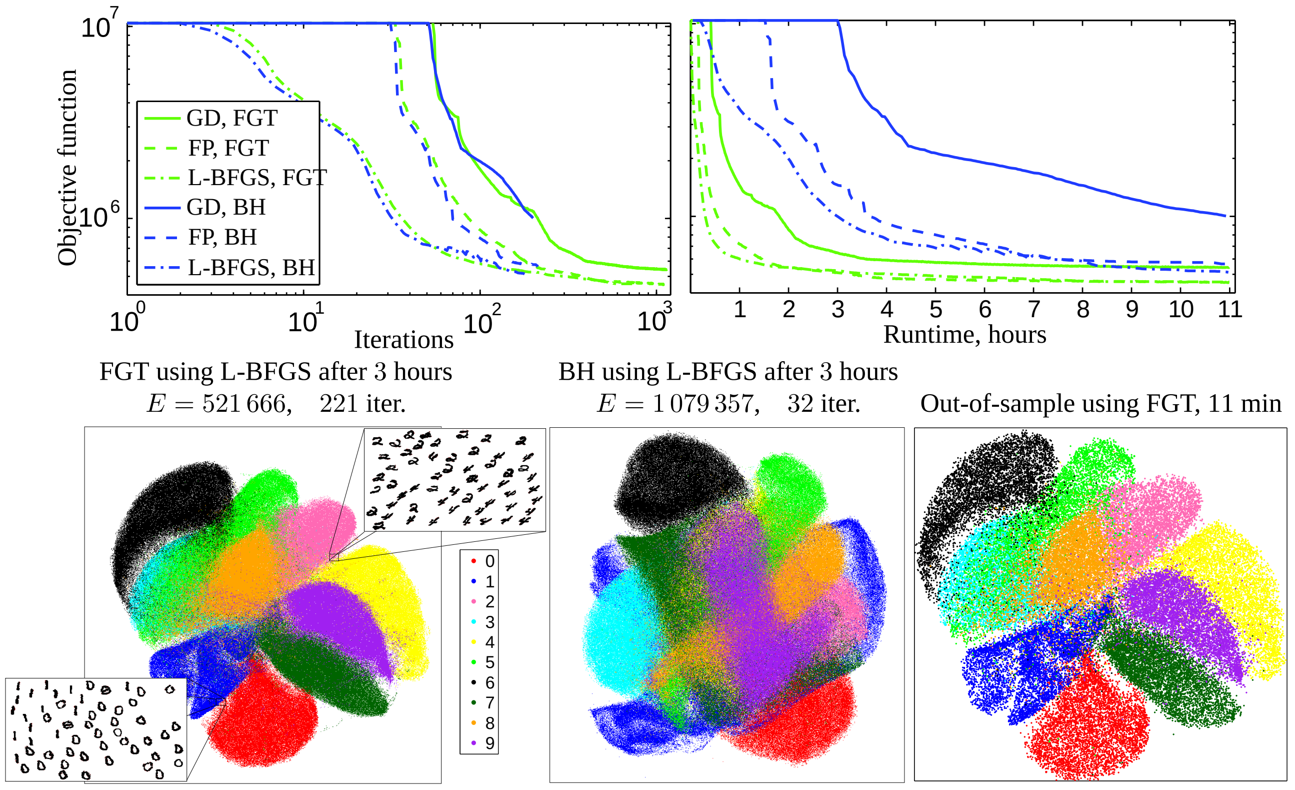

Figure 6: Embeddings of $1\,020\,000$ digits from the infinite MNIST dataset using the elastic embedding algorithm with FGT and BH, optimized with gradient descent (GD), fixed-point iteration (FP) and L-BFGS. *Top*: objective value change with respect to the number of iterations and runtime. *Bottom left two plots*: 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. *Bottom right plot*: out-of-sample projection of $60\,000$ digits using the embedding of L-BFGS as a training set.

objective function values shown in the plot are not needed in the optimization and are computed exactly offline.

Next, we used an infinite MNIST dataset (Loosli et al., 2007) where $960\,000$ handwritten digits were generated using elastic deformations to the original MNIST dataset. Together with the original MNIST digits the dataset consists of $1\,020\,000$ points. 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. 6 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-sample mapping (7). 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. We obtained the embedding of the test points in just 11 minutes and the embedding agrees with the structure of the training dataset.

5 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.

References

J. Barnes and P. Hut. A hierarchical $\mathcal{O}(N \log N)$ force-calculation algorithm. *Nature*, 324(6096):446–449, Dec. 4 1986.

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

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.

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

M. Á. Carreira-Perpiñán. The elastic embedding algorithm for dimensionality reduction. In J. Fürnkranz and T. Joachims, editors, *Proc. of the 27th Int. Conf. Machine Learning (ICML 2010)*, pages 167–174, 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.

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.

W. Fong and E. Darve. The black-box fast multipole method. *J. Comp. Phys.*, 228(23):8712–8725, Dec. 10 2009.

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.

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

A. G. Gray and A. W. Moore. 'N-body' problems in statistical learning. In T. K. Leen, T. G. Dietterich, 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.

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.

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.

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 Series, pages 301–320. MIT Press, 2007.

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.

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 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 2006.

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*, Neural Information Processing Series, pages 175–202. MIT Press, 2007.

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, Mar. 1994.

H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 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, pages 1225–1232. MIT Press, Cambridge, MA, 2006.

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

J. Strain. The fast Gauss transform with variable scales. *SIAM J. Sci. Stat. Comput.*, 12(5):1131–1139, Sept. 1991.

L. J. P. van der Maaten. Barnes-Hut-SNE. In *Int. Conf. Learning Representations (ICLR 2013)*, Scottsdale, AZ, May 2–4 2013.

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

M. Vladymyrov and M. Á. Carreira-Perpiñán. Partial-Hessian strategies for fast learning of nonlinear embeddings. In J. Langford and J. Pineau, editors, *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 S. Dasgupta and D. McAllester, editors, *Proc. of the 30th Int. Conf. Machine Learning (ICML 2013)*, pages 477–485, Atlanta, GA, June 16–21 2013.

X. Wan and G. E. Karniadakis. A sharp error estimate for the fast Gauss transform. *J. Comp. Phys.*, 219(1):7–12, Nov. 20 2006.

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 S. Dasgupta and D. McAllester, editors, *Proc. of the 30th Int. Conf. Machine Learning (ICML 2013)*, pages 127–135, Atlanta, GA, June 16–21 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, May 20 2004.