11 Self-Organizing Algorithms

A "self-organizing" algorithm is an algorithm that does not use a supervisor when learning. That is, there are no target values available and the learning must therefore be based on some other principle than approximating the teacher signal. The most natural is to look for some order in the input data. "Order" here meaning something that deviates from total randomness. Examples of order are:

- Some input variables are correlated.
- Data is clustered into smaller sub-clusters in the input space. (E.g. finding two distinct clusters among the input stimuli indicates that there are two categories of stimuli.)

11.1 The autoencoder

A simple example of a self-organizing algorithm is the one-hidden-layer autoencoder, where the input $\mathbf{x}$ is also used as the desired output $\mathbf{y}$ and the network is a bottle-neck multilayer perceptron (MLP). The network is then trained by minimizing the summed square error. If there are D inputs, M hidden units, and D outputs (with $M < D$) then the weight vectors $\mathbf{w}_j (j = 1, \ldots, M)$ will span the same subspace as the M leading principal components (Bishop 1995). This does not, however, mean that the autoencoder is equivalent to PCA. This is for instance demonstrated in (Frosini, Gori & Priami 1996), where it is argued that the nonlinear autoencoder is to be preferred for fault-detection and verification tasks.

The three-hidden-layer autoencoder performs nonlinear PCA. However, this is a very complicated network structure and takes a long time to train, and it is questionable if it is of any use for practical problems.

11.1.1 Principal Component Analysis

The idea behind Principal component analysis (PCA) is to have an effective linear coding method for multivariate data. If some of the input variables x_k are linearly correlated with each other then it is obviously possible to recode them using uncorrelated variables and possibly fewer variables.

We have an input data set $\mathcal{X} = \{\mathbf{x}(n)\}_{n=1,\ldots,N}$, where $\mathbf{x}(n)$ is a D-dimensional vector. The covariance matrix for the data set $\mathcal{X}$ is

$$\Sigma_{xx} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1D} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{D1} & \sigma_{D2} & \cdots & \sigma_{DD} \end{pmatrix} \tag{11.1}$$

where

$$\begin{aligned} \sigma_{kl} &= \langle x_k x_l \rangle - \langle x_k \rangle \langle x_l \rangle \\ &= \frac{1}{N} \sum_n x_k(n) x_l(n) - \left(\frac{1}{N} \sum_n x_k(n) \right) \left(\frac{1}{N} \sum_n x_l(n) \right). \end{aligned} \tag{11.2}$$

The covariance matrix describes the correlations between the components of $\mathbf{x}$. If we want to code $\mathbf{x}$ as efficiently as possible, using a linear transformation, then we should transform $\mathbf{x}$ to a new representation $\mathbf{z}$

$$\mathbf{z} = \mathbf{A}\mathbf{x} \tag{11.3}$$

such that the covariance matrix in this new representation $\mathbf{z}$ will be diagonal (i.e. no correlations exists between the components of $\mathbf{z}$).

It is obvious from equation (11.2) that the covariance matrix is a symmetric matrix. It follows from the spectral theorem that the covariance matrix can be diagonalized using an orthogonal transformation $\mathbf{Q}$ (i.e. a matrix $\mathbf{Q}$ such that $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$). That is, there exists an orthogonal matrix $\mathbf{Q}$ such that

$$\mathbf{\Lambda} = \mathbf{Q}^T \mathbf{\Sigma}_{xx} \mathbf{Q} \tag{11.4}$$

where $\mathbf{\Lambda}$ is a diagonal matrix. Hence, if we choose $\mathbf{A} = \mathbf{Q}^T$ then the covariance matrix for $\mathbf{z}$ will be equal to $\mathbf{\Lambda}$.

The matrix $\mathbf{Q}$ has the eigenvectors of $\mathbf{\Sigma}_{xx}$ as columns and the diagonal elements of $\mathbf{\Lambda}$ are the eigenvalues of $\mathbf{\Sigma}_{xx}$. The components of $\mathbf{z}$ are given by the scalar products of $\mathbf{x}$ with the eigenvectors of $\mathbf{\Sigma}_{xx}$

$$z_i = \mathbf{q}_i^T \mathbf{x} \tag{11.5}$$

where

$$\mathbf{\Sigma}_{xx} \mathbf{q}_i = \lambda_i \mathbf{q}_i. \tag{11.6}$$

From now on, we assume that we have sorted the eigenvalues, and corresponding eigenvectors, of $\mathbf{\Sigma}_{xx}$ such that

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_D. \tag{11.7}$$

The set $\{\mathbf{q}_i\}$ consists of D orthogonal vectors, which means we can use them as a basis vector set (all we need to do is to normalize them to unit length, which is easily done). The vector $\mathbf{x}$ will be expressed, in this new basis, as

$$\mathbf{x} = \sum_{k=1}^{D} \alpha_k \mathbf{q}_k \tag{11.8}$$

and if we multiply by $\mathbf{q}^T$ from the left then we see that $\alpha_k = z_k$, i.e.

$$\mathbf{x} = \sum_{k=1}^{D} z_k \boldsymbol{q}_k. \tag{11.9}$$

It is now appropriate to ask how much information we loose if we express $\mathbf{x}$ using only the M first components (where $M < D$). If we measure the loss using a sum square error then we have

$$E = \sum_n \|\mathbf{x}(n) - \tilde{\mathbf{x}}(n)\|^2 \tag{11.10}$$

where

$$\tilde{\mathbf{x}}(n) = \sum_{k=1}^{M} z_k(n) \mathbf{q}_k \tag{11.11}$$

is the reconstructed signal.

The reconstruction error is, assuming that $\langle z_k \rangle = 0$,

$$\begin{aligned} E &= \sum_n \sum_{k=M+1}^{D} \|z_k(n)\mathbf{q}_k\|^2 \\ &= \sum_n \sum_{k=M+1}^{D} z_k^2(n) \\ &= N \sum_{k=M+1}^{D} \lambda_k \end{aligned} \tag{11.12}$$

since the eigenvectors are orthornormal.

In summary, we can, by coding $\mathbf{x}$ using the eigenvectors of the covariance matrix $\boldsymbol{\Sigma}_{xx}$ ordered by descending eigenvalues, minimize the information loss ("information" is here measured by the sum squared loss) if the last components are left out. (It is often the case that the last components correspond to noise in the signal so it is not a bad idea to omit them.)

The components z_k are called the *principal components* for $\mathbf{x}$, and the vectors $\mathbf{q}_k$ are called *principal directions* or *principal basis*.

11.1.2 ANN algorithms for PCA

Haykin (1999) describes several ANN algorithms for doing PCA. Bishop (1995) also briefly goes through the essentials for them.

As mentioned above, an autoencoder with $M(< D)$ hidden units will generate weight vectors that span the same subspace as the first M principal directions. Thus, the autoencoder will be essentially a linear machine. However, if we allow three hidden layers, then we can perform nonlinear PCA (but no-one has shown any empirical benefit over linear PCA using such nonlinear PCA). The latter of course requires much more training time.

In principle, the standard methods for doing PCA reduction (e.g. diagonalization of a matrix using the singular value decomposition) are robust, well-designed and fast. There is no reason to do PCA using ANN models.

11.2 Clustering algorithms (K-means/VQ)

A common self-organizing principle is to search for clusters in the input data distribution. In this context the data is not coded in terms of projections, as in the principal component case, but in terms of radial basis functions. The idea is to find the set of center locations $\mathbf{w}_k$, or cluster centers, that best describe the data set. This is done by minimizing e.g. the reconstruction error

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} \Lambda_k[\mathbf{x}(n)] \|\mathbf{x}(n) - \mathbf{w}_k\|^2, \tag{11.13}$$

subject to the constraint that $K \ll N$ (K can be either fixed or adaptive). Here $\Lambda_k[\mathbf{x}(n)]$ is a membership function that takes the values

$$\Lambda_k[\mathbf{x}(n)] = \begin{cases} 1 & \text{if } \|\mathbf{x}(n) - \mathbf{w}_k\| < \|\mathbf{x}(n) - \mathbf{w}_j\| \ \forall \ j \neq k \\ 0 & \text{otherwise} \end{cases} . \tag{11.14}$$

Gradient descent on (11.13) leads to the updating equation

$$\mathbf{w}_k(n+1) = \mathbf{w}_k(n) + \eta \sum_{n=1}^{N} \Lambda_k[\mathbf{x}(n)] \left[\mathbf{x}(n) - \mathbf{w}_k\right], \tag{11.15}$$

which is the K-means algorithm. It is also sometimes called vector quantization (VQ).

We have already mentioned the K-means algorithm in an earlier lecture. However, we repeat it here for later reference in this lecture (actually the on-line form of it).

1. Start with initial values for $\mathbf{w}_k$, $k = 1, \ldots, K$, e.g. random.
2. Repeat until the reconstruction error (11.13) is below some prespecified value, or until the weight changes are very small.
   2.1 Present a pattern $\mathbf{x}(n)$.
   2.2 Find the center location $\mathbf{w}_k$ that lies closest to the presented pattern. That is, the weight vector $\mathbf{w}_k$ that fulfills $\|\mathbf{x}(n) - \mathbf{w}_k\| < \|\mathbf{x}(n) - \mathbf{w}_j\| \ \forall \ j \neq k$. Denote this center location as the "winner".
   2.3 Compute the update $\Delta\mathbf{w}_k(n) = \eta\,[\mathbf{x}(n) - \mathbf{w}_k]$ for the winner unit.
   2.4 Update the winner unit: $\mathbf{w}_k(n+1) = \mathbf{w}_k(n) + \Delta\mathbf{w}_k(n)$. Do not update any other unit.

For this algorithm to really converge, it is necessary to let the learning rate (or step length) η decrease with time.

Once all the center locations, or code-book vectors, $\mathbf{w}_k$ have converged to stable locations, then we can use them to code the data. The simplest coding would be to replace the observation $\mathbf{x}(n)$ with (the index of) its closest matching code-book vector. Another, slightly more complicated, is to replace $\mathbf{x}(n)$ with the indices of the L closest code-book vectors, and the distance to them. This information can then be used to "triangulate" and thereby reproduce the observation with a higher precision than if only the winning unit is used.

11.3 Learning vector quantization (LVQ)

Kohonen (1988) suggested to complement the VQ algorithm with a "learning" phase, if a teacher[1] was available, into an algorithm called learning vector quantization (LVQ) (see e.g. (Kangas, Kohonen & Laaksonen 1990)). LVQ is designed for classification problems and assumes that there is a teacher available that has supplied the correct categories for every observation $\mathbf{x}(n)$. The algorithm is:

1. Start with initial values for $\mathbf{w}_k$, $k = 1, \ldots, K$, e.g. random. Or with starting values from a VQ/K-means clustering.
2. Assign each weight vector $\mathbf{w}_k$ to a category, for instance to the category most often represented in the observations that are closest to this weight vector.

[1] Of course, this is no longer a self-organizing algorithm when a teacher is involved.

3. Repeat until the classification error is below some prespecified value, or until the weight changes are very small.

   3.1 Present a pattern $\mathbf{x}(n)$.

   3.2 Find the center location $\mathbf{w}_k$ that lies closest to the presented pattern. That is, the weight vector $\mathbf{w}_k$ that fulfills $\|\mathbf{x}(n)-\mathbf{w}_k\| < \|\mathbf{x}(n)-\mathbf{w}_j\| \; \forall \; j \neq k$. Denote this center location as the "winner".

   3.3 Compute the update $\Delta\mathbf{w}_k(n) = \pm\eta\,[\mathbf{x}(n) - \mathbf{w}_k]$ for the winner unit. The "+" sign is used if the observation $\mathbf{x}(n)$ and $\mathbf{w}_k$ belong to the same category, otherwise the "−" sign is used. This means that $\mathbf{w}_k$ is moved closer to $\mathbf{x}(n)$ if their category labels match, otherwise it is moved away.

   3.4 Update the winner unit: $\mathbf{w}_k(n+1) = \mathbf{w}_k(n) + \Delta\mathbf{w}_k(n)$. Do not update any other unit.

Just as in the VQ case, η must decrease with time.

It is sometimes claimed that LVQ is a better algorithm than the MLP for classification. This is not true. To my knowledge, the only cases where LVQ has been empirically shown to outperform the MLP are cases where the MLP has been severely mistreated!

11.4 Self-organizing maps

Kohonen (1988) is most well-known for his self-organizing maps, which are quite frequently used. The self-organizing map (SOM) is an extension of the K-means algorithm (or VQ) where a topological structure is imposed on the weight vectors $\mathbf{w}_k$, by introducing a "neighborhood" function Λ_{jk}. For instance, the $\mathbf{w}_k$ could be organized on a two-dimensional grid. The "neighborhood" function Λ_{jk} could be of the form

$$\Lambda_{jk} = \exp\left(\frac{-d_{jk}}{2\sigma}\right), \tag{11.16}$$

where d_{jk} is the distance between $\mathbf{w}_k$ and $\mathbf{w}_j$ in the topological structure. For instance, if a two-dimensional lattice structure is imposed and $\mathbf{w}_k$ and $\mathbf{w}_j$ are neighbors in the lattice, then $d_{jk} = 1$.

The neighborhood function is used in the weight update so that also neighbors of the winning unit are updated along with the winner. The algorithm goes as:

1. Start with initial values for $\mathbf{w}_k$, $k = 1, \ldots, K$, e.g. random.
2. Repeat until the weight changes are very small.
   - 2.1 Present a pattern $\mathbf{x}(n)$.
   - 2.2 Find the center location $\mathbf{w}_k$ that lies closest to the presented pattern. That is, the weight vector $\mathbf{w}_k$ that fulfills $\|\mathbf{x}(n) - \mathbf{w}_k\| < \|\mathbf{x}(n) - \mathbf{w}_j\| \ \forall \ j \neq k$. Denote this center location as the "winner".
   - 2.3 Compute the update $\Delta\mathbf{w}_j(n) = \eta\Lambda_{jk}\left[\mathbf{x}(n) - \mathbf{w}_j\right]$ for all units.
   - 2.4 Update all units: $\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \Delta\mathbf{w}_j(n)$.

The learning rate η is decreased with time. Furthermore, the width σ in (11.16) starts out wide (several lattice distance units) and shrinks with time. This means that the lattice will eventually cover the data distribution and be a map of it. However, a requirement for the SOM to be a faithful map of the data distribution is that the topology of the map matches the topology of the data distribution. For instance, if the data distribution is three dimensional and the SOM is two dimensional, then the map will be distorted.

The SOM is useful for data analysis, since it is possible to see in the network what type of observations that lie close to each other in the input space. This is also why it is called a map.

Some problems with the SOM algorithm are that the number of cluster locations K is fixed, and that the topology is fixed. There are, however, extensions and adjustments to the SOM which address these problems, and we cover two of them below but we will first describe an extension of the SOM to a local linear mapping.

11.4.1 SOM combined with local linear maps

A very interesting development of the SOM algorithm is to connect a linear mapping to each node in the SOM (Martinetz, Ritter & Schulten 1990). This produces a non-linear (but piecewise linear) function. This is similar to the gain scheduling algorithm in control, where the system is linearized around a set of operation points.

The SOM with local linear maps has been used with some success to control a robot arm using visual feedback (Martinetz *et al.* 1990). In this case the task is to realize a mapping from two 2-dimensional visual inputs (camera images) to a vector with three angles, representing the position of the robot

arm. The robot is required to learn this mapping without using any teacher, so that it can be visually commanded to go to a specific position afterwards. The equations that control the process are

$$\theta(\mathbf{x}) = \theta_{j*} + \mathbf{A}_{j*}(\mathbf{x} - \mathbf{w}_{j*}) \tag{11.17}$$

where $\mathbf{x}$ is the visual input (4-dimensional), $\mathbf{w}_{j*}$ is the 4-dimensional weight vector for the winning node, θ_{j*} is a set of angles associated with the winning node (actually the center position for the winning node), and $\mathbf{A}_{j*}$ is a matrix connected with the winning node which gives a first order correction to θ_{j*} when $\mathbf{x} \neq \mathbf{w}_{j*}$ (but $j*$ is still the winning node).

The training of this system is not too complicated and we will cover this in the seminars.

11.4.2 The neural gas

The *neural gas* algorithm (Martinetz & Schulten 1991) is an approach to avoid the assumption of the topology of the SOM, and let the algorithm discover the topology itself. However, the algorithm uses a fixed number of units.

The neighborhood function is

$$\Lambda_{jj*} = \exp\left[\frac{-k_j}{\Delta}\right] \tag{11.18}$$

where k_j is the ranking that node j has in relation to the input signal $\mathbf{x}$. The winning node (i.e. the node closest to $\mathbf{x}$) is given the ranking $k_{j*} = 0$. The second closest node is given the rank 1, and so on. The weights $\mathbf{w}_j$ are then updated according to

$$\mathbf{w}_j = \mathbf{w}_j + \eta \exp[-k_j/\Delta](\mathbf{x}(n) - \mathbf{w}_j). \tag{11.19}$$

The topology of the map is constucted using a connectivity matrix $\mathbf{C}$ with the elements 0 or 1

$$C_{jk} = \begin{cases} 1 & \text{if nodes } j \text{ and } k \text{ are neighbors} \\ 0 & \text{otherwise} \end{cases} . \tag{11.20}$$

The connection elements C_{jk} are then given an age, i.e. the ages for all elements $\mathbf{C}$ increase by one every iteration by the algorithm. If the age for a connection is above some threshold, then that connection is set ot 0. The connection element C_{jj*} connecting the winning node with the second closest node is set to 1 at every iteration, and the age for the connection is set to zero.

1. Choose how many cluster centers to use.
2. Initiate all w_{ij} randomly. Set $C_{jk} = 0$ for all j and k. Set the age of all connections to $\tau_{jk} = 0$.
3. Repeat until all $\mathbf{w}_j$ and C_{jk} have converged (i.e. when they do not change much anymore):
   - 3.1 Select a random input pattern $\mathbf{x}(n)$ from the data set $\mathcal{X}$.
   - 3.2 Find the winning node $y_{j*}(n)$ and give it the rank $k_{j*} = 0$. Rank the remaining nodes by $k_j = 1, 2, 3, \ldots$ depending on close they are to the input pattern.
   - 3.3 Uppdate the weights according to: $\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta \exp[-k_j/\Delta][\mathbf{x}(n) - \mathbf{w}_j(t)]$. You must let the step length $\eta \to 0$ as the algorithm converges. It is also common to let the neighborhood function's $\Delta \to 0$.
   - 3.4 Uppdate the ages $\tau_{jk} = \tau_{jk} + 1$ for all j and k. Set the connection $C_{jj*} = 1$ between the winning node and and the second closest node, also set the age $\tau_{jj*} = 0$.
   - 3.5 Set $C_{jk} = 0$ for all connections that have grown too old, i.e. for which τ_{jk} is above the predetermined threshold.

Figure 11.1: Self-organizing "neuron gas".

References

Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford.

Carpenter, G. & Grossberg, S. (1988), 'The ART of adaptive pattern recognition by a self-organizing neural network', *Computer* pp. 77–88.

Fritzke, B. (1994), 'Growing cell structures - a self-organizing network for unsupervised and supervised learning', *Neural Networks* **7**, 1441–1460.

Frosini, A., Gori, M. & Priami, P. (1996), 'A neural network-based model for paper currency recognition and verification', *IEEE Transactions on Neural Networks* **7**, 1482–1490.

Haykin, S. (1999), *Neural Networks – A Comprehensive Foundation*, second edn, Prentice Hall Inc., Upper Saddle River, New Jersey.

Kangas, J. A., Kohonen, T. K. & Laaksonen, J. T. (1990), 'Variants of self-organizing maps', *IEEE Transactions on Neural Networks* **1**, 93–99.

Kohonen, T. (1988), *Self-Organization and Associative Memory* 2^{nd}*ed.*, Springer-Verlag, Berlin, Germany.

Martinetz, T. & Schulten, K. (1991), A neural-gas network learns topologies, *in* 'Artificial Neural Networks', Elsevier Science Publishers, Amsterdam, pp. 397–402.

Martinetz, T. M., Ritter, H. J. & Schulten, K. J. (1990), 'Three-dimensional neural net for learning visuomotor coordination of a robot arm', *IEEE Transactions on Neural Networks* **1**, 131–136.

Journal of Machine Learning Research 9 (2008) 2579-2605 Submitted 5/08; Revised 9/08; Published 11/08

Visualizing Data using t-SNE

Laurens van der Maaten LVDMAATEN@GMAIL.COM
TiCC
Tilburg University
P.O. Box 90153, 5000 LE Tilburg, The Netherlands

Geoffrey Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
6 King's College Road, M5S 3G4 Toronto, ON, Canada

Editor: Yoshua Bengio

Abstract

We present a new technique called "t-SNE" that visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map. The technique is a variation of Stochastic Neighbor Embedding (Hinton and Roweis, 2002) that is much easier to optimize, and produces significantly better visualizations by reducing the tendency to crowd points together in the center of the map. t-SNE is better than existing techniques at creating a single map that reveals structure at many different scales. This is particularly important for high-dimensional data that lie on several different, but related, low-dimensional manifolds, such as images of objects from multiple classes seen from multiple viewpoints. For visualizing the structure of very large data sets, we show how t-SNE can use random walks on neighborhood graphs to allow the implicit structure of all of the data to influence the way in which a subset of the data is displayed. We illustrate the performance of t-SNE on a wide variety of data sets and compare it with many other non-parametric visualization techniques, including Sammon mapping, Isomap, and Locally Linear Embedding. The visualizations produced by t-SNE are significantly better than those produced by the other techniques on almost all of the data sets.

Keywords: visualization, dimensionality reduction, manifold learning, embedding algorithms, multidimensional scaling

1. Introduction

Visualization of high-dimensional data is an important problem in many different domains, and deals with data of widely varying dimensionality. Cell nuclei that are relevant to breast cancer, for example, are described by approximately 30 variables (Street et al., 1993), whereas the pixel intensity vectors used to represent images or the word-count vectors used to represent documents typically have thousands of dimensions. Over the last few decades, a variety of techniques for the visualization of such high-dimensional data have been proposed, many of which are reviewed by de Oliveira and Levkowitz (2003). Important techniques include iconographic displays such as Chernoff faces (Chernoff, 1973), pixel-based techniques (Keim, 2000), and techniques that represent the dimensions in the data as vertices in a graph (Battista et al., 1994). Most of these techniques simply provide tools to display more than two data dimensions, and leave the interpretation of the

data to the human observer. This severely limits the applicability of these techniques to real-world data sets that contain thousands of high-dimensional datapoints.

In contrast to the visualization techniques discussed above, dimensionality reduction methods convert the high-dimensional data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ into two or three-dimensional data $\mathcal{Y} = \{y_1, y_2, ..., y_n\}$ that can be displayed in a scatterplot. In the paper, we refer to the low-dimensional data representation $\mathcal{Y}$ as a map, and to the low-dimensional representations y_i of individual datapoints as map points. The aim of dimensionality reduction is to preserve as much of the significant structure of the high-dimensional data as possible in the low-dimensional map. Various techniques for this problem have been proposed that differ in the type of structure they preserve. Traditional dimensionality reduction techniques such as Principal Components Analysis (PCA; Hotelling, 1933) and classical multidimensional scaling (MDS; Torgerson, 1952) are linear techniques that focus on keeping the low-dimensional representations of dissimilar datapoints far apart. For high-dimensional data that lies on or near a low-dimensional, non-linear manifold it is usually more important to keep the low-dimensional representations of very similar datapoints close together, which is typically not possible with a linear mapping.

A large number of nonlinear dimensionality reduction techniques that aim to preserve the local structure of data have been proposed, many of which are reviewed by Lee and Verleysen (2007). In particular, we mention the following seven techniques: (1) Sammon mapping (Sammon, 1969), (2) curvilinear components analysis (CCA; Demartines and Hérault, 1997), (3) Stochastic Neighbor Embedding (SNE; Hinton and Roweis, 2002), (4) Isomap (Tenenbaum et al., 2000), (5) Maximum Variance Unfolding (MVU; Weinberger et al., 2004), (6) Locally Linear Embedding (LLE; Roweis and Saul, 2000), and (7) Laplacian Eigenmaps (Belkin and Niyogi, 2002). Despite the strong performance of these techniques on artificial data sets, they are often not very successful at visualizing real, high-dimensional data. In particular, most of the techniques are not capable of retaining both the local and the global structure of the data in a single map. For instance, a recent study reveals that even a semi-supervised variant of MVU is not capable of separating handwritten digits into their natural clusters (Song et al., 2007).

In this paper, we describe a way of converting a high-dimensional data set into a matrix of pairwise similarities and we introduce a new technique, called "t-SNE", for visualizing the resulting similarity data. t-SNE is capable of capturing much of the local structure of the high-dimensional data very well, while also revealing global structure such as the presence of clusters at several scales. We illustrate the performance of t-SNE by comparing it to the seven dimensionality reduction techniques mentioned above on five data sets from a variety of domains. Because of space limitations, most of the $(7+1) \times 5 = 40$ maps are presented in the supplemental material, but the maps that we present in the paper are sufficient to demonstrate the superiority of t-SNE.

The outline of the paper is as follows. In Section 2, we outline SNE as presented by Hinton and Roweis (2002), which forms the basis for t-SNE. In Section 3, we present t-SNE, which has two important differences from SNE. In Section 4, we describe the experimental setup and the results of our experiments. Subsequently, Section 5 shows how t-SNE can be modified to visualize real-world data sets that contain many more than $10,000$ datapoints. The results of our experiments are discussed in more detail in Section 6. Our conclusions and suggestions for future work are presented in Section 7.

2. Stochastic Neighbor Embedding

Stochastic Neighbor Embedding (SNE) starts by converting the high-dimensional Euclidean distances between datapoints into conditional probabilities that represent similarities.[1] The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$, that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i. For nearby datapoints, $p_{j|i}$ is relatively high, whereas for widely separated datapoints, $p_{j|i}$ will be almost infinitesimal (for reasonable values of the variance of the Gaussian, σ_i). Mathematically, the conditional probability $p_{j|i}$ is given by

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-\|x_i - x_k\|^2/2\sigma_i^2\right)}, \tag{1}$$

where σ_i is the variance of the Gaussian that is centered on datapoint x_i. The method for determining the value of σ_i is presented later in this section. Because we are only interested in modeling pairwise similarities, we set the value of $p_{i|i}$ to zero. For the low-dimensional counterparts y_i and y_j of the high-dimensional datapoints x_i and x_j, it is possible to compute a similar conditional probability, which we denote by $q_{j|i}$. We set[2] the variance of the Gaussian that is employed in the computation of the conditional probabilities $q_{j|i}$ to $\frac{1}{\sqrt{2}}$. Hence, we model the similarity of map point y_j to map point y_i by

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k\neq i}\exp\left(-\|y_i - y_k\|^2\right)}.$$

Again, since we are only interested in modeling pairwise similarities, we set $q_{i|i} = 0$.

If the map points y_i and y_j correctly model the similarity between the high-dimensional datapoints x_i and x_j, the conditional probabilities $p_{j|i}$ and $q_{j|i}$ will be equal. Motivated by this observation, SNE aims to find a low-dimensional data representation that minimizes the mismatch between $p_{j|i}$ and $q_{j|i}$. A natural measure of the faithfulness with which $q_{j|i}$ models $p_{j|i}$ is the Kullback-Leibler divergence (which is in this case equal to the cross-entropy up to an additive constant). SNE minimizes the sum of Kullback-Leibler divergences over all datapoints using a gradient descent method. The cost function C is given by

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}, \tag{2}$$

in which P_i represents the conditional probability distribution over all other datapoints given datapoint x_i, and Q_i represents the conditional probability distribution over all other map points given map point y_i. Because the Kullback-Leibler divergence is not symmetric, different types of error in the pairwise distances in the low-dimensional map are not weighted equally. In particular, there is a large cost for using widely separated map points to represent nearby datapoints (i.e., for using

1. SNE can also be applied to data sets that consist of pairwise similarities between objects rather than high-dimensional vector representations of each object, provided these simiarities can be interpreted as conditional probabilities. For example, human word association data consists of the probability of producing each possible word in response to a given word, as a result of which it is already in the form required by SNE.
2. Setting the variance in the low-dimensional Gaussians to another value only results in a rescaled version of the final map. Note that by using the same variance for every datapoint in the low-dimensional map, we lose the property that the data is a perfect model of itself if we embed it in a space of the same dimensionality, because in the high-dimensional space, we used a different variance σ_i in each Gaussian.

a small $q_{j|i}$ to model a large $p_{j|i}$), but there is only a small cost for using nearby map points to represent widely separated datapoints. This small cost comes from wasting some of the probability mass in the relevant Q distributions. In other words, the SNE cost function focuses on retaining the local structure of the data in the map (for reasonable values of the variance of the Gaussian in the high-dimensional space, σ_i).

The remaining parameter to be selected is the variance σ_i of the Gaussian that is centered over each high-dimensional datapoint, x_i. It is not likely that there is a single value of σ_i that is optimal for all datapoints in the data set because the density of the data is likely to vary. In dense regions, a smaller value of σ_i is usually more appropriate than in sparser regions. Any particular value of σ_i induces a probability distribution, P_i, over all of the other datapoints. This distribution has an entropy which increases as σ_i increases. SNE performs a binary search for the value of σ_i that produces a P_i with a fixed perplexity that is specified by the user.[3] The perplexity is defined as

$$Perp(P_i) = 2^{H(P_i)},$$

where $H(P_i)$ is the Shannon entropy of P_i measured in bits

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}.$$

The perplexity can be interpreted as a smooth measure of the effective number of neighbors. The performance of SNE is fairly robust to changes in the perplexity, and typical values are between 5 and 50.

The minimization of the cost function in Equation 2 is performed using a gradient descent method. The gradient has a surprisingly simple form

$$\frac{\delta C}{\delta y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j).$$

Physically, the gradient may be interpreted as the resultant force created by a set of springs between the map point y_i and all other map points y_j. All springs exert a force along the direction $(y_i - y_j)$. The spring between y_i and y_j repels or attracts the map points depending on whether the distance between the two in the map is too small or too large to represent the similarities between the two high-dimensional datapoints. The force exerted by the spring between y_i and y_j is proportional to its length, and also proportional to its stiffness, which is the mismatch $(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})$ between the pairwise similarities of the data points and the map points.

The gradient descent is initialized by sampling map points randomly from an isotropic Gaussian with small variance that is centered around the origin. In order to speed up the optimization and to avoid poor local minima, a relatively large momentum term is added to the gradient. In other words, the current gradient is added to an exponentially decaying sum of previous gradients in order to determine the changes in the coordinates of the map points at each iteration of the gradient search. Mathematically, the gradient update with a momentum term is given by

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right),$$

3. Note that the perplexity increases monotonically with the variance σ_i.

where $\mathcal{Y}^{(t)}$ indicates the solution at iteration t, η indicates the learning rate, and $\alpha(t)$ represents the momentum at iteration t.

In addition, in the early stages of the optimization, Gaussian noise is added to the map points after each iteration. Gradually reducing the variance of this noise performs a type of simulated annealing that helps the optimization to escape from poor local minima in the cost function. If the variance of the noise changes very slowly at the critical point at which the global structure of the map starts to form, SNE tends to find maps with a better global organization. Unfortunately, this requires sensible choices of the initial amount of Gaussian noise and the rate at which it decays. Moreover, these choices interact with the amount of momentum and the step size that are employed in the gradient descent. It is therefore common to run the optimization several times on a data set to find appropriate values for the parameters.[4] In this respect, SNE is inferior to methods that allow convex optimization and it would be useful to find an optimization method that gives good results without requiring the extra computation time and parameter choices introduced by the simulated annealing.

3. t-Distributed Stochastic Neighbor Embedding

Section 2 discussed SNE as it was presented by Hinton and Roweis (2002). Although SNE constructs reasonably good visualizations, it is hampered by a cost function that is difficult to optimize and by a problem we refer to as the "crowding problem". In this section, we present a new technique called "t-Distributed Stochastic Neighbor Embedding" or "t-SNE" that aims to alleviate these problems. The cost function used by t-SNE differs from the one used by SNE in two ways: (1) it uses a symmetrized version of the SNE cost function with simpler gradients that was briefly introduced by Cook et al. (2007) and (2) it uses a Student-t distribution rather than a Gaussian to compute the similarity between two points *in the low-dimensional space*. t-SNE employs a heavy-tailed distribution in the low-dimensional space to alleviate both the crowding problem and the optimization problems of SNE.

In this section, we first discuss the symmetric version of SNE (Section 3.1). Subsequently, we discuss the crowding problem (Section 3.2), and the use of heavy-tailed distributions to address this problem (Section 3.3). We conclude the section by describing our approach to the optimization of the t-SNE cost function (Section 3.4).

3.1 Symmetric SNE

As an alternative to minimizing the sum of the Kullback-Leibler divergences between the conditional probabilities $p_{j|i}$ and $q_{j|i}$, it is also possible to minimize a single Kullback-Leibler divergence between a joint probability distribution, P, in the high-dimensional space and a joint probability distribution, Q, in the low-dimensional space:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

where again, we set p_{ii} and q_{ii} to zero. We refer to this type of SNE as symmetric SNE, because it has the property that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ for $\forall i, j$. In symmetric SNE, the pairwise similarities in

4. Picking the best map after several runs as a visualization of the data is not nearly as problematic as picking the model that does best on a test set during supervised learning. In visualization, the aim is to see the structure in the training data, not to generalize to held out test data.

the low-dimensional map q_{ij} are given by

$$q_{ij} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq l} \exp\left(-\|y_k - y_l\|^2\right)}, \tag{3}$$

The obvious way to define the pairwise similarities in the high-dimensional space p_{ij} is

$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|^2/2\sigma^2\right)}{\sum_{k \neq l} \exp\left(-\|x_k - x_l\|^2/2\sigma^2\right)},$$

but this causes problems when a high-dimensional datapoint x_i is an outlier (i.e., all pairwise distances $\|x_i - x_j\|^2$ are large for x_i). For such an outlier, the values of p_{ij} are extremely small for all j, so the location of its low-dimensional map point y_i has very little effect on the cost function. As a result, the position of the map point is not well determined by the positions of the other map points. We circumvent this problem by defining the joint probabilities p_{ij} in the high-dimensional space to be the symmetrized conditional probabilities, that is, we set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$. This ensures that $\sum_j p_{ij} > \frac{1}{2n}$ for all datapoints x_i, as a result of which each datapoint x_i makes a significant contribution to the cost function. In the low-dimensional space, symmetric SNE simply uses Equation 3. The main advantage of the symmetric version of SNE is the simpler form of its gradient, which is faster to compute. The gradient of symmetric SNE is fairly similar to that of asymmetric SNE, and is given by

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j).$$

In preliminary experiments, we observed that symmetric SNE seems to produce maps that are just as good as asymmetric SNE, and sometimes even a little better.

3.2 The Crowding Problem

Consider a set of datapoints that lie on a two-dimensional curved manifold which is approximately linear on a small scale, and which is embedded within a higher-dimensional space. It is possible to model the small pairwise distances between datapoints fairly well in a two-dimensional map, which is often illustrated on toy examples such as the "Swiss roll" data set. Now suppose that the manifold has ten intrinsic dimensions[5] and is embedded within a space of much higher dimensionality. There are several reasons why the pairwise distances in a two-dimensional map cannot faithfully model distances between points on the ten-dimensional manifold. For instance, in ten dimensions, it is possible to have 11 datapoints that are mutually equidistant and there is no way to model this faithfully in a two-dimensional map. A related problem is the very different distribution of pairwise distances in the two spaces. The volume of a sphere centered on datapoint i scales as r^m, where r is the radius and m the dimensionality of the sphere. So if the datapoints are approximately uniformly distributed in the region around i on the ten-dimensional manifold, and we try to model the distances from i to the other datapoints in the two-dimensional map, we get the following "crowding problem": the area of the two-dimensional map that is available to accommodate moderately distant datapoints will not be nearly large enough compared with the area available to accommodate nearby datapoints. Hence, if we want to model the small distances accurately in the map, most of the points

5. This is approximately correct for the images of handwritten digits we use in our experiments in Section 4.

that are at a moderate distance from datapoint i will have to be placed much too far away in the two-dimensional map. In SNE, the spring connecting datapoint i to each of these too-distant map points will thus exert a very small attractive force. Although these attractive forces are very small, the very large number of such forces crushes together the points in the center of the map, which prevents gaps from forming between the natural clusters. Note that the crowding problem is not specific to SNE, but that it also occurs in other local techniques for multidimensional scaling such as Sammon mapping.

An attempt to address the crowding problem by adding a slight repulsion to all springs was presented by Cook et al. (2007). The slight repulsion is created by introducing a uniform background model with a small mixing proportion, ρ. So however far apart two map points are, q_{ij} can never fall below $\frac{2\rho}{n(n-1)}$ (because the uniform background distribution is over $n(n-1)/2$ pairs). As a result, for datapoints that are far apart in the high-dimensional space, q_{ij} will always be larger than p_{ij}, leading to a slight repulsion. This technique is called UNI-SNE and although it usually outperforms standard SNE, the optimization of the UNI-SNE cost function is tedious. The best optimization method known is to start by setting the background mixing proportion to zero (i.e., by performing standard SNE). Once the SNE cost function has been optimized using simulated annealing, the background mixing proportion can be increased to allow some gaps to form between natural clusters as shown by Cook et al. (2007). Optimizing the UNI-SNE cost function directly does not work because two map points that are far apart will get almost all of their q_{ij} from the uniform background. So even if their p_{ij} is large, there will be no attractive force between them, because a small change in their separation will have a vanishingly small *proportional* effect on q_{ij}. This means that if two parts of a cluster get separated early on in the optimization, there is no force to pull them back together.

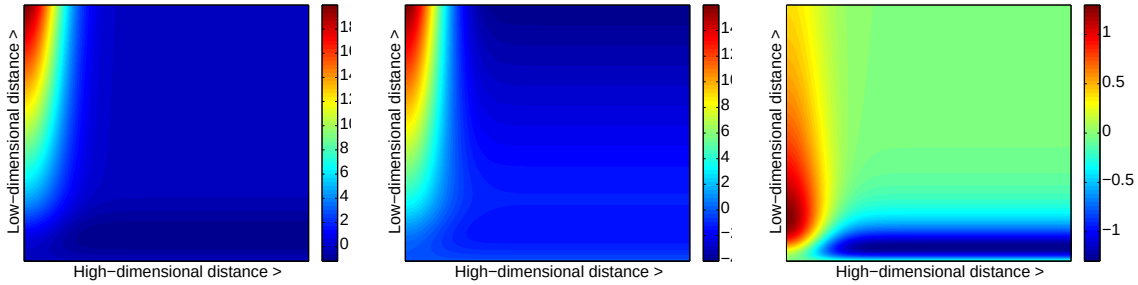
3.3 Mismatched Tails can Compensate for Mismatched Dimensionalities

Since symmetric SNE is actually matching the joint probabilities of pairs of datapoints in the high-dimensional and the low-dimensional spaces rather than their distances, we have a natural way of alleviating the crowding problem that works as follows. In the high-dimensional space, we convert distances into probabilities using a Gaussian distribution. In the low-dimensional map, we can use a probability distribution that has much heavier tails than a Gaussian to convert distances into probabilities. This allows a moderate distance in the high-dimensional space to be faithfully modeled by a much larger distance in the map and, as a result, it eliminates the unwanted attractive forces between map points that represent moderately dissimilar datapoints.

In t-SNE, we employ a Student t-distribution with one degree of freedom (which is the same as a Cauchy distribution) as the heavy-tailed distribution in the low-dimensional map. Using this distribution, the joint probabilities q_{ij} are defined as

$$q_{ij} = \frac{\left(1+\|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1+\|y_k - y_l\|^2\right)^{-1}}. \tag{4}$$

We use a Student t-distribution with a single degree of freedom, because it has the particularly nice property that $\left(1+\|y_i - y_j\|^2\right)^{-1}$ approaches an inverse square law for large pairwise distances $\|y_i - y_j\|$ in the low-dimensional map. This makes the map's representation of joint probabilities (almost) invariant to changes in the scale of the map for map points that are far apart. It also means that large clusters of points that are far apart interact in just the same way as individual points, so the optimization operates in the same way at all but the finest scales. A theoretical justification for our



(a) Gradient of SNE. (b) Gradient of UNI-SNE. (c) Gradient of t-SNE.

Figure 1: Gradients of three types of SNE as a function of the pairwise Euclidean distance between two points in the high-dimensional and the pairwise distance between the points in the low-dimensional data representation.

selection of the Student t-distribution is that it is closely related to the Gaussian distribution, as the Student t-distribution is an infinite mixture of Gaussians. A computationally convenient property is that it is much faster to evaluate the density of a point under a Student t-distribution than under a Gaussian because it does not involve an exponential, even though the Student t-distribution is equivalent to an infinite mixture of Gaussians with different variances.

The gradient of the Kullback-Leibler divergence between P and the Student-t based joint probability distribution Q (computed using Equation 4) is derived in Appendix A, and is given by

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1}. \tag{5}$$

In Figure 1(a) to 1(c), we show the gradients between two low-dimensional datapoints y_i and y_j as a function of their pairwise Euclidean distances in the high-dimensional and the low-dimensional space (i.e., as a function of $\|x_i - x_j\|$ and $\|y_i - y_j\|$) for the symmetric versions of SNE, UNI-SNE, and t-SNE. In the figures, positive values of the gradient represent an attraction between the low-dimensional datapoints y_i and y_j, whereas negative values represent a repulsion between the two datapoints. From the figures, we observe two main advantages of the t-SNE gradient over the gradients of SNE and UNI-SNE.

First, the t-SNE gradient strongly repels dissimilar datapoints that are modeled by a small pairwise distance in the low-dimensional representation. SNE has such a repulsion as well, but its effect is minimal compared to the strong attractions elsewhere in the gradient (the largest attraction in our graphical representation of the gradient is approximately 19, whereas the largest repulsion is approximately 1). In UNI-SNE, the amount of repulsion between dissimilar datapoints is slightly larger, however, this repulsion is only strong when the pairwise distance between the points in the low-dimensional representation is already large (which is often not the case, since the low-dimensional representation is initialized by sampling from a Gaussian with a very small variance that is centered around the origin).

Second, although t-SNE introduces strong repulsions between dissimilar datapoints that are modeled by small pairwise distances, these repulsions do not go to infinity. In this respect, t-SNE differs from UNI-SNE, in which the strength of the repulsion between very dissimilar datapoints

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for *t=1* **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
  end
end

is proportional to their pairwise distance in the low-dimensional map, which may cause dissimilar datapoints to move much too far away from each other.

Taken together, t-SNE puts emphasis on (1) modeling dissimilar datapoints by means of large pairwise distances, and (2) modeling similar datapoints by means of small pairwise distances. Moreover, as a result of these characteristics of the t-SNE cost function (and as a result of the approximate scale invariance of the Student t-distribution), the optimization of the t-SNE cost function is much easier than the optimization of the cost functions of SNE and UNI-SNE. Specifically, t-SNE introduces long-range forces in the low-dimensional map that can pull back together two (clusters of) similar points that get separated early on in the optimization. SNE and UNI-SNE do not have such long-range forces, as a result of which SNE and UNI-SNE need to use simulated annealing to obtain reasonable solutions. Instead, the long-range forces in t-SNE facilitate the identification of good local optima without resorting to simulated annealing.

3.4 Optimization Methods for t-SNE

We start by presenting a relatively simple, gradient descent procedure for optimizing the t-SNE cost function. This simple procedure uses a momentum term to reduce the number of iterations required and it works best if the momentum term is small until the map points have become moderately well organized. Pseudocode for this simple algorithm is presented in Algorithm 1. The simple algorithm can be sped up using the adaptive learning rate scheme that is described by Jacobs (1988), which gradually increases the learning rate in directions in which the gradient is stable.

Although the simple algorithm produces visualizations that are often much better than those produced by other non-parametric dimensionality reduction techniques, the results can be improved further by using either of two tricks. The first trick, which we call "early compression", is to force the map points to stay close together at the start of the optimization. When the distances between map points are small, it is easy for clusters to move through one another so it is much easier to explore the space of possible global organizations of the data. Early compression is implemented by adding an additional L2-penalty to the cost function that is proportional to the sum of squared

distances of the map points from the origin. The magnitude of this penalty term and the iteration at which it is removed are set by hand, but the behavior is fairly robust across variations in these two additional optimization parameters.

A less obvious way to improve the optimization, which we call "early exaggeration", is to multiply all of the p_{ij}'s by, for example, 4, in the initial stages of the optimization. This means that almost all of the q_{ij}'s, which still add up to 1, are much too small to model their corresponding p_{ij}'s. As a result, the optimization is encouraged to focus on modeling the large p_{ij}'s by fairly large q_{ij}'s. The effect is that the natural clusters in the data tend to form tight widely separated clusters in the map. This creates a lot of relatively empty space in the map, which makes it much easier for the clusters to move around relative to one another in order to find a good global organization.

In all the visualizations presented in this paper and in the supporting material, we used exactly the same optimization procedure. We used the early exaggeration method with an exaggeration of 4 for the first 50 iterations (note that early exaggeration is not included in the pseudocode in Algorithm 1). The number of gradient descent iterations T was set 1000, and the momentum term was set to $\alpha^{(t)} = 0.5$ for $t < 250$ and $\alpha^{(t)} = 0.8$ for $t \geq 250$. The learning rate η is initially set to 100 and it is updated after every iteration by means of the adaptive learning rate scheme described by Jacobs (1988). A Matlab implementation of the resulting algorithm is available at `http://ticc.uvt.nl/~lvdrmaaten/tsne`.

4. Experiments

To evaluate t-SNE, we present experiments in which t-SNE is compared to seven other non-parametric techniques for dimensionality reduction. Because of space limitations, in the paper, we only compare t-SNE with: (1) Sammon mapping, (2) Isomap, and (3) LLE. In the supporting material, we also compare t-SNE with: (4) CCA, (5) SNE, (6) MVU, and (7) Laplacian Eigenmaps. We performed experiments on five data sets that represent a variety of application domains. Again because of space limitations, we restrict ourselves to three data sets in the paper. The results of our experiments on the remaining two data sets are presented in the supplemental material.

In Section 4.1, the data sets that we employed in our experiments are introduced. The setup of the experiments is presented in Section 4.2. In Section 4.3, we present the results of our experiments.

4.1 Data Sets

The five data sets we employed in our experiments are: (1) the MNIST data set, (2) the Olivetti faces data set, (3) the COIL-20 data set, (4) the word-features data set, and (5) the Netflix data set. We only present results on the first three data sets in this section. The results on the remaining two data sets are presented in the supporting material. The first three data sets are introduced below.

The MNIST data set[6] contains 60,000 grayscale images of handwritten digits. For our experiments, we randomly selected 6,000 of the images for computational reasons. The digit images have $28 \times 28 = 784$ pixels (i.e., dimensions). The Olivetti faces data set[7] consists of images of 40 individuals with small variations in viewpoint, large variations in expression, and occasional addition of glasses. The data set consists of 400 images (10 per individual) of size $92 \times 112 = 10,304$ pixels, and is labeled according to identity. The COIL-20 data set (Nene et al., 1996) contains images of 20

6. The MNIST data set is publicly available from `http://yann.lecun.com/exdb/mnist/index.html`.
7. The Olivetti faces data set is publicly available from `http://mambo.ucsc.edu/psl/olivetti.html`.

different objects viewed from 72 equally spaced orientations, yielding a total of 1,440 images. The images contain $32 \times 32 = 1,024$ pixels.

4.2 Experimental Setup

In all of our experiments, we start by using PCA to reduce the dimensionality of the data to 30. This speeds up the computation of pairwise distances between the datapoints and suppresses some noise without severely distorting the interpoint distances. We then use each of the dimensionality reduction techniques to convert the 30-dimensional representation to a two-dimensional map and we show the resulting map as a scatterplot. For all of the data sets, there is information about the class of each datapoint, but the class information is only used to select a color and/or symbol for the map points. The class information is not used to determine the spatial coordinates of the map points. The coloring thus provides a way of evaluating how well the map preserves the similarities within each class.

The cost function parameter settings we employed in our experiments are listed in Table 1. In the table, $Perp$ represents the perplexity of the conditional probability distribution induced by a Gaussian kernel and k represents the number of nearest neighbors employed in a neighborhood graph. In the experiments with Isomap and LLE, we only visualize datapoints that correspond to vertices in the largest connected component of the neighborhood graph.[8] For the Sammon mapping optimization, we performed Newton's method for 500 iterations.

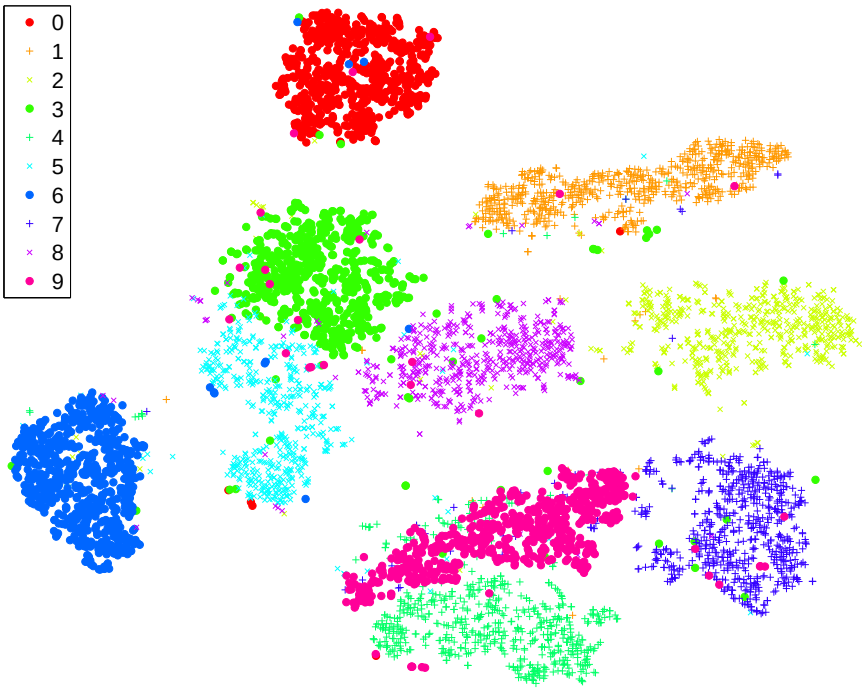
Technique	*Cost function parameters*
t-SNE	$Perp = 40$
Sammon mapping	none
Isomap	$k = 12$
LLE	$k = 12$

Table 1: Cost function parameter settings for the experiments.

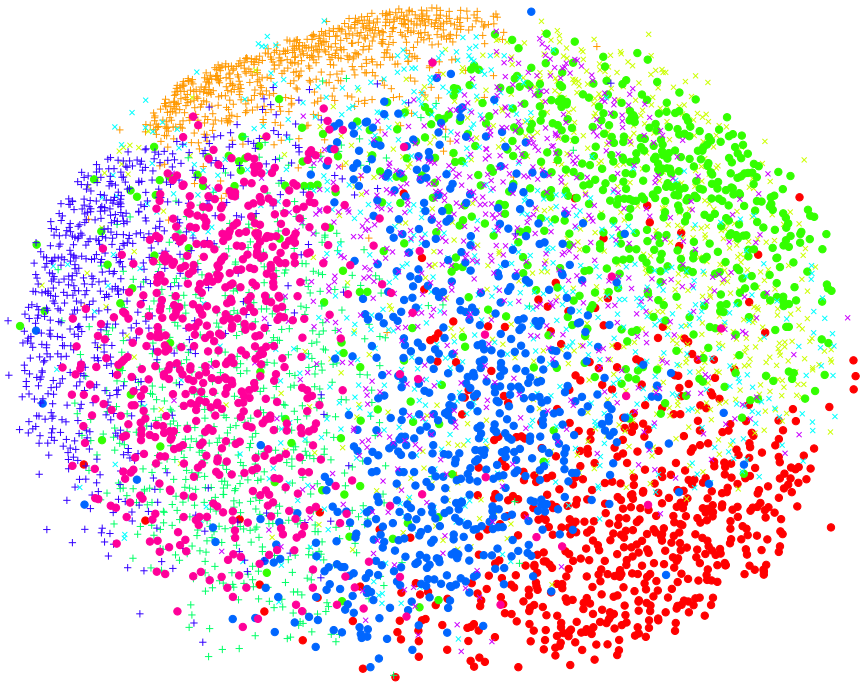
4.3 Results

In Figures 2 and 3, we show the results of our experiments with t-SNE, Sammon mapping, Isomap, and LLE on the MNIST data set. The results reveal the strong performance of t-SNE compared to the other techniques. In particular, Sammon mapping constructs a "ball" in which only three classes (representing the digits 0, 1, and 7) are somewhat separated from the other classes. Isomap and LLE produce solutions in which there are large overlaps between the digit classes. In contrast, t-SNE constructs a map in which the separation between the digit classes is almost perfect. Moreover, detailed inspection of the t-SNE map reveals that much of the local structure of the data (such as the orientation of the ones) is captured as well. This is illustrated in more detail in Section 5 (see Figure 7). The map produced by t-SNE contains some points that are clustered with the wrong class, but most of these points correspond to distorted digits many of which are difficult to identify.
Figure 4 shows the results of applying t-SNE, Sammon mapping, Isomap, and LLE to the Olivetti faces data set. Again, Isomap and LLE produce solutions that provide little insight into the class

8. Isomap and LLE require data that gives rise to a neighborhood graph that is connected.

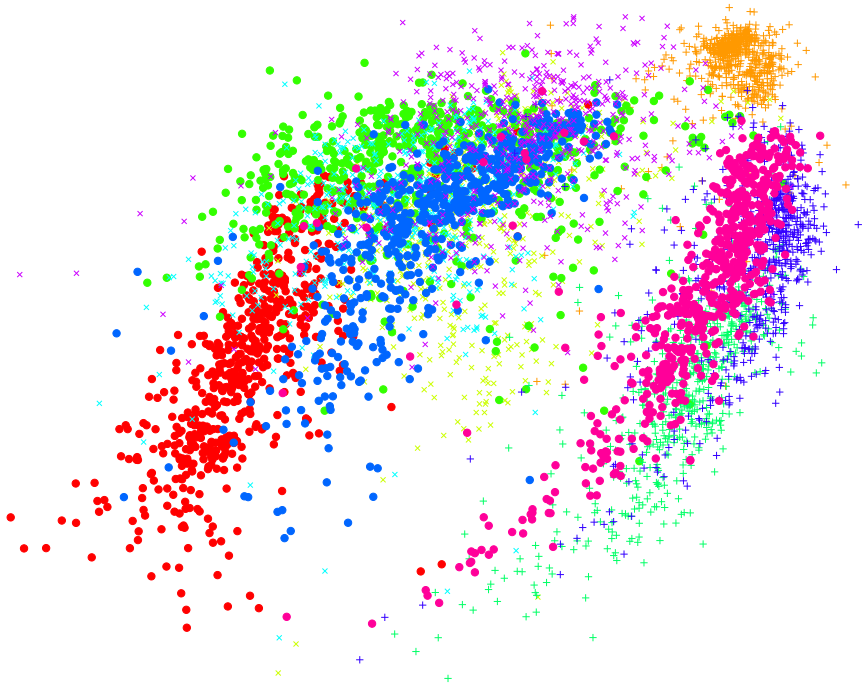


(a) Visualization by t-SNE.

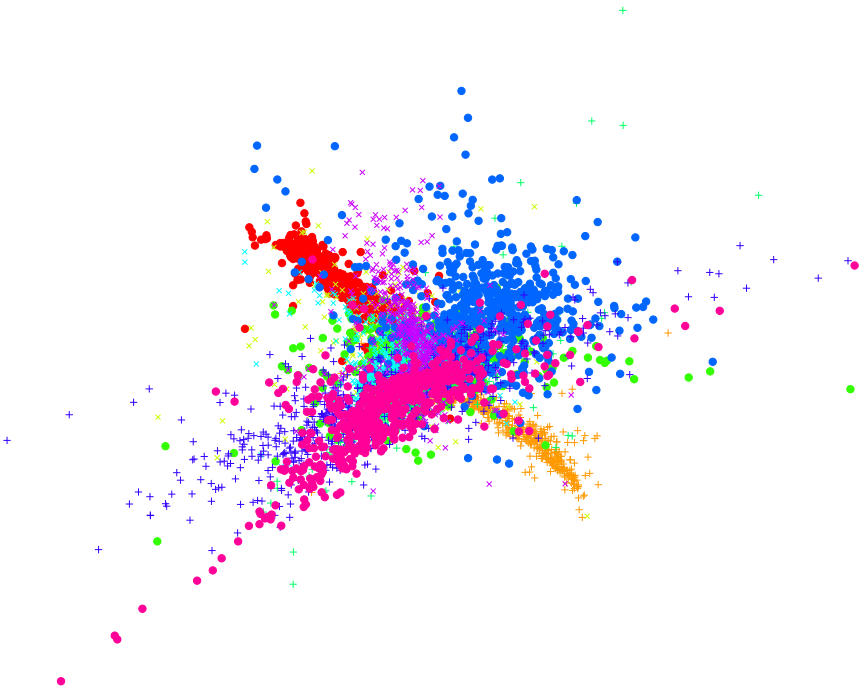


(b) Visualization by Sammon mapping.

Figure 2: Visualizations of 6,000 handwritten digits from the MNIST data set.

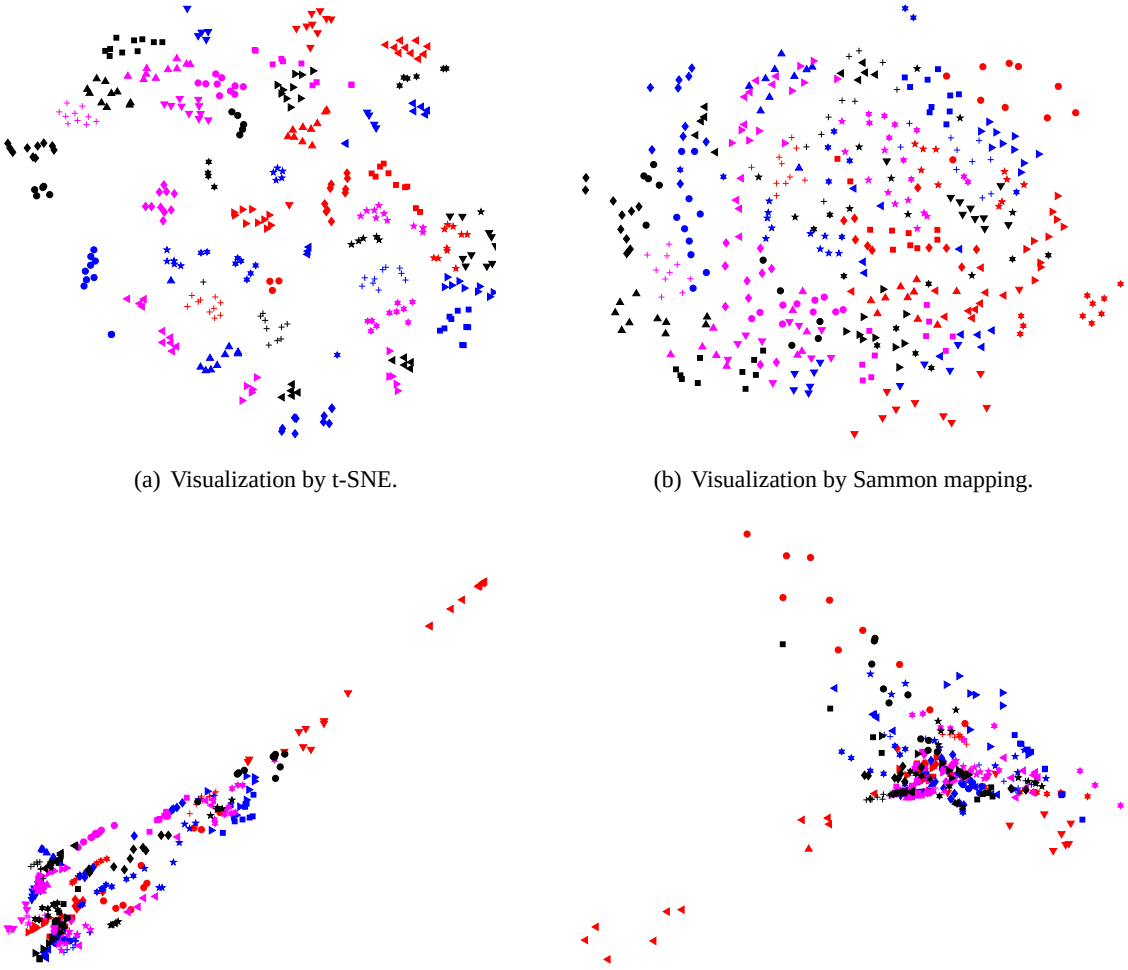


(a) Visualization by Isomap.



(b) Visualization by LLE.

Figure 3: Visualizations of 6,000 handwritten digits from the MNIST data set.



(a) Visualization by t-SNE.

(b) Visualization by Sammon mapping.

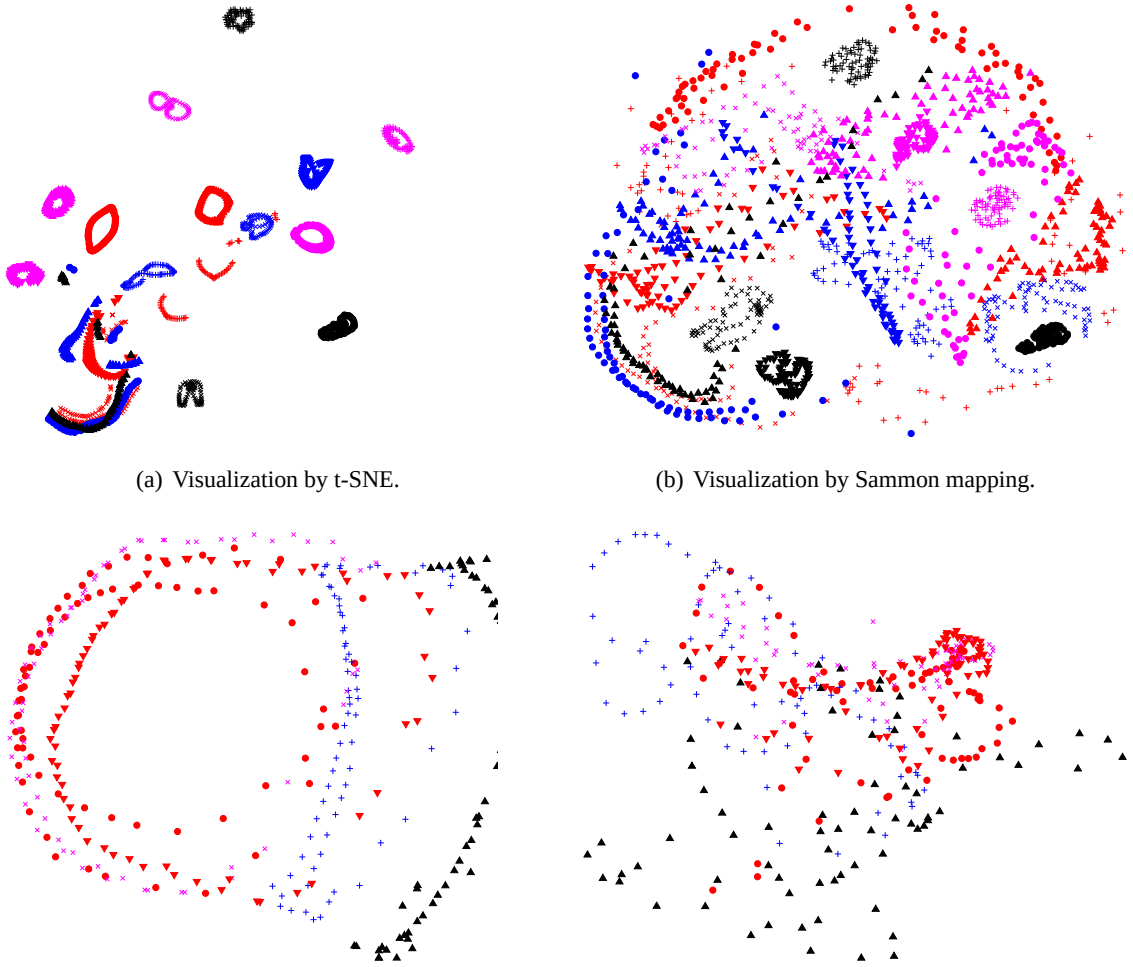
(c) Visualization by Isomap.

(d) Visualization by LLE.

Figure 4: Visualizations of the Olivetti faces data set.

structure of the data. The map constructed by Sammon mapping is significantly better, since it models many of the members of each class fairly close together, but none of the classes are clearly separated in the Sammon map. In contrast, t-SNE does a much better job of revealing the natural classes in the data. Some individuals have their ten images split into two clusters, usually because a subset of the images have the head facing in a significantly different direction, or because they have a very different expression or glasses. For these individuals, it is not clear that their ten images form a natural class when using Euclidean distance in pixel space.

Figure 5 shows the results of applying t-SNE, Sammon mapping, Isomap, and LLE to the COIL-20 data set. For many of the 20 objects, t-SNE accurately represents the one-dimensional manifold of viewpoints as a closed loop. For objects which look similar from the front and the back, t-SNE distorts the loop so that the images of front and back are mapped to nearby points. For the four types of toy car in the COIL-20 data set (the four aligned "sausages" in the bottom-left of the t-SNE map), the four rotation manifolds are aligned by the orientation of the cars to capture the high



(a) Visualization by t-SNE.

(b) Visualization by Sammon mapping.

(c) Visualization by Isomap.

(d) Visualization by LLE.

Figure 5: Visualizations of the COIL-20 data set.

similarity between different cars at the same orientation. This prevents t-SNE from keeping the four manifolds clearly separate. Figure 5 also reveals that the other three techniques are not nearly as good at cleanly separating the manifolds that correspond to very different objects. In addition, Isomap and LLE only visualize a small number of classes from the COIL-20 data set, because the data set comprises a large number of widely separated submanifolds that give rise to small connected components in the neighborhood graph.

5. Applying t-SNE to Large Data Sets

Like many other visualization techniques, t-SNE has a computational and memory complexity that is quadratic in the number of datapoints. This makes it infeasible to apply the standard version of t-SNE to data sets that contain many more than, say, 10,000 points. Obviously, it is possible to pick a random subset of the datapoints and display them using t-SNE, but such an approach fails to

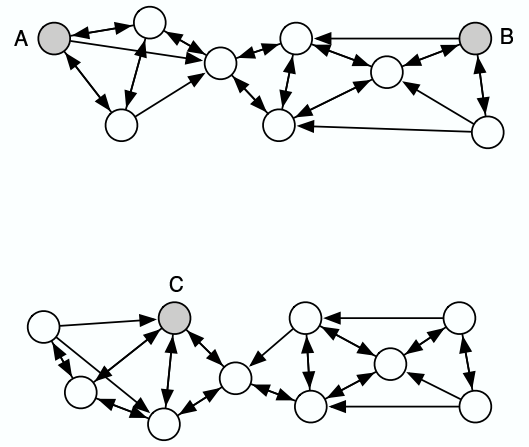


Figure 6: An illustration of the advantage of the random walk version of t-SNE over a standard landmark approach. The shaded points A, B, and C are three (almost) equidistant landmark points, whereas the non-shaded datapoints are non-landmark points. The arrows represent a directed neighborhood graph where $k = 3$. In a standard landmark approach, the pairwise affinity between A and B is approximately equal to the pairwise affinity between A and C. In the random walk version of t-SNE, the pairwise affinity between A and B is much larger than the pairwise affinity between A and C, and therefore, it reflects the structure of the data much better.

make use of the information that the undisplayed datapoints provide about the underlying manifolds. Suppose, for example, that A, B, and C are all equidistant in the high-dimensional space. If there are many undisplayed datapoints between A and B and none between A and C, it is much more likely that A and B are part of the same cluster than A and C. This is illustrated in Figure 6. In this section, we show how t-SNE can be modified to display a random subset of the datapoints (so-called landmark points) in a way that uses information from the entire (possibly very large) data set.

We start by choosing a desired number of neighbors and creating a neighborhood graph for all of the datapoints. Although this is computationally intensive, it is only done once. Then, for each of the landmark points, we define a random walk starting at that landmark point and terminating as soon as it lands on another landmark point. During a random walk, the probability of choosing an edge emanating from node x_i to node x_j is proportional to $e^{-\|x_i - x_j\|^2}$. We define $p_{j|i}$ to be the fraction of random walks starting at landmark point x_i that terminate at landmark point x_j. This has some resemblance to the way Isomap measures pairwise distances between points. However, as in diffusion maps (Lafon and Lee, 2006; Nadler et al., 2006), rather than looking for the shortest path through the neighborhood graph, the random walk-based affinity measure integrates over all paths through the neighborhood graph. As a result, the random walk-based affinity measure is much less sensitive to "short-circuits" (Lee and Verleysen, 2005), in which a single noisy datapoint provides a bridge between two regions of dataspace that should be far apart in the map. Similar approaches using random walks have also been successfully applied to, for example, semi-supervised learning (Szummer and Jaakkola, 2001; Zhu et al., 2003) and image segmentation (Grady, 2006).

The most obvious way to compute the random walk-based similarities $p_{j|i}$ is to explicitly perform the random walks on the neighborhood graph, which works very well in practice, given that one can easily perform one million random walks per second. Alternatively, Grady (2006) presents an analytical solution to compute the pairwise similarities $p_{j|i}$ that involves solving a sparse linear system. The analytical solution to compute the similarities $p_{j|i}$ is sketched in Appendix B. In preliminary experiments, we did not find significant differences between performing the random walks explicitly and the analytical solution. In the experiment we present below, we explicitly performed the random walks because this is computationally less expensive. However, for very large data sets in which the landmark points are very sparse, the analytical solution may be more appropriate.

Figure 7 shows the results of an experiment, in which we applied the random walk version of t-SNE to 6,000 randomly selected digits from the MNIST data set, using all 60,000 digits to compute the pairwise affinities $p_{j|i}$. In the experiment, we used a neighborhood graph that was constructed using a value of $k = 20$ nearest neighbors.[9] The inset of the figure shows the same visualization as a scatterplot in which the colors represent the labels of the digits. In the t-SNE map, all classes are clearly separated and the "continental" sevens form a small separate cluster. Moreover, t-SNE reveals the main dimensions of variation within each class, such as the orientation of the ones, fours, sevens, and nines, or the "loopiness" of the twos. The strong performance of t-SNE is also reflected in the generalization error of nearest neighbor classifiers that are trained on the low-dimensional representation. Whereas the generalization error (measured using 10-fold cross validation) of a 1-nearest neighbor classifier trained on the original 784-dimensional datapoints is 5.75%, the generalization error of a 1-nearest neighbor classifier trained on the two-dimensional data representation produced by t-SNE is only 5.13%. The computational requirements of random walk t-SNE are reasonable: it took only one hour of CPU time to construct the map in Figure 7.

6. Discussion

The results in the previous two sections (and those in the supplemental material) demonstrate the performance of t-SNE on a wide variety of data sets. In this section, we discuss the differences between t-SNE and other non-parametric techniques (Section 6.1), and we also discuss a number of weaknesses and possible improvements of t-SNE (Section 6.2).

6.1 Comparison with Related Techniques

Classical scaling (Torgerson, 1952), which is closely related to PCA (Mardia et al., 1979; Williams, 2002), finds a linear transformation of the data that minimizes the sum of the squared errors between high-dimensional pairwise distances and their low-dimensional representatives. A linear method such as classical scaling is not good at modeling curved manifolds and it focuses on preserving the distances between widely separated datapoints rather than on preserving the distances between nearby datapoints. An important approach that attempts to address the problems of classical scaling is the Sammon mapping (Sammon, 1969) which alters the cost function of classical scaling by dividing the squared error in the representation of each pairwise Euclidean distance by the original Euclidean distance in the high-dimensional space. The resulting cost function is given by

$$C = \frac{1}{\sum_{ij} \|x_i - x_j\|} \sum_{i \neq j} \frac{(\|x_i - x_j\| - \|y_i - y_j\|)^2}{\|x_i - x_j\|},$$

9. In preliminary experiments, we found the performance of random walk t-SNE to be very robust under changes of k.

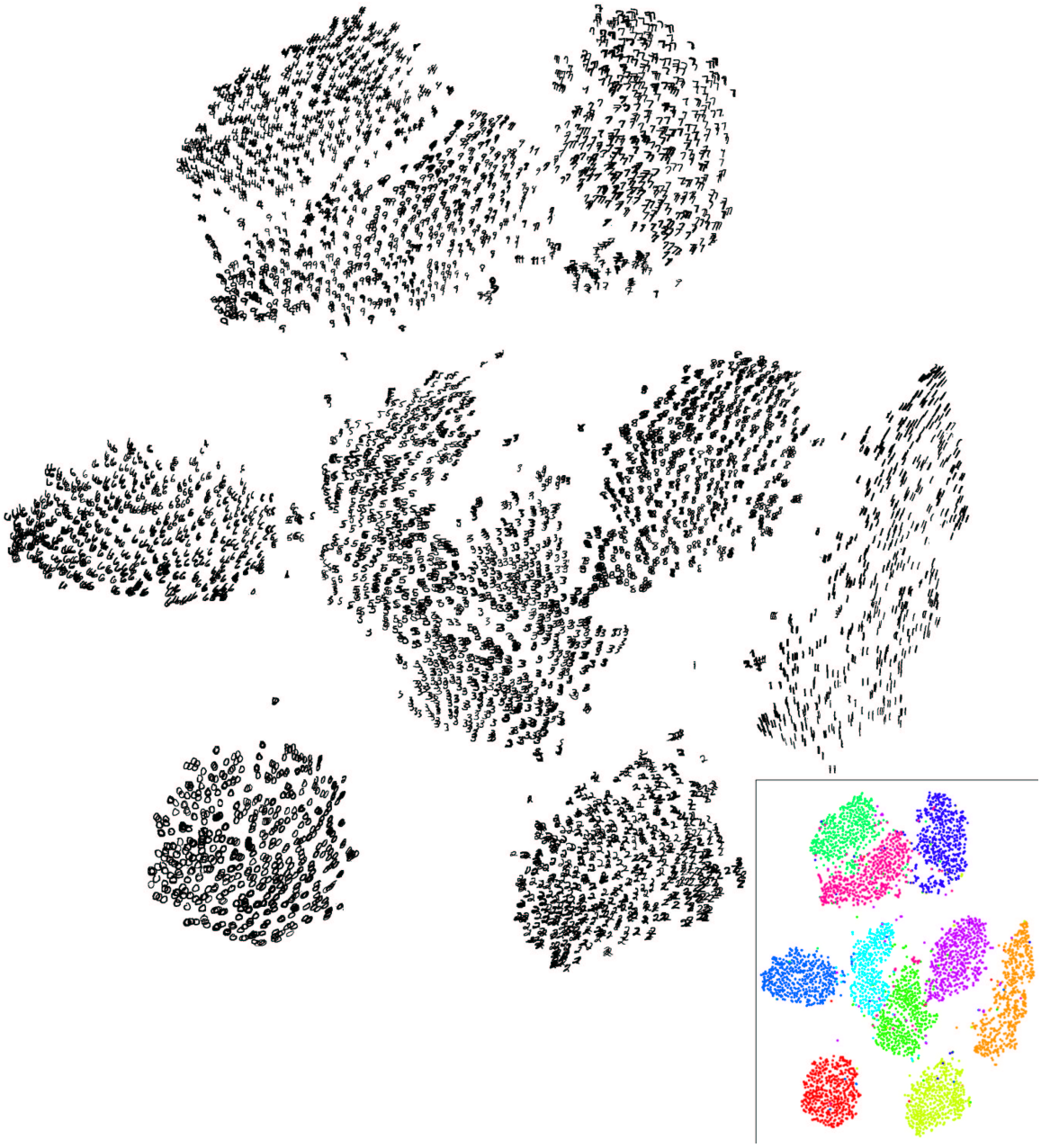


Figure 7: Visualization of 6,000 digits from the MNIST data set produced by the random walk version of t-SNE (employing all 60,000 digit images).

where the constant outside of the sum is added in order to simplify the derivation of the gradient. The main weakness of the Sammon cost function is that the importance of retaining small pairwise distances in the map is largely dependent on small differences in these pairwise distances. In particular, a small error in the model of two high-dimensional points that are extremely close together results in a large contribution to the cost function. Since all small pairwise distances constitute the local structure of the data, it seems more appropriate to aim to assign approximately equal importance to all small pairwise distances.

In contrast to Sammon mapping, the Gaussian kernel employed in the high-dimensional space by t-SNE defines a soft border between the local and global structure of the data and for pairs of datapoints that are close together relative to the standard deviation of the Gaussian, the importance of modeling their separations is almost independent of the magnitudes of those separations. Moreover, t-SNE determines the local neighborhood size for each datapoint separately based on the local density of the data (by forcing each conditional probability distribution P_i to have the same perplexity).

The strong performance of t-SNE compared to Isomap is partly explained by Isomap's susceptibility to "short-circuiting". Also, Isomap mainly focuses on modeling large geodesic distances rather than small ones.

The strong performance of t-SNE compared to LLE is mainly due to a basic weakness of LLE: the only thing that prevents all datapoints from collapsing onto a single point is a constraint on the covariance of the low-dimensional representation. In practice, this constraint is often satisfied by placing most of the map points near the center of the map and using a few widely scattered points to create large covariance (see Figure 3(b) and 4(d)). For neighborhood graphs that are almost disconnected, the covariance constraint can also be satisfied by a "curdled" map in which there are a few widely separated, collapsed subsets corresponding to the almost disconnected components. Furthermore, neighborhood-graph based techniques (such as Isomap and LLE) are not capable of visualizing data that consists of two or more widely separated submanifolds, because such data does not give rise to a connected neighborhood graph. It is possible to produce a separate map for each connected component, but this loses information about the relative similarities of the separate components.

Like Isomap and LLE, the random walk version of t-SNE employs neighborhood graphs, but it does not suffer from short-circuiting problems because the pairwise similarities between the high-dimensional datapoints are computed by integrating over all paths through the neighborhood graph. Because of the diffusion-based interpretation of the conditional probabilities underlying the random walk version of t-SNE, it is useful to compare t-SNE to diffusion maps. Diffusion maps define a "diffusion distance" on the high-dimensional datapoints that is given by

$$D^{(t)}(x_i, x_j) = \sqrt{\sum_k \frac{\left(p_{ik}^{(t)} - p_{jk}^{(t)}\right)^2}{\psi(x_k)^{(0)}}},$$

where $p_{ij}^{(t)}$ represents the probability of a particle traveling from x_i to x_j in t timesteps through a graph on the data with Gaussian emission probabilities. The term $\psi(x_k)^{(0)}$ is a measure for the local density of the points, and serves a similar purpose to the fixed perplexity Gaussian kernel that is employed in SNE. The diffusion map is formed by the principal non-trivial eigenvectors of the Markov matrix of the random walks of length t. It can be shown that when all $(n-1)$ non-trivial eigenvec-

tors are employed, the Euclidean distances in the diffusion map are equal to the diffusion distances in the high-dimensional data representation (Lafon and Lee, 2006). Mathematically, diffusion maps minimize

$$C = \sum_i \sum_j \left(D^{(t)}(x_i, x_j) - \|y_i - y_j\| \right)^2.$$

As a result, diffusion maps are susceptible to the same problems as classical scaling: they assign much higher importance to modeling the large pairwise diffusion distances than the small ones and as a result, they are not good at retaining the local structure of the data. Moreover, in contrast to the random walk version of t-SNE, diffusion maps do not have a natural way of selecting the length, t, of the random walks.

In the supplemental material, we present results that reveal that t-SNE outperforms CCA (Demartines and Hérault, 1997), MVU (Weinberger et al., 2004), and Laplacian Eigenmaps (Belkin and Niyogi, 2002) as well. For CCA and the closely related CDA (Lee et al., 2000), these results can be partially explained by the hard border λ that these techniques define between local and global structure, as opposed to the soft border of t-SNE. Moreover, within the range λ, CCA suffers from the same weakness as Sammon mapping: it assigns extremely high importance to modeling the distance between two datapoints that are extremely close.

Like t-SNE, MVU (Weinberger et al., 2004) tries to model all of the small separations well but MVU insists on modeling them perfectly (i.e., it treats them as constraints) and a single erroneous constraint may severely affect the performance of MVU. This can occur when there is a short-circuit between two parts of a curved manifold that are far apart in the intrinsic manifold coordinates. Also, MVU makes no attempt to model longer range structure: It simply pulls the map points as far apart as possible subject to the hard constraints so, unlike t-SNE, it cannot be expected to produce sensible large-scale structure in the map.

For Laplacian Eigenmaps, the poor results relative to t-SNE may be explained by the fact that Laplacian Eigenmaps have the same covariance constraint as LLE, and it is easy to cheat on this constraint.

6.2 Weaknesses

Although we have shown that t-SNE compares favorably to other techniques for data visualization, t-SNE has three potential weaknesses: (1) it is unclear how t-SNE performs on general dimensionality reduction tasks, (2) the relatively local nature of t-SNE makes it sensitive to the curse of the intrinsic dimensionality of the data, and (3) t-SNE is not guaranteed to converge to a global optimum of its cost function. Below, we discuss the three weaknesses in more detail.

1) Dimensionality reduction for other purposes. It is not obvious how t-SNE will perform on the more general task of dimensionality reduction (i.e., when the dimensionality of the data is not reduced to two or three, but to $d > 3$ dimensions). To simplify evaluation issues, this paper only considers the use of t-SNE for data visualization. The behavior of t-SNE when reducing data to two or three dimensions cannot readily be extrapolated to $d > 3$ dimensions because of the heavy tails of the Student-t distribution. In high-dimensional spaces, the heavy tails comprise a relatively large portion of the probability mass under the Student-t distribution, which might lead to d-dimensional data representations that do not preserve the local structure of the data as well. Hence, for tasks

in which the dimensionality of the data needs to be reduced to a dimensionality higher than three, Student t-distributions with more than one degree of freedom[10] are likely to be more appropriate.

2) Curse of intrinsic dimensionality. t-SNE reduces the dimensionality of data mainly based on local properties of the data, which makes t-SNE sensitive to the curse of the intrinsic dimensionality of the data (Bengio, 2007). In data sets with a high intrinsic dimensionality and an underlying manifold that is highly varying, the local linearity assumption on the manifold that t-SNE implicitly makes (by employing Euclidean distances between near neighbors) may be violated. As a result, t-SNE might be less successful if it is applied on data sets with a very high intrinsic dimensionality (for instance, a recent study by Meytlis and Sirovich (2007) estimates the space of images of faces to be constituted of approximately 100 dimensions). Manifold learners such as Isomap and LLE suffer from exactly the same problems (see, e.g., Bengio, 2007; van der Maaten et al., 2008). A possible way to (partially) address this issue is by performing t-SNE on a data representation obtained from a model that represents the highly varying data manifold efficiently in a number of nonlinear layers such as an autoencoder (Hinton and Salakhutdinov, 2006). Such deep-layer architectures can represent complex nonlinear functions in a much simpler way, and as a result, require fewer datapoints to learn an appropriate solution (as is illustrated for a d-bits parity task by Bengio 2007). Performing t-SNE on a data representation produced by, for example, an autoencoder is likely to improve the quality of the constructed visualizations, because autoencoders can identify highly-varying manifolds better than a local method such as t-SNE. However, the reader should note that it is by definition impossible to fully represent the structure of intrinsically high-dimensional data in two or three dimensions.

3) Non-convexity of the t-SNE cost function. A nice property of most state-of-the-art dimensionality reduction techniques (such as classical scaling, Isomap, LLE, and diffusion maps) is the convexity of their cost functions. A major weakness of t-SNE is that the cost function is not convex, as a result of which several optimization parameters need to be chosen. The constructed solutions depend on these choices of optimization parameters and may be different each time t-SNE is run from an initial random configuration of map points. We have demonstrated that the same choice of optimization parameters can be used for a variety of different visualization tasks, and we found that the quality of the optima does not vary much from run to run. Therefore, we think that the weakness of the optimization method is insufficient reason to reject t-SNE in favor of methods that lead to convex optimization problems but produce noticeably worse visualizations. A local optimum of a cost function that accurately captures what we want in a visualization is often preferable to the global optimum of a cost function that fails to capture important aspects of what we want. Moreover, the convexity of cost functions can be misleading, because their optimization is often computationally infeasible for large real-world data sets, prompting the use of approximation techniques (de Silva and Tenenbaum, 2003; Weinberger et al., 2007). Even for LLE and Laplacian Eigenmaps, the optimization is performed using iterative Arnoldi (Arnoldi, 1951) or Jacobi-Davidson (Fokkema et al., 1999) methods, which may fail to find the global optimum due to convergence problems.

7. Conclusions

The paper presents a new technique for the visualization of similarity data that is capable of retaining the local structure of the data while also revealing some important global structure (such as clusters

10. Increasing the degrees of freedom of a Student-t distribution makes the tails of the distribution lighter. With infinite degrees of freedom, the Student-t distribution is equal to the Gaussian distribution.

at multiple scales). Both the computational and the memory complexity of t-SNE are $O(n^2)$, but we present a landmark approach that makes it possible to successfully visualize large real-world data sets with limited computational demands. Our experiments on a variety of data sets show that t-SNE outperforms existing state-of-the-art techniques for visualizing a variety of real-world data sets. Matlab implementations of both the normal and the random walk version of t-SNE are available for download at `http://ticc.uvt.nl/~lvdrmaaten/tsne`.

In future work we plan to investigate the optimization of the number of degrees of freedom of the Student-t distribution used in t-SNE. This may be helpful for dimensionality reduction when the low-dimensional representation has many dimensions. We will also investigate the extension of t-SNE to models in which each high-dimensional datapoint is modeled by several low-dimensional map points as in Cook et al. (2007). Also, we aim to develop a parametric version of t-SNE that allows for generalization to held-out test data by using the t-SNE objective function to train a multilayer neural network that provides an explicit mapping to the low-dimensional space.

Acknowledgments

The authors thank Sam Roweis for many helpful discussions, Andriy Mnih for supplying the word-features data set, Ruslan Salakhutdinov for help with the Netflix data set (results for these data sets are presented in the supplemental material), and Guido de Croon for pointing us to the analytical solution of the random walk probabilities.

Laurens van der Maaten is supported by the CATCH-programme of the Dutch Scientific Organization (NWO), project RICH (grant 640.002.401), and cooperates with RACM. Geoffrey Hinton is a fellow of the Canadian Institute for Advanced Research, and is also supported by grants from NSERC and CFI and gifts from Google and Microsoft.

Appendix A. Derivation of the t-SNE gradient

t-SNE minimizes the Kullback-Leibler divergence between the joint probabilities p_{ij} in the high-dimensional space and the joint probabilities q_{ij} in the low-dimensional space. The values of p_{ij} are defined to be the symmetrized conditional probabilities, whereas the values of q_{ij} are obtained by means of a Student-t distribution with one degree of freedom

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n},$$

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}},$$

where $p_{j|i}$ and $p_{i|j}$ are either obtained from Equation 1 or from the random walk procedure described in Section 5. The values of p_{ii} and q_{ii} are set to zero. The Kullback-Leibler divergence between the two joint probability distributions P and Q is given by

$$\begin{aligned} C = KL(P||Q) &= \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \\ &= \sum_i \sum_j p_{ij} \log p_{ij} - p_{ij} \log q_{ij}. \end{aligned} \tag{6}$$

In order to make the derivation less cluttered, we define two auxiliary variables d_{ij} and Z as follows

$$d_{ij} = \|y_i - y_j\|,$$
$$Z = \sum_{k \neq l} (1 + d_{kl}^2)^{-1}.$$

Note that if y_i changes, the only pairwise distances that change are d_{ij} and d_{ji} for $\forall j$. Hence, the gradient of the cost function C with respect to y_i is given by

$$\begin{aligned} \frac{\delta C}{\delta y_i} &= \sum_j \left(\frac{\delta C}{\delta d_{ij}} + \frac{\delta C}{\delta d_{ji}} \right) (y_i - y_j) \\ &= 2 \sum_j \frac{\delta C}{\delta d_{ij}} (y_i - y_j). \end{aligned} \tag{7}$$

The gradient $\frac{\delta C}{\delta d_{ij}}$ is computed from the definition of the Kullback-Leibler divergence in Equation 6 (note that the first part of this equation is a constant).

$$\begin{aligned} \frac{\delta C}{\delta d_{ij}} &= -\sum_{k \neq l} p_{kl} \frac{\delta (\log q_{kl})}{\delta d_{ij}} \\ &= -\sum_{k \neq l} p_{kl} \frac{\delta (\log q_{kl} Z - \log Z)}{\delta d_{ij}} \\ &= -\sum_{k \neq l} p_{kl} \left(\frac{1}{q_{kl} Z} \frac{\delta ((1 + d_{kl}^2)^{-1})}{\delta d_{ij}} - \frac{1}{Z} \frac{\delta Z}{\delta d_{ij}} \right) \end{aligned}$$

The gradient $\frac{\delta ((1+d_{kl}^2)^{-1})}{\delta d_{ij}}$ is only nonzero when $k = i$ and $l = j$. Hence, the gradient $\frac{\delta C}{\delta d_{ij}}$ is given by

$$\frac{\delta C}{\delta d_{ij}} = 2 \frac{p_{ij}}{q_{ij} Z} (1 + d_{ij}^2)^{-2} - 2 \sum_{k \neq l} p_{kl} \frac{(1 + d_{ij}^2)^{-2}}{Z}.$$

Noting that $\sum_{k \neq l} p_{kl} = 1$, we see that the gradient simplifies to

$$\begin{aligned} \frac{\delta C}{\delta d_{ij}} &= 2 p_{ij} (1 + d_{ij}^2)^{-1} - 2 q_{ij} (1 + d_{ij}^2)^{-1} \\ &= 2 (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1}. \end{aligned}$$

Substituting this term into Equation 7, we obtain the gradient

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij}) (1 + \|y_i - y_j\|^2)^{-1} (y_i - y_j).$$

Appendix B. Analytical Solution to Random Walk Probabilities

Below, we describe the analytical solution to the random walk probabilities that are employed in the random walk version of t-SNE (see Section 5). The solution is described in more detail by Grady (2006).

It can be shown that computing the probability that a random walk initiated from a non-landmark point (on a graph that is specified by adjacency matrix W) first reaches a specific landmark point is equal to computing the solution to the combinatorial Dirichlet problem in which the boundary conditions are at the locations of the landmark points, the considered landmark point is fixed to unity, and the other landmarks points are set to zero (Kakutani, 1945; Doyle and Snell, 1984). In practice, the solution can thus be obtained by minimizing the combinatorial formulation of the Dirichlet integral

$$D[x] = \frac{1}{2} x^T L x,$$

where L represents the graph Laplacian. Mathematically, the graph Laplacian is given by $L = D - W$, where $D = \text{diag}\left(\sum_j w_{1j}, \sum_j w_{2j}, ..., \sum_j w_{nj}\right)$. Without loss of generality, we may reorder the landmark points such that the landmark points come first. As a result, the combinatorial Dirichlet integral decomposes into

$$\begin{aligned} D[x_N] &= \frac{1}{2} \begin{bmatrix} x_L^T & x_N^T \end{bmatrix} \begin{bmatrix} L_L & B \\ B^T & L_N \end{bmatrix} \begin{bmatrix} x_L \\ x_N \end{bmatrix} \\ &= \frac{1}{2} \left(x_L^T L_L x_L + 2 x_N^T B^T x_M + x_N^T L_N x_N \right), \end{aligned}$$

where we use the subscript $\cdot_L$ to indicate the landmark points, and the subscript $\cdot_N$ to indicate the non-landmark points. Differentiating $D[x_N]$ with respect to x_N and finding its critical points amounts to solving the linear systems

$$L_N x_N = -B^T. \tag{8}$$

Please note that in this linear system, B^T is a matrix containing the columns from the graph Laplacian L that correspond to the landmark points (excluding the rows that correspond to landmark points). After normalization of the solutions to the systems X_N, the column vectors of X_N contain the probability that a random walk initiated from a non-landmark point terminates in a landmark point. One should note that the linear system in Equation 8 is only nonsingular if the graph is completely connected, or if each connected component in the graph contains at least one landmark point (Biggs, 1974).

Because we are interested in the probability of a random walk initiated from a *landmark point* terminating at another landmark point, we duplicate all landmark points in the neighborhood graph, and initiate the random walks from the duplicate landmarks. Because of memory constraints, it is not possible to store the entire matrix X_N into memory (note that we are only interested in a small number of rows from this matrix, viz., in the rows corresponding to the duplicate landmark points). Hence, we solve the linear systems defined by the columns of $-B^T$ one-by-one, and store only the parts of the solutions that correspond to the duplicate landmark points. For computational reasons, we first perform a Cholesky factorization of L_N, such that $L_N = CC^T$, where C is an upper-triangular matrix. Subsequently, the solution to the linear system in Equation 8 is obtained by solving the linear systems $Cy = -B^T$ and $Cx_N = y$ using a fast backsubstitution method.

References

W.E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9:17–25, 1951.

G.D. Battista, P. Eades, R. Tamassia, and I.G. Tollis. Annotated bibliography on graph drawing. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.

M. Belkin and P. Niyogi. Laplacian Eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems*, volume 14, pages 585–591, Cambridge, MA, USA, 2002. The MIT Press.

Y. Bengio. Learning deep architectures for AI. Technical Report 1312, Université de Montréal, 2007.

N. Biggs. Algebraic graph theory. In *Cambridge Tracts in Mathematics*, volume 67. Cambridge University Press, 1974.

H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68:361–368, 1973.

J.A. Cook, I. Sutskever, A. Mnih, and G.E. Hinton. Visualizing similarity data with a mixture of maps. In *Proceedings of the* 11^{th} *International Conference on Artificial Intelligence and Statistics*, volume 2, pages 67–74, 2007.

M.C. Ferreira de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):378–394, 2003.

V. de Silva and J.B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems*, volume 15, pages 721–728, Cambridge, MA, USA, 2003. The MIT Press.

P. Demartines and J. Hérault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1):148–154, 1997.

P. Doyle and L. Snell. Random walks and electric networks. In *Carus mathematical monographs*, volume 22. Mathematical Association of America, 1984.

D.R. Fokkema, G.L.G. Sleijpen, and H.A. van der Vorst. Jacobi–Davidson style QR and QZ algorithms for the reduction of matrix pencils. *SIAM Journal on Scientific Computing*, 20(1):94–125, 1999.

L. Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, 2006.

G.E. Hinton and S.T. Roweis. Stochastic Neighbor Embedding. In *Advances in Neural Information Processing Systems*, volume 15, pages 833–840, Cambridge, MA, USA, 2002. The MIT Press.

G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 1933.

R.A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1: 295–307, 1988.

S. Kakutani. Markov processes and the Dirichlet problem. *Proceedings of the Japan Academy*, 21: 227–233, 1945.

D.A. Keim. Designing pixel-oriented visualization techniques: Theory and applications. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):59–78, 2000.

S. Lafon and A.B. Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1393–1403, 2006.

J.A. Lee and M. Verleysen. Nonlinear dimensionality reduction of data manifolds with essential loops. *Neurocomputing*, 67:29–53, 2005.

J.A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer, New York, NY, USA, 2007.

J.A. Lee, A. Lendasse, N. Donckers, and M. Verleysen. A robust nonlinear projection method. In *Proceedings of the* 8^{th} *European Symposium on Artificial Neural Networks*, pages 13–20, 2000.

K.V. Mardia, J.T. Kent, and J.M. Bibby. *Multivariate Analysis*. Academic Press, 1979.

M. Meytlis and L. Sirovich. On the dimensionality of face space. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 29(7):1262–1267, 2007.

B. Nadler, S. Lafon, R.R. Coifman, and I.G. Kevrekidis. Diffusion maps, spectral clustering and the reaction coordinates of dynamical systems. *Applied and Computational Harmonic Analysis: Special Issue on Diffusion Maps and Wavelets*, 21:113–127, 2006.

S.A. Nene, S.K. Nayar, and H. Murase. Columbia Object Image Library (COIL-20). Technical Report CUCS-005-96, Columbia University, 1996.

S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, 2000.

J.W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18(5):401–409, 1969.

L. Song, A.J. Smola, K. Borgwardt, and A. Gretton. Colored Maximum Variance Unfolding. In *Advances in Neural Information Processing Systems*, volume 21 (in press), 2007.

W.N. Street, W.H. Wolberg, and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *Proceedings of the IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology*, volume 1905, pages 861–870, 1993.

M. Szummer and T. Jaakkola. Partially labeled classification with Markov random walks. In *Advances in Neural Information Processing Systems*, volume 14, pages 945–952, 2001.

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

W.S. Torgerson. Multidimensional scaling I: Theory and method. *Psychometrika*, 17:401–419, 1952.

L.J.P. van der Maaten, E.O. Postma, and H.J. van den Herik. Dimensionality reduction: A comparative review. Online Preprint, 2008.

K.Q. Weinberger, F. Sha, and L.K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the* 21^{st} *International Confernence on Machine Learning*, 2004.

K.Q. Weinberger, F. Sha, Q. Zhu, and L.K. Saul. Graph Laplacian regularization for large-scale semidefinite programming. In *Advances in Neural Information Processing Systems*, volume 19, 2007.

C.K.I. Williams. On a connection between Kernel PCA and metric multidimensional scaling. *Machine Learning*, 46(1-3):11–19, 2002.

X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the* 20^{th} *International Conference on Machine Learning*, pages 912–919, 2003.