Erasmus University Rotterdam

Erasmus School of Economics

Obtaining a Parametric Mapping for Dimensionality Reduction by using t-SNE in a Regularized Autoencoder Network

A thesis submitted in partial fulfillment for the degree of

Bachelor Econometrie en Operationele Research

Stefan Lam

Student ID: 481922

Thesis supervisor: prof. dr. Patrick Groenen

Second assessor: Anoek Castelein

July 5, 2020

Abstract

In this paper, we extend parametric t-distributed Stochastic Neighbor Embedding (t-SNE) as introduced by Van Der Maaten (2009). In particular, we combine the abilities of autoencoders and parametric t-SNE into one efficient autoencoder network by introducing Regularized Parametric t-SNE (RP t-SNE), a technique that learns a parametric mapping for dimensionality reduction able to preserve the local structure of data in low-dimensional projections, and simultaneously learns to reconstruct these projections with high precision. We evaluate the performance of RP t-SNE by comparing it with Principal Component Analysis (PCA), kernel t-SNE, autoencoders and parametric t-SNE in terms of generalization and reconstruction performance on the MNIST and COIL-20 image datasets. Results show that RP t-SNE provides similar reconstruction and generalization performances as autoencoders and parametric t-SNE respectively. Moreover, RP t-SNE is able to reconstruct projections more precisely than autoencoders for smaller sample sizes.

Contents

1 Introduction

Real world data is often represented in a high-dimensional space, and manipulation in this space is difficult because of the so-called "curse of dimensionality" (Schubert and Gertz, 2017), which states that the amount of data needed to obtain a statistically sound and reliable result increases exponentially with the dimensionality. Typically, the number of dimensions needed to represent all the information in the input data, the intrinsic dimensionality (Schubert and Gertz, 2017), is often lower than the actual dimensionality of the high-dimensional input data. Hence, dimensionality reduction techniques are able to alleviate the curse of dimensionality by finding low-dimensional projections of high-dimensional datapoints from the input data. Solving this curse is important in many computer vision and machine learning applications, both as a preprocessing step for other algorithms and as a goal for interpolation, data compression or visualization. Many dimensionality reduction techniques are thus proposed with the aim to preserve as much of the significant structure from the high-dimensional data as possible. In particular, we focus on t-distributed Stochastic Neighbor Embedding (t-SNE) as introduced by Maaten and Hinton (2008), which is a nonlinear dimensionality reduction technique mainly used for data visualizations. The visualizations obtained from t-SNE distinguish different clusters from high-dimensional data fairly well, since t-SNE is able to preserve the local structure of data.

However, t-SNE, just like many other nonlinear dimensionality reduction techniques are non-parametric, that is, they do not provide an explicit parametric mapping $f : \mathbf{X} \to \mathbf{Y}$ between the high-dimensional input space $\mathbf{X} = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbb{R}^D, i = 1, ..., n\}$ and the low-dimensional projection space $\mathbf{Y} = \{\mathbf{y}_i : \mathbf{y}_i \in \mathbb{R}^d, i = 1, ..., n\}$, where $D > d$ and d is usually equal to one or two. One benefit from the lack of a parametric mapping is the flexibility of visualization that the technique could provide, since no constraints have to be met due to a predefined form of the mapping. Though, this flexibility comes with the drawback that a direct way to map new datapoints on demand does not exist after having computed the projection space. This drawback causes non-parametric dimensionality reduction techniques to be unsuitable for streaming applications, where the full dataset is often not available a priori. Parametric mappings for out-of-sample extension of dimensionality reduction techniques are thus in high demand.

Ideally, the parametric mapping is able to project nonlinear input data into a lower dimension. As an example, Principal Component Analysis (PCA) (Hotelling, 1933) is a parametric technique, but due to its linear nature it is unable to successfully project data which lays on a nonlinear manifold. For this reason, we specifically focus on parameterizing t-SNE, since this is a nonlinear dimensionality reduction technique outperforming many existing dimensionality reduction techniques as will be discussed in Section 2.2. In particular, Van Der Maaten (2009) already described a way to obtain a parametric mapping for dimensionality reduction with t-SNE by introducing parametric t-SNE, a technique which trains an encoder network by minimizing the t-SNE cost function. Though, another popular approach to obtain nonlinear parametric mappings is by training autoencoder networks (Hinton and Salakhutdinov, 2006), which minimizes the reconstruction loss. Both these neural networks have their own abilities: autoencoders have the ability to reconstruct the input data from their projections, while parametric t-SNE has the ability to preserve the local structure fairly well in the projection space (Van Der Maaten, 2009).

Currently, there does not exist a way to train one network containing both of these abilities. For this reason, we extend parametric t-SNE by introducing Regularized Parametric t-SNE (RP t-SNE), a technique which retains both abilities of autoencoders and parametric t-SNE by optimizing their cost functions into one autoencoder network. Consequently, to investigate whether RP t-SNE is able to retain or even improve these abilities, we formulate the following research question: *"To what extent are the abilities of autoencoders and parametric t-SNE retained or improved in an autoencoder network that optimizes both the reconstruction loss and t-SNE cost?"*. The research question is answered by comparing RP t-SNE against autoencoders in terms of its reconstruction performance, and against parametric t-SNE in terms of its generalization performance on the MNIST and COIL-20 image datasets. Additionally, the generalization performance of RP t-SNE is compared against two simple benchmark: PCA and kernel t-SNE (Gisbrecht et al., 2012). We find that RP t-SNE is able to retain the abilities of autoencoders, and parametric t-SNE. Moreover, for smaller sample sizes RP t-SNE seems to improve autoencoders due to less susceptibility to overfit on the reconstruction loss.

The remainder of this paper is structured as follows. In Section 2 we discuss relevant work found in the literature. Section 3 describes RP t-SNE, the employed benchmarks, and several generalization performance measures. Afterwards, the experimental setup and results of our experiments are given in Sections 4 and 5 respectively. Lastly, in Section 6 we provide the concluding remarks, limitations, and possible future research.

2 Related Work

In this section we discuss relevant literature to our research. First, in Section 2.1 we describe the t-SNE algorithm. Next, Section 2.2 provides an overview of the advantages of t-SNE over several existing dimensionality reduction techniques, and lastly Section 2.3 elaborates on autoencoder networks.

2.1 t-Distributed Stochastic Neighbor Embedding

The main approach of t-SNE is to construct an affinity probability distribution P for the high-dimensional input data, and then use gradient descent to optimize the low-dimensional projections with respect to the Kullback-Leibler (KL) divergence between P and the projection distribution Q. The KL divergence is a measure for the mismatch between two probability distribution such that gradient descent results in projections exhibiting affinities where Q is similar to P. Moreover, t-SNE preserves the local neighborhoods of high-dimensional data, because the affinities have more weight on datapoints that are nearby each other.

In particular, t-SNE constructs the joint probability distribution P by first converting the pairwise distances between high-dimensional datapoints into conditional probabilities $p_{j|i}$ representing similarities using a Gaussian kernel, that is, given a high-dimensional datapoint $\mathbf{x}_i$, the conditional probability $p_{j|i}$ that $\mathbf{x}_i$ would pick $\mathbf{x}_j$ as its neighbor is computed as

$$p_{j|i} = \frac{\exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||\mathbf{x}_i - \mathbf{x}_k||^2/2\sigma_i^2)}, \tag{1}$$

where $||\cdot||$ denotes the Euclidean distance and σ_i is the kernel width which is chosen with binary search such that every datapoint $\mathbf{x}_i$ has a desired user-specified perplexity. The perplexity is an input parameter roughly corresponding to the number of neighbors to preserve (Maaten and Hinton, 2008). A detailed procedure to find the kernel width is given in Appendix A.1. Furthermore, $p_{i|i}$ is set to zero, because we are only interested in pairwise similarities. The joint probabilities p_{ij} from P are then obtained as the average of the conditional probabilities $p_{ij} = \frac{p_{i|j}+p_{j|i}}{2n}$, which has the property that $\sum_j p_{ij} > \frac{1}{2n}$ for all datapoints $\mathbf{x}_i$.

Next, t-SNE constructs the joint projection distribution Q by using a Student-t distribution. The joint probabilities q_{ij} are thus given by

$$q_{ij} = \frac{(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2/\gamma)^{-\frac{\gamma+1}{2}}}{\sum_{k \neq l}(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2/\gamma)^{-\frac{\gamma+1}{2}}}, \tag{2}$$

where γ denotes the degrees of freedom that is set to one for 2-dimensional projections, and the denominator normalizes q_{ij} such that $\sum_{i \neq j} q_{ij} = 1$. A nice property of these probability distributions is that they are symmetric, that is, $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ for $\forall i, j$. Hence, we only have to minimize a single KL divergence between P and Q to find the optimal low-dimensional projection from the high-dimensional input data. The t-SNE cost function $C_{\text{t-SNE}}$ to minimize is then given as

$$C_{t-SNE} = KL(P||Q) = \sum_i^n \sum_j^n p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{3}$$

where every datapoint $\mathbf{x}_i$ makes a significant contribution to this cost function due to the property that $\sum_j p_{ij} > \frac{1}{2n}$. The vector gradient $\frac{\partial C_{\text{t-SNE}}}{\partial \mathbf{y}_i}$ is then used to find the corresponding optimal projections $\mathbf{y}_i$ with gradient descent. Specifically, this vector gradient is derived by Maaten and Hinton (2008) as

$$\frac{\partial C_{\text{t-SNE}}}{\partial \mathbf{y}_i} = \frac{2\gamma + 2}{\gamma} \sum_j^n (p_{ij} - q_{ij})(\mathbf{y}_i - \mathbf{y}_j)(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2/\gamma)^{-\frac{\gamma+1}{2}}. \tag{4}$$

The resulting projection $\mathbf{y}_i$ after gradient descent produces visualizations (i.e. projections of two or three dimensions) often better than other dimensionality reduction techniques, since t-SNE obtains nonlinear projections that preserve local neighborhoods. Moreover, it solves a big part of the "crowding problem" (Maaten and Hinton, 2008), which is the problem that the projection space has not enough room to accommodate all neighbors of a datapoint $\mathbf{x}_i$ from the high-dimensional input space. Though, the crowding problem becomes less severe for high projection dimensionalities, since the extent to which this problem occurs depends on the ratio between the intrinsic and projection dimensionality (Maaten and Hinton, 2008).

2.2 Comparative overview of t-SNE

First, we compare t-SNE against Stochastic Neighborhood Embedding (SNE) (Hinton and Roweis, 2003), the method which Maaten and Hinton (2008) have extended and improved. In particular, t-SNE improves SNE by two differences: (1) t-SNE uses a symmetrized version of the SNE cost function, and (2) t-SNE uses a Student-t distribution rather than a Gaussian to compute the similarity between two points in the projection space. The combination of these two differences results in a cost function that is easier to optimize. Moreover, t-SNE resolves a big part of the crowding problem which SNE has, since the heavier tails of the Student-t distribution ensures that moderately dissimilar high-dimensional datapoints are faithfully projected by a much larger distance in the projection space.

Next, t-SNE is compared against two linear techniques: PCA and classical scaling (Gower, 1966). These techniques are closely related to each other since they both construct a linear mapping to the projection space that describes most of the variance from the input data. As a result they preserve large dissimilarities between different objects, whereby the problem arises that the low-dimensional projection of dissimilar points are far apart. In contrast, t-SNE focuses on placing similar objects close to each other, that is, it preserves local neighborhoods rather than distances or density. Furthermore, for high-dimensional data that lies on or near a low-dimensional nonlinear manifold it is not possible to keep the low-dimensional projections of similar objects close together with a linear mapping, which t-SNE is able to do (Maaten and Hinton, 2008).

Other techniques such as Isomap (Tenenbaum et al., 2000) and Locally Linear Embedding (LLE) (Roweis and Saul, 2000) are nonlinear techniques, which first construct a neighborhood graph representation of the high-dimensional datapoints. Isomap tries to preserve the geodesic distances by finding the shortest paths between two datapoints in the graph (Tenenbaum et al., 2000), opposed to LLE, which tries to preserve more of the local properties of the data by writing the datapoints as a linear combination of their nearest neighbors (Roweis and Saul, 2000). However, these two techniques perform poorly on data consisting of two or more widely separated submanifolds, since such data gives rise to a disconnected neighborhood graph (Maaten and Hinton, 2008). This problem is not encountered by t-SNE, because it constructs an affinity probability distribution for the high-dimensional datapoints, instead of a neighborhood graph.

Lastly, we consider Sammon mapping (Sammon, 1969) a nonlinear technique related to multidimensional scaling (Groenen and van de Velden, 2005). As mentioned before, the problem of classical scaling is that it does not preserve local neighborhoods, since it mainly focuses on preserving large dissimilarities. Sammon mapping tries to solve this problem by adding weights to the cost function of classical scaling as proposed by Sammon (1969). The weakness of Sammon mapping is that the importance of preserving small pairwise distances is highly dependent on the small differences between these distances (Maaten and Hinton, 2008), that is, a small error in the model of two high-dimensional datapoints that are extremely close together could result in a large contribution to its cost function. The advantage of t-SNE over Sammon mapping is that the employed Gaussian kernel in high-dimensional space causes the modeling of separation between pairs of close datapoints, relative to the standard deviation of the Gaussian, to be almost independent of the magnitude of those separations (Maaten and Hinton, 2008), therefore these kinds of errors do not have such a large impact on its cost function.

2.3 Autoencoder networks

The increase in popularity of neural networks is due to their high flexibility, and capability to learn arbitrarily complex nonlinear functions using nonlinear activation functions. In particular, autoencoder networks, also referred to as autoassociative neural networks (Kramer, 1991), are popular networks for unsupervised dimensionality reduction. They are often symmetric with an odd number of hidden layers, and they consist of an encoder and decoder network as illustrated in Figure 1. The bottleneck layer of an autoencoder contains the projection $\mathbf{y}_i$ and the reconstruction layer contains the reconstructed input $\hat{\mathbf{x}}_i$. Autoencoders, thus learn the mapping from the high-dimensional input data to the low-dimensional projections, but also the mapping to reconstruct these projections back to the input data. However, autoencoders learn these mappings by minimizing the reconstruction loss, causing them to primarily focus on maximizing the variance of this data in the projection space (Maaten and Hinton, 2008). As a result, autoencoders are often not successful in preserving the local structure of neighborhoods when mapping high-dimensional datapoints to a projection space. In contrast, Van Der Maaten (2009) introduces parametric t-SNE, a technique that trains an encoder network with the t-SNE cost as objective. It is shown by Van Der Maaten (2009) that the mapping function obtained from this network is able to preserve the local structure of neigborhoods fairly well compared to techniques as PCA or autoencoders.

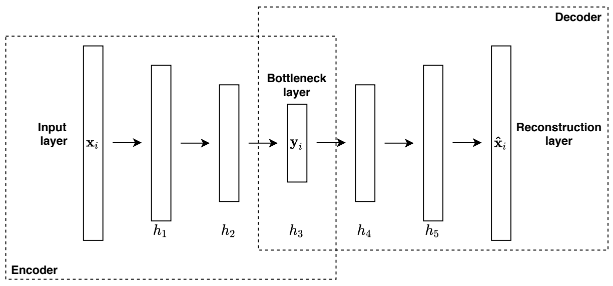


Figure 1: Schematic structure of an autoencoder with five hidden layers.

Overall, neural networks have shown to be reliable in many machine learning applications. However, they also have some drawbacks such as the lack of interpretability, long training times and tendency to get stuck in poor local minima. Though, the tendency to get stuck in poor local minima can be partially solved by following a three-stage training procedure using RBMs (Hinton and Salakhutdinov, 2006). The main goal of the RBMs is to pretrain the weights of the neural network such that it is closer to a good optimal solution before finetuning the weights via backpropagation.

3 Methodology

This section elaborates on the methods used in this paper. First, in Section 3.1 we discuss RP t-SNE. Then, in Sections 3.2 and 3.3 we describe the two benchmarks: PCA and kernel t-SNE respectively. Lastly, Section 3.4 describes several generalization performance measures.

3.1 Regularized Parametric t-SNE

The main objective of dimensionality reduction is to preserve the available information of the high-dimensional input data as much as possible in the projection space (Lee and Verleysen, 2009). Consequently, the ability to reconstruct the datapoint $\mathbf{x}_i$ from its projection $\mathbf{y}_i$, measured by the reconstruction loss E, could act as a valid measure. Ideally, this would mean that the mapping function f has an inverse mapping f^{-1}, but in general the exact inverse does not exist. So instead, we consider an approximate inverse mapping function $g : \mathbf{Y} \mapsto \hat{\mathbf{X}}$, where $\hat{\mathbf{X}}$ consists of the reconstructed high-dimensional datapoints $\hat{\mathbf{x}}$. Then, the reconstruction loss can be defined as the mean squared error between $\mathbf{x}_i$ and $\hat{\mathbf{x}}_i$:

$$E = \frac{1}{n} \sum_{i}^{n} ||\mathbf{x}_i - \hat{\mathbf{x}}_i||^2. \tag{5}$$

In particular, the decoder part of autoencoders has the ability to provide an approximate inverse mapping g that optimizes this reconstruction loss, unlike parametric t-SNE which only consists of an encoder network, and is thus not able to provide an explicit inverse mapping g. However, the advantage of parametric t-SNE is the ability to preserve the local structure by optimizing the t-SNE cost function (Van Der Maaten, 2009). Thus, to combine the abilities of both autoencoders and parametric t-SNE, we propose RP t-SNE, a technique that optimizes a weighted loss function between the reconstruction loss and the t-SNE cost in an autoencoder network. The weighted loss function is obtained by using the decoder part of an autoencoder as a regularizer on the encoder network from parametric t-SNE, resulting in the regularized t-SNE cost function C_{reg} with $\theta \in [0, 1]$ as a trade-off parameter between $C_{\text{t-SNE}}$ and E:

$$C_{\text{reg}} = \theta C_{\text{t-SNE}} + (1 - \theta)E. \tag{6}$$

The autoencoder network of RP t-SNE is trained with a three-stage training procedure using RBMs as described by Hinton and Salakhutdinov (2006). A short summary of RBMs is given in Section 3.1.1, where more details RBMs are given in Appendix A.2. Afterwards, in Section 3.1.2 we provide the three-stage training procedure.

3.1.1 Restricted Boltzmann Machine

RBMs are Markov Random Fields (Kindermann, 1980) with a bipartite graph structure consisting of visible nodes $\mathbf{v}$ modelling the input data, and hidden nodes $\mathbf{h}$ modelling the features of this data as illustrated in the left panel of Figure 2. The objective of RBMs is to learn the probability distribution over a set of input data by minimizing the contrastive divergence (Hinton, 2002). By doing this, the features of the hidden layer learn to represent the higher-order correlations of the input data (Hinton, 2009). Thus, when a RBM is given some input data it has to learn to generate this data with high probability. To accomplish this the RBM must find weights on its connections such that, relative to other possible data, the input data has low values for the contrastive divergence (Hinton, 2009). Intuitively, this means that the structure of the input distribution of a layer is reflected in the initial weights of the network. Therefore, less information should be lost by the changes made during the finetuning stage of the three-stage training procedure.

The visual and hidden nodes are typically Bernoulli distributed, meaning that these nodes only model binary data. However, if the mean field approximation is used, then the nodes may follow any exponential family distribution (Welling et al., 2005), allowing RBMs to model real-valued data with Gaussian distributed units as described by Salakhutdinov (2015). In particular, the RBM corresponding to the top layer uses Gaussian distributed hidden units to give rise to a linear activation function, since this makes the output of the neural network more stable (Van Der Maaten, 2009).

3.1.2 Three-stage training procedure

The training of our autoencoder networks consists of three stages as illustrated in the right panel of Figure 2. The first stage starts at the bottom layer where the first RBM is trained on the input data $\mathbf{x}_i$. Afterwards, this trained RBM infers the most likely values for the hidden nodes from each datapoint. These values are then used as input data to train the next RBM. This iterative procedure results in a stack of RBMs. In the second stage the weights of the trained RBMs are used to construct a pretrained autoencoder network by "unrolling" the stack of RBMs, that is, the inverse of the pretrained weights between the input layer and the the bottleneck layer is used as the weights between the bottleneck layer and the reconstruction layer. Finally, in the third stage the pretrained autoencoder network is finetuned by backpropagation.

Notice that the pretrained autoencoder network provides us with two mapping functions $f_W : \mathbf{X} \mapsto \mathbf{Y}$ and $h_W : \mathbf{X} \mapsto \hat{\mathbf{X}}$, where W are the weights of the network parameterizing the mappings. The gradient of C_{reg} with respect to W can be written as

$$\frac{\partial C_{\text{reg}}}{\partial W} = \theta \frac{\partial C_{\text{t-SNE}}}{\partial W} + (1-\theta)\frac{\partial E}{\partial W}, \tag{7}$$

using $\mathbf{y}_i = f_W(\mathbf{x}_i)$, $\hat{\mathbf{x}}_i = h_W(\mathbf{x}_i)$, and the chain rule the gradient can be further written down as

$$\theta \frac{\partial C_{\text{t-SNE}}}{\partial f_W(\mathbf{x}_i)} \frac{\partial f_W(\mathbf{x}_i)}{\partial W} + (1-\theta)\frac{\partial E}{\partial h_W(\mathbf{x_i})}\frac{\partial h_W(\mathbf{x_i})}{\partial W}. \tag{8}$$

This gradient can be computed using standard backpropagation for the terms $\frac{\partial f_W(\mathbf{x}_i)}{\partial W}$ and $\frac{\partial h_W(\mathbf{x_i})}{\partial W}$. Moreover, $\frac{\partial C_{\text{t-SNE}}}{\partial f_W(\mathbf{x}_i)}$ can be simply obtained by plugging $\mathbf{y}_i = f_W(\mathbf{x}_i)$ into Equation (4), and $\frac{\partial E}{\partial h_W(\mathbf{x_i})}$ can be derived as

$$\frac{\partial E}{\partial h_W(\mathbf{x_i})} = -\frac{2}{n}(\mathbf{x}_i - h_W(\mathbf{x}_i)). \tag{9}$$

The backpropagation starts at the reconstruction layer with the gradient of E, and it is combined with the gradient of $C_{\text{t-SNE}}$ when it reaches the bottleneck layer containing the projections $\mathbf{y}_i$.

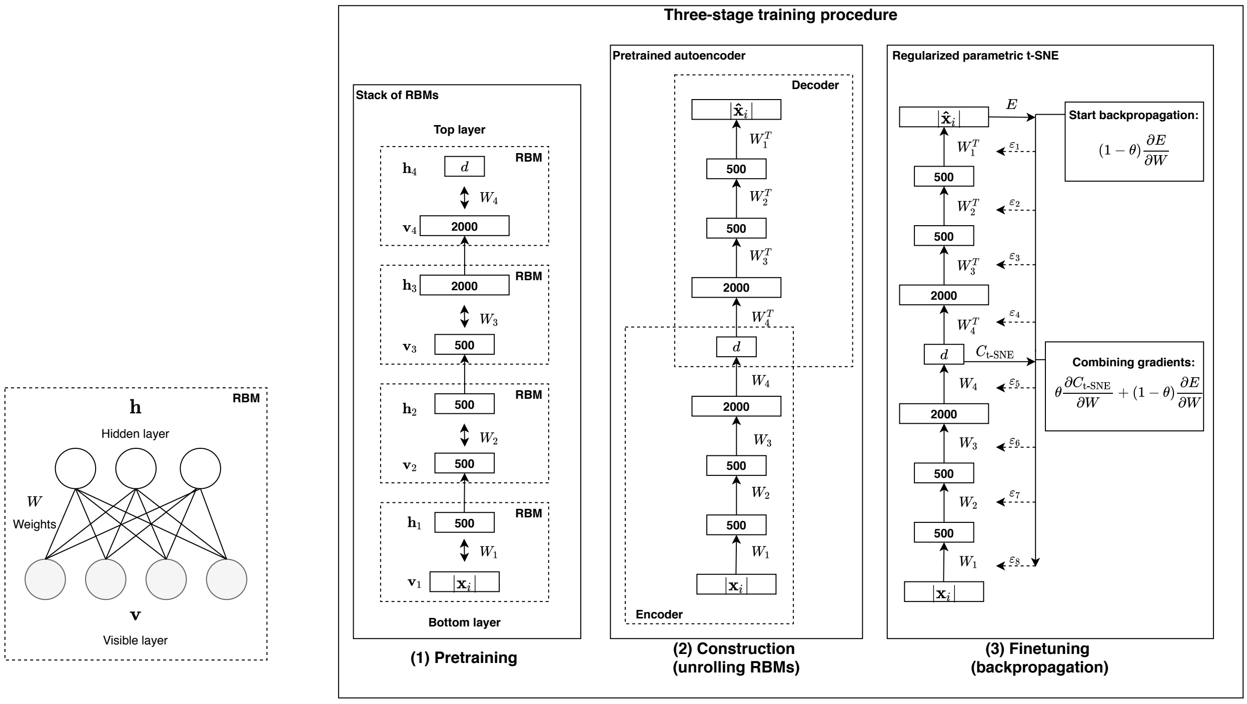

RBM
h
Hidden layer
W
Weights
v
Visible layer
Three-stage training procedure
Stack of RBMs
Top layer
Bottom layer
Pretrained autoencoder
Decoder
Encoder
Regularized parametric t-SNE
Start backpropagation:
$(1-\theta)\frac{\partial E}{\partial W}$
Combining gradients:
$\theta\frac{\partial C_{\text{t-SNE}}}{\partial W} + (1-\theta)\frac{\partial E}{\partial W}$
(1) Pretraining
(2) Construction (unrolling RBMs)
(3) Finetuning (backpropagation)

Figure 2: Left panel: schematic structure of a Restricted Boltzmann Machine. Right panel: schematic structure of the three-stage training procedure inspired by Hinton and Salakhutdinov (2006) for a $|\mathbf{x}_i| - 500 - 500 - 2000 - d$ autoencoder network, where $|\mathbf{x}_i|$ denotes the dimensionality of $\mathbf{x}_i$, and the ε's denote the changes in the weights during backpropagation. Notice that setting $\theta = 1$ results in parametric t-SNE, setting $\theta = 0$ results in an autoencoder, and setting $\theta \in (0, 1)$ results in RP t-SNE.

3.2 Principal Component Analysis

As described by Hotelling (1933) PCA is an unsupervised technique that constructs low-dimensional linear projections of high-dimensional data by preserving as much of the variance in the data as possible. PCA finds the directions of these projections with a linear basis $\underset{D\times d}{\mathbf{M}} = [\mathbf{m}_1, ..., \mathbf{m}_d]$, where $\underset{D\times 1}{\mathbf{m}_i}$ is the "principal component" that seeks to maximize the variance of the linear combination from the columns of $\underset{n\times D}{\mathbf{X}}$. If we denote $\text{cov}(\mathbf{X})$ as the sample covariance matrix of $\mathbf{X}$, which is equal to $\frac{1}{n-1}\mathbf{X}^\intercal\mathbf{X}$, then the variance of the linear combination $\mathbf{X}\mathbf{m}_i$ is given by

$$\underset{1\times 1}{\text{var}(\mathbf{X}\mathbf{m}_i)} = \underset{1\times D}{\mathbf{m}_i^\intercal}\,\underset{D\times D}{\text{cov}(\mathbf{X})}\,\underset{D\times 1}{\mathbf{m}_i}\,. \tag{10}$$

However, without a constraint on $\mathbf{m}_i$, $\text{var}(\mathbf{X}\mathbf{m}_i)$ could be maximized by picking $\mathbf{m}_i$ arbitrarily large. Thus, a normalization constraint is used such that the $\mathbf{m}_i$'s have a unit norm, that is,

$||\mathbf{m}_i||^2 = 1$. To enforce this constraint when maximizing $\mathrm{var}(\mathbf{X}\mathbf{m}_i)$, the method of Lagrange multipliers (Bertsekas, 2014) is employed by maximizing the following Lagrangian function:

$$\mathcal{L}(\mathbf{m}_i, \lambda_i) = \mathrm{var}(\mathbf{X}\mathbf{m}_i) - \lambda_i(\mathbf{m}_i^\intercal \mathbf{m}_i - 1), \tag{11}$$

where λ_i is the Lagrange multiplier. Furthermore, it can be shown that $\mathbf{X}\mathbf{m}_i$ is able to maximize its variance subject to uncorrelatedness with previous linear combinations (Hotelling, 1933), if also the constraint of orthogonality between the principal components $\mathbf{m}_i$ is added to $\mathcal{L}(\mathbf{m}_i, \lambda_i)$, that is, $\mathbf{m}_i^\intercal \mathbf{m}_j = 0$ if $i \neq j$. In particular, differentiating the Lagrangian function with respect to $\mathbf{m}_i$ for all $\mathbf{m}_i$'s subject to orthonormality and equating to the null vector results in the following eigenproblem:

$$\underset{D\times D}{\mathrm{cov}(\mathbf{X})}\ \underset{D\times d}{\mathbf{M}} = \underset{D\times d}{\mathbf{M}}\ \underset{d\times d}{\mathrm{diag}(\lambda_1, ..., \lambda_d)}, \tag{12}$$

where the principal components $\mathbf{m}_i$'s form an orthonormal set of vectors and $\mathrm{diag}(\lambda_1, ..., \lambda_d)$ denotes the diagonal matrix with the λ_i's as the diagonal elements.

PCA solves the eigenproblem given in equation (12) for the d largest principal eigenvalues. The linear basis $\mathbf{M}$ is then constructed with the principal components $\mathbf{m}_i$'s corresponding to the d largest eigenvalues λ_i in decreasing order. The out-of-sample projections $\underset{p\times d}{\mathbf{Y}_{\text{test}}}$ can thus be computed by first employing PCA on the training set $\underset{m\times D}{\mathbf{X}_{\text{train}}}$ to obtain the linear basis $\underset{D\times d}{\mathbf{M}_{\text{train}}}$, where p and m denote the number of datapoints in the test and training set respectively. Then, $\mathbf{Y}_{\text{test}}$ is obtained by mapping the test set $\underset{p\times D}{\mathbf{X}_{\text{test}}}$ onto $\mathbf{M}_{\text{train}}$ as

$$\underset{p\times d}{\mathbf{Y}_{\text{test}}} = \underset{p\times D}{\mathbf{X}_{\text{test}}}\underset{D\times d}{\mathbf{M}_{\text{train}}}. \tag{13}$$

3.3 Kernel t-SNE

Kernel functions allow us to operate in high-dimensional space without computing the datapoints in that space, since instead we compute the inner products between the high-dimensional datapoints. This is also referred to as the "kernel trick" (Hofmann et al., 2008) and it allows us to give a linear model the ability to construct nonlinear mappings. Specifically, the mapping of kernel t-SNE is obtained by a linear combination of normalized Gaussian kernels as

$$f_\alpha : \mathbf{x}_i \mapsto \mathbf{y}_i = \sum_j^m \frac{\kappa(\mathbf{x}_i, \mathbf{x}_j)}{\sum_l^m \kappa(\mathbf{x}_i, \mathbf{x}_l)} \boldsymbol{\alpha}_j, \tag{14}$$

where j runs over a training set $\mathbf{X}_{\text{train}} = \{\mathbf{x}_1, ..., \mathbf{x}_m\}$ sampled from the input data $\mathbf{X}$ such that $\mathbf{x}_j$ corresponds to a fixed sample of datapoints, $\boldsymbol{\alpha}_j \in \mathbb{R}^{d\times 1}$ the parameters to be optimized corresponding to the projection $\mathbf{y}_j$, and κ the Gaussian kernel defined as

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma_j^2), \tag{15}$$

where the kernel width σ_j is chosen as the distance of $\mathbf{x}_j$ to its closest neighbor scaled with a small factor $c = 0.05$ as done by Gisbrecht et al. (2012).

The mapping f_α is obtained with a two-step approach, where the first step is to retrieve a pair of train samples $\mathbf{X}_{\text{train}}$ and projections $\mathbf{Y}_{\text{train}}$ by employing the t-SNE algorithm on $\mathbf{X}_{\text{train}}$ as described in Section 2.1. Then, the second step is to optimize $\boldsymbol{\alpha}_j$ by minimizing the least squares error between $\mathbf{y}_j$ and $f_\alpha(\mathbf{x}_j)$:

$$\sum_j^m ||\mathbf{y}_j - f_\alpha(\mathbf{x}_j)||^2. \tag{16}$$

Let $\underset{m\times d}{\mathbf{A}}$ be the matrix containing the vectors $\boldsymbol{\alpha}_j^\intercal$ in its rows and $\underset{n\times n}{\mathbf{K}}$ the Gram matrix with as entries the normalized inproduct between the high-dimensional datapoints in $\mathbf{X}$:

$$[\mathbf{K}]_{i,j} = \frac{\kappa(\mathbf{x}_i, \mathbf{x}_j)}{\sum_l^m \kappa(\mathbf{x}_i, \mathbf{x}_l)}. \tag{17}$$

If we define $\underset{m\times m}{\mathbf{K}_{\text{train}}}$ to be the Gram matrix with as entries the inproduct between the datapoints in $\mathbf{X}_{\text{train}}$, then the rows $\boldsymbol{\alpha}_j^\intercal$ of $\mathbf{A}$ can be retrieved as

$$\underset{m\times d}{\mathbf{A}} = \underset{m\times m}{\mathbf{K}^+_{\text{train}}}\underset{m\times d}{\mathbf{Y}_{\text{train}}}, \tag{18}$$

where $\mathbf{K}^+_{\text{train}}$ is the Moore-Penrose inverse (Penrose, 1956) of $\mathbf{K}_{\text{train}}$ providing the solution to the least squares problem. Consequently, if we define $\underset{p\times m}{\mathbf{K}_{\text{test}}}$ to be the Gram matrix with as entries the inproduct between the datapoints in $\mathbf{X}_{\text{test}}$ and $\mathbf{X}_{\text{train}}$, then the out-of-sample projections $\mathbf{Y}_{\text{test}}$ are obtained as

$$\underset{p\times d}{\mathbf{Y}_{\text{test}}} = \underset{p\times m}{\mathbf{K}_{\text{test}}}\underset{m\times d}{\mathbf{A}}. \tag{19}$$

3.4 Generalization performance measures

When the dimensionality of high-dimensional data is reduced it is generally not possible to preserve all similarities in the data. Consequently, a criterion of a good dimensionality reduction technique is the ability to cluster the datapoints correctly which is similar to preserving the local neighborhood of a high-dimensional datapoint in the projection space. For this reason, we consider general measures that employ the k-Nearest Neighbor (k-NN) algorithm, since this algorithm is able to find the corresponding nearest neighbor clusters of datapoints. Using this algorithm we compute the trustworthiness $T(k) \in [0,1]$ and continuity $M(k) \in [0,1]$ quality measures as introduced by Kaski et al. (2003), and the *generalization error* $\in [0,1]$.

The trustworthiness and continuity are based on two kinds of errors caused by dimensionality reduction: (1) datapoints that are not neighbors in the input space can be mapped too close to each other in the projection space causing datapoints to be falsely identified as neighbors, and (2) datapoints that are originally close to each other in the input space can be mapped faraway in the projection space causing some neighbor relations to be absent in the projection space. These two errors negatively affect the trustworthiness and continuity respectively. The precise computations of these two measures are described in Appendix A.3. Note that these measures have an interpretation similar to precision and recall, that is, $T(k)$ measures to what extent the k nearest neighbors of $\mathbf{y}_i$ correspond to the k nearest neighbors of $\mathbf{x}_i$, and $M(k)$ measures to what extend the k nearest neighbors of $\mathbf{x}_i$ are retrieved in the k nearest neighbors of $\mathbf{y}_i$. Thus, contrary to the precision and recall, the trustworthiness and continuity do incorporate the similarity ranking between the k nearest neighbors, resulting in larger error for datapoints that come into a neighborhood from faraway than from closer by.

Furthermore, the *generalization error* can be computed for labeled data and it measures how accurately our mappings are able to place projections of unseen data in the correct cluster. This measure is computed by training an 1-NN classifier on the train projections $\mathbf{Y}_{\text{train}}$, and then measuring the error rate of this classifier on the out-of-sample projections $\mathbf{Y}_{\text{test}}$.

4 Experimental setup

The generalization performances of the parametric mappings from our techniques are evaluated in terms of visualizations and the generalization performance measures as previously described in Section 3.4. Furthermore, the reconstruction performances for the autoencoder and RP t-SNE are evaluated in terms of reconstruction losses and quality of the reconstructed images.

In Section 4.1 we specify the employed image datasets for our experiments. Afterwards, Section 4.2 discusses our optimization procedure of the t-SNE algorithm. Lastly, in Section 4.3 we provide the architecture and optimization procedure of the implemented autoencoder networks.

4.1 Datasets and preprocessing

To evaluate the generalization and reconstruction performance of RP t-SNE, we employ the MNIST[1] and COIL-20[2] grayscale image datasets. The MNIST dataset is employed, since it is a commonly used dataset in many machine learning applications such that this paper is easily comparable with other papers in the literature, while the COIL-20 dataset is employed due to its more complex images.

Specifically, the MNIST dataset contains 70,000 grayscale images of handwritten digits. The images have $28 \times 28 = 784$ pixels (i.e. 784 dimensions), and there are 10 classes corresponding to the integers ranging from 0 to 9. The COIL-20 dataset contains $20 \times 72 = 1440$ grayscale images of 20 different objects taken at 72 different angels, where each image has $32 \times 32 = 1024$ pixels (i.e. 1024 dimensions), and there are 20 classes corresponding to the different objects. Furthermore, the pixels of grayscale images correspond to single RGB values in the range of [0, 255]. Thus, to convert the data into a more manageable form for our techniques, we can normalize the datasets by dividing each pixel by 255 such that the values of these features are in the range of [0, 1].

Furthermore, we split the MNIST and COIL-20 datasets into a training and test dataset. The MNIST dataset has already been split into a training and test dataset consisting of 60,000 and 10,000 images respectively, but due to computational reasons we do not employ the complete datasets. Instead, we randomly sample 10,000 training and 5,000 test images from the original

[1]The MNIST dataset is publicly available at `http://yann.lecun.com/exdb/mnist/index.html`

[2]The COIL-20 dataset is publicly available at `https://cs.columbia.edu/CAVE/software/softlib/coil-20.php`

MNIST training and test datasets respectively. From the COIL-20 dataset we randomly sample 960 and 480 images for our training and test dataset respectively. However, due to the relatively small sample size we sample the images in a way such that each class in our training and test datasets contain an equal number of images. By balancing the classes we avoid the risk of our parametric mappings not learning certain classes due to the limited number of samples.

4.2 Optimization of t-SNE

To optimize the t-SNE algorithm we first discuss the setting of the degrees of freedom γ of the Student-t distribution that is used to model the probabilities in the projection space. The t-SNE cost function $C_{\text{t-SNE}}$ with a degree of freedom γ set to one performs well for 2-dimensional projections as shown by Maaten and Hinton (2008). However, the volume of the projection space grows exponentially with its dimensionality d due to the "curse of dimensionality". Implying that to appropriately model the local structure of input data for projections with a dimensionality higher than two, a degree of freedom higher than one is required. This is because higher degrees of freedom lead to more space availability in the projection space due to the lighter tails of the Student-t distribution as discussed by Van Der Maaten (2009). Specifically, the thickness of the tail of a Student-t distribution decreases exponentially with the degrees of freedom γ. It thus seems likely that γ is linearly dependent on d. Therefore, we set $\gamma = d - 1$ as proposed by Van Der Maaten (2009) such that γ is linearly dependent on d for which the 2-dimensional projections still uses a single degree of freedom.

In addition, to validate our implementation of t-SNE, we employ and compare three different gradient descent methods: regular Stochastic Gradient Descent (SGD) as a benchmark, adaptive SGD (a-SGD) with momentum as the implementation of Maaten and Hinton (2008), and the recently introduced Adaptive moment estimation (Adam) (Kingma and Ba, 2014) as our implementation. The update steps for these methods are given as

$$\text{SGD: } \mathbf{y}_i^{(t)} \leftarrow \mathbf{y}_i^{(t-1)} - \eta \Delta \mathbf{y}_i^{(t-1)}, \tag{20}$$

$$\text{a-SGD with momentum: } \mathbf{y}_i^{(t)} \leftarrow \mathbf{y}_i^{(t-1)} - \eta_{\text{a}}^{(t)} \Delta \mathbf{y}_i^{(t-1)} + \rho^{(t)}(\mathbf{y}_i^{(t-1)} - \mathbf{y}_i^{(t-2)}), \text{ and} \tag{21}$$

$$\text{Adam: } \mathbf{y}_i^{(t)} \leftarrow \mathbf{y}_i^{(t-1)} - \eta \frac{m^{(t)}/(1-\beta_1^t)}{\sqrt{v^{(t)}/(1-\beta_2^t)} + \epsilon}. \tag{22}$$

In these update steps, t is the current iteration, $\Delta \mathbf{y}_i$ is equal to the gradient $\frac{\partial C_{\text{t-SNE}}}{\partial \mathbf{y}_i}$, η is the learning rate, $\eta_{\text{a}}^{(t)}$ is the learning rate using the adaptive learning scheme described by Jacobs (1988), $\rho^{(t)}$ is a momentum term that is set to 0.5 for $t < 20$ and 0.8 for $t > 20$, ϵ is a small number set to 10^{-9} to avoid dividing by zero, and the hyperparameters β_1 and β_2 are the exponential decay rates controlling the biased first moment estimate $m^{(t)}$ and second moment estimate $v^{(t)}$ as described by Kingma and Ba (2014). The initial learning rates for both SGD and a-SGD with momentum are set to 100 as done by Van Der Maaten (2009). Furthermore, for Adam we find that a learning rate of 0.1 works well for the implementation of t-SNE combined with the hyperparameters β_1 and β_2 set to 0.85 and 0.90 respectively.

The remainder of the setup for t-SNE is kept similar as the approach by Maaten and Hinton (2008), that is, first we set the perplexity to 40 and reduce the dimensionality of the MNIST and COIL-20 datasets to 30 with PCA for noise suppression and computational reasons. Moreover, the number of gradient descent iterations is set to 1000, and the "early exaggeration" trick is applied, which is to multiply all the p_{ij}'s with 4 in the first 100 iterations of the gradient descent to help the clusters find a good global organization as discussed by Maaten and Hinton (2008).

4.3 Optimization of autoencoder networks

The same network architecture is used for the autoencoder, parametric t-SNE and RP t-SNE networks to make the comparison between these techniques fair, where it has to be noted that parametric t-SNE only consists of the encoder part of an autoencoder network. Specifically, for the MNIST dataset we use a $784 - 500 - 500 - 2000 - d$ autoencoder network motivated by the experimental results from Hinton and Salakhutdinov (2006). Consequently, we use a $1024 - 500 - 500 - 2000 - d$ autoencoder network for the COIL-20 dataset. All the layers of these networks use a logistic activation function except for the bottleneck layer which instead uses a linear activation function to make the projections more stable (Van Der Maaten, 2009).

These networks are trained using the three-stage training procedure as previously described in Section 3.1.2. The pretraining of the three networks are similar. Though, the finetuning stage differ for the three networks due to the different objective function. Specifically, parametric t-SNE minimizes $C_{\text{t-SNE}}$, the autoencoder minimizes E and RP t-SNE minimizes C_{reg} as given in equations (3), (5) and (6) respectively. The networks are finetuned for 50 epochs using Adam with β_1, β_2 and η set to 0.90, 0.99 and 0.1 respectively.

Furthermore, the optimal trade-off parameters θ^* for RP t-SNE are found with 3-fold cross-validation from the range of values $\theta \in \{0.01, 0.10, 0.30, 0.50, 0.70, 0.90, 0.99\}$. We consider this range, since θ is a continuous trade-off parameter between zero and one, and thus it is not possible to consider all possibilities. Therefore, we instead choose the range such that it sufficiently covers the values between zero and one by starting with a low value 0.1 and incrementing with small steps of 0.2. In addition, we also include two extreme cases 0.01 and 0.99 to account for the possibilities when θ gives very high weights to $C_{\text{t-SNE}}$ or E. Next, we require a criterion to pick the optimal trade-off parameters θ^*. A logical choice for the criterion would be C_{reg}, since this is the objective of RP t-SNE. However, the magnitudes of the terms $C_{\text{t-SNE}}$ and E might substantially differ depending on the input data. This means that C_{reg} is not a fair criterion, since θ would have a bias by giving the highest weight to the term with the lowest magnitude as this achieves the lowest value of C_{reg}. Therefore, we instead use the generalization error as the criterion, since it is not dependent on the values of $C_{\text{t-SNE}}$ and E, and thus it does not has the bias that C_{reg} has.

5 Results

In this section we provide the results of our techniques. The techniques are implemented using Python 3.7, where our autoencoder networks employ the Keras library (Chollet et al., 2015) with TensorFlow (Abadi et al., 2016) as backend. The code can be found at: `https://github.com/StefanLam99/OOS__tSNE`, where a brief description of the Python files is given in Appendix C.

In Section 5.1 we compare and validate our implementation of the t-SNE algorithm with the implementation of Maaten and Hinton (2008). Following, Section 5.2 provides the optimal trade-off parameters for RP t-SNE. Lastly, in Sections 5.3 and 5.4 we compare the generalization and reconstruction performances of our techniques.

5.1 Implementation of t-SNE

We validate our implementation of t-SNE by comparing the convergence rates of the t-SNE cost and visualizations by the three implemented gradient descent methods as shown in Figures 3 and 4, where similar convergence rates are retrieved for the 10- and 20-dimensional projections as shown in Appendix B.1. The visualizations of the implementations are compared with each other, because t-SNE mainly focuses on preserving the local neighborhoods, implying that our implementation should provide similar clusters in the visualizations as the original implementation of Maaten and Hinton (2008). Furthermore, the convergence rates are able to show us the differences in update steps between the implementations, and whether it converges to similar t-SNE costs. It has to be noted however that similar t-SNE costs are not able to guarantee similar projections, but it does demonstrate similar performances from the projections of the implementations.

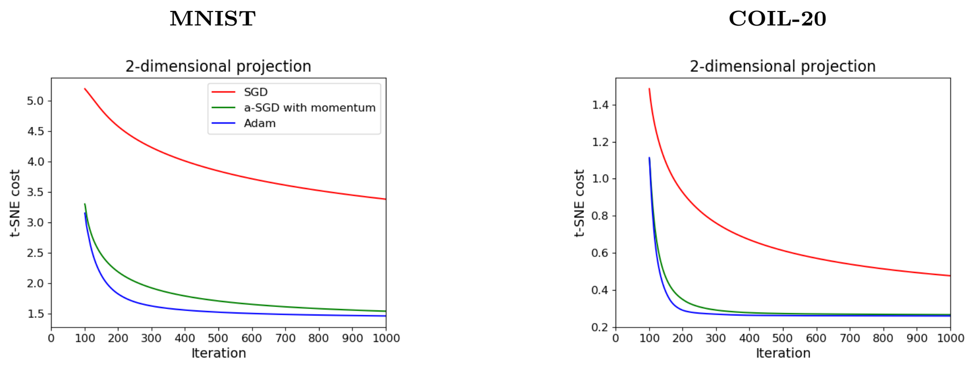
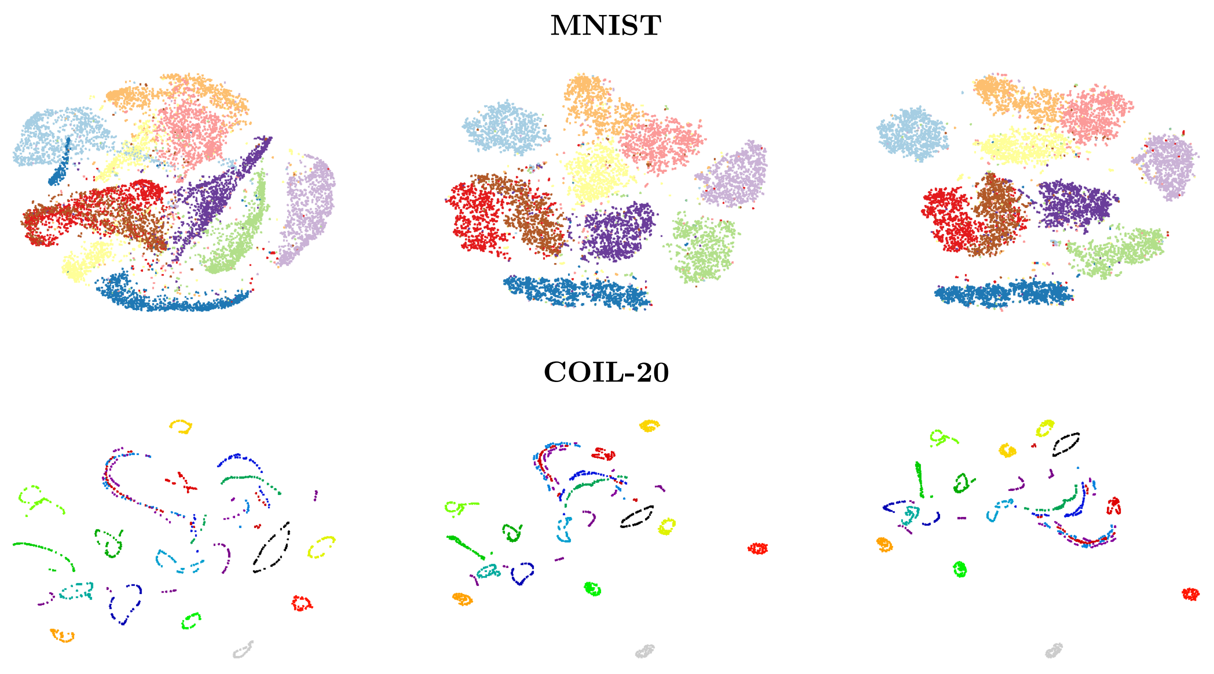


Figure 3: Plots of the t-SNE cost against the number of iterations for the three gradient descent methods when applying the t-SNE algorithm on the MNIST and COIL-20 training datasets for 2-dimensional projections. The plots start after the early exaggeration, which is after the 100th iteration.



SGD (benchmark implementation)

a-SGD with momentum (Implementation of Maaten and Hinton (2008))

Adam (our implementation)

Figure 4: Visualizations of the MNIST and COIL-20 training datasets by the t-SNE algorithm using (from left to right) SGD, a-SGD with momentum, and Adam as the gradient descent methods, where each color corresponds to a different class.

In particular, Figures 3 and 4 show two important observations. First, the benchmark implementation using SGD achieves a relatively high t-SNE cost due to the slow convergence rate which results in inferior visualization compared to the other methods. Second, our implementation with Adam converges to a similar t-SNE cost as a-SGD with momentum, the implementation of Maaten and Hinton (2008). This can also be observed in the visualizations, since similar clusters are formed for the two datasets. However, we do observe that the clusters of these visualizations might differ slightly in shape or even relative placements in the projection space. The latter can be explained, because t-SNE focuses on preserving clusters, rather than the distance between clusters. Implying that the similar t-SNE costs of the implementations are reflected in similar shaped clusters, and not the placements of those clusters. In general, the differences in shape and placements of the clusters can be explained by the non-convexity of the t-SNE cost function. The non-convexity implies that while our implementation converges to a similar t-SNE cost as that of Maaten and Hinton (2008), the similar t-SNE costs might still correspond to local optima with different projections due to the different update steps between Adam and a-SGD with momentum.

Thus, we validate our implementation, since it provides similar results as that from Maaten and Hinton (2008), where small differences between the implementations are explained by the different update steps of the gradient descent methods. Moreover, Adam has a better convergence rate than a-SGD with momentum, since it converges slightly faster and it reaches a similar or lower t-SNE cost. For this reason, when implementing kernel t-SNE we opt for Adam as the gradient descent method to obtain the train projections.

5.2 Optimal trade-off parameters

To validate our choice of the generalization error as the criterion for 3-fold cross-validation, we illustrate the difference in magnitudes between the t-SNE cost and reconstruction loss by presenting in Figure 5 plots of the costs when finetuning the autoencoder and parametric t-SNE networks for 2-dimensional projections. From this it is clear that the magnitude of the t-SNE cost is higher than the reconstruction loss. This implies that the optimal trade-off parameter should give the highest weight to the reconstruction loss, since this choice would achieve the lowest regularized t-SNE cost. Thus, the regularized t-SNE cost is not a fair criterion as discussed in Section 4.3.

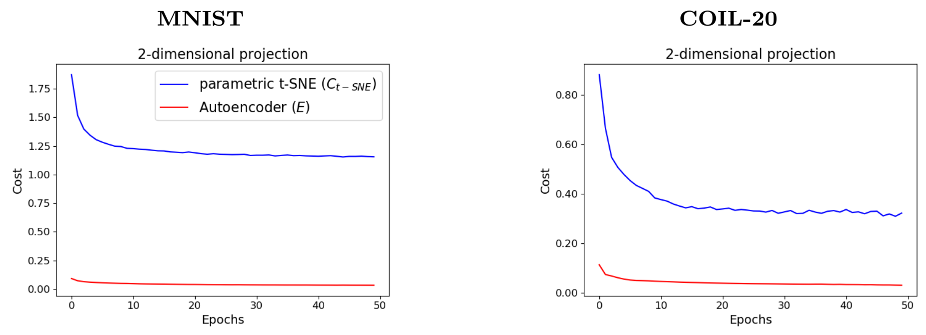


Figure 5: Plots of the t-SNE cost and reconstruction loss against the number of epochs, when finetuning the autoencoder and parametric t-SNE networks for 2-dimensional projections on the MNIST and COIL-20 training datasets.

Moreover, we provide plots of the generalization errors against θ in Figure 6, where the star marker indicates the lowest generalization errors. This figure shows that the generalization error does not has a bias for a specific trade-off parameter, since the generalization errors are fairly steady for different values of the trade-off parameter. Therefore, we validate the choice of the generalization error as the criterion, where Table 1 provides the optimal trade-off parameters.

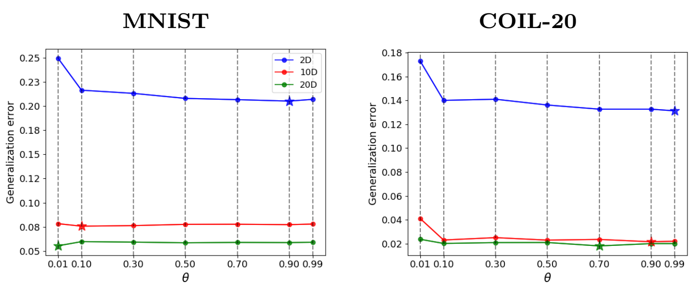


Figure 6: Plots of the generalization errors of RP t-SNE against $\theta \in \{0.01, 0.10, 0.30, 0.50, 0.70, 0.90, 0.99\}$ obtained by 3-fold cross-validation on the MNIST and COIL-20 training datasets, where the star marker indicates the lowest generalization error.

Table 1: Optimal trade-off parameters for RP t-SNE corresponding to the lowest generalization error when using 3-fold cross-validation on the MNIST and COIL-20 training datasets.

	MNIST			**COIL-20**		
	2D	**10D**	**20D**	**2D**	**10D**	**20D**
θ^*	0.90	0.10	0.01	0.99	0.90	0.70

5.3 Generalization performance

First, we compare the out-of-sample visualizations of the MNIST test dataset by our parametric mappings as shown in Figure 7, where the visualizations of the COIL-20 test dataset can be found in Appendix B.2. From the results, it is clear that the mappings of PCA and the autoencoder are not able to preserve the local structure well enough compared to the other mappings. PCA has an inferior performance due to its linear nature which is too restrictive to find good projections for nonlinear data. The autoencoder does demonstrate some cluster forming, but there is no clear separation between these clusters which is caused by the maximization of its variance in the projection space to achieve low reconstruction losses. In contrast, kernel, parametric and RP t-SNE do preserve the local structure fairly well, likely because these techniques are all focused on obtaining a parametric mapping that optimizes the t-SNE cost. It is notable that the visualizations of parametric and RP t-SNE are almost identical to each other.

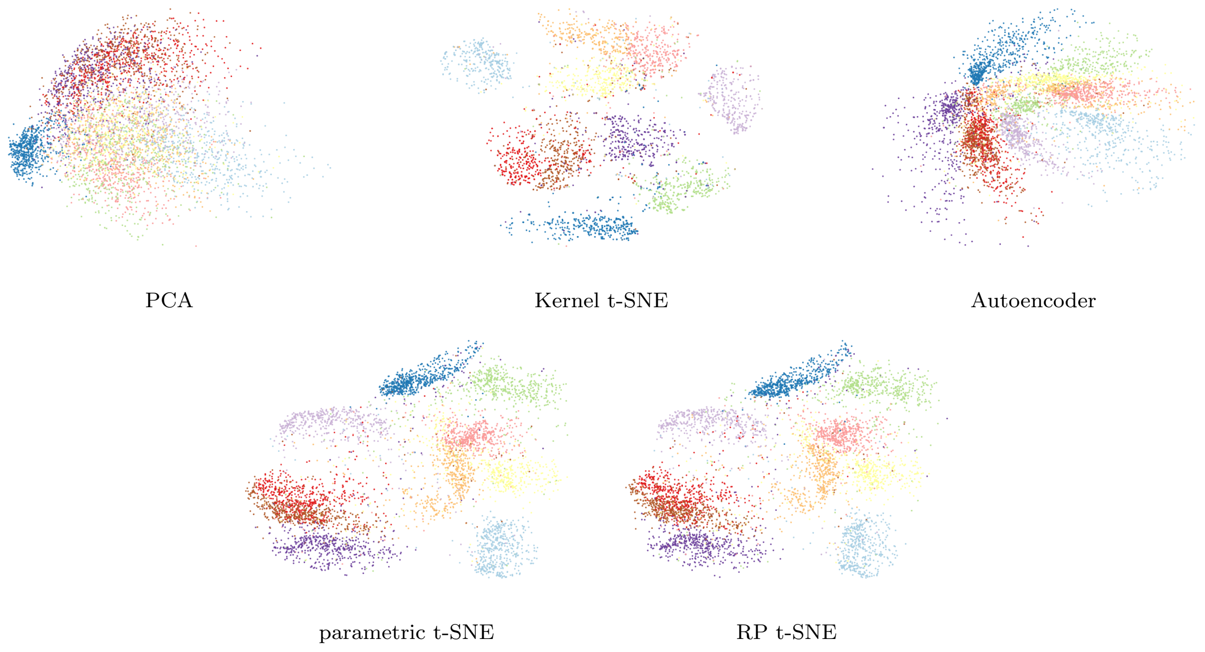


Figure 7: Out-of-sample visualizations of the MNIST test dataset by PCA, Kernel t-SNE, an autoencoder, parametric t-SNE and RP t-SNE.

Next, the generalization errors, trustworthiness and continuity of our mappings are compared by presenting the generalization errors with the lowest error in bold in Table 2 and the plots of the trustworthiness-continuity curves in Figure 8. In particular, the best performing trustworthiness-continuity curves are at the upper right corner of these plots, since this indicates both a high trustworthiness and continuity. Moreover, the curves demonstrate that generally the trustworthiness and continuity decreases as the neighborhood size increases, which is to be expected since it is harder to preserve a large neighborhood than a small one. We can thus interpret short curves as a measure on how robust our parametric mappings are for changes in neighborhood size, while long curves indicate that the mapping is not able to preserve large neighborhoods fairly well.

Table 2: Generalization errors of the projections from the trained models on the MNIST and COIL-20 test datasets.

	Generalization errors					
	MNIST			**COIL-20**		
	2D	**10D**	**20D**	**2D**	**10D**	**20D**
PCA	0.633	0.140	0.069	0.344	0.080	0.021
Autoencoder	0.319	0.096	0.065	0.331	0.194	0.104
Kernel t-SNE	**0.175**	0.185	0.184	**0.021**	**0.017**	0.017
Parametric t-SNE	0.201	0.087	0.073	0.158	**0.017**	**0.015**
RP t-SNE	0.200	**0.083**	**0.063**	0.163	0.019	0.017

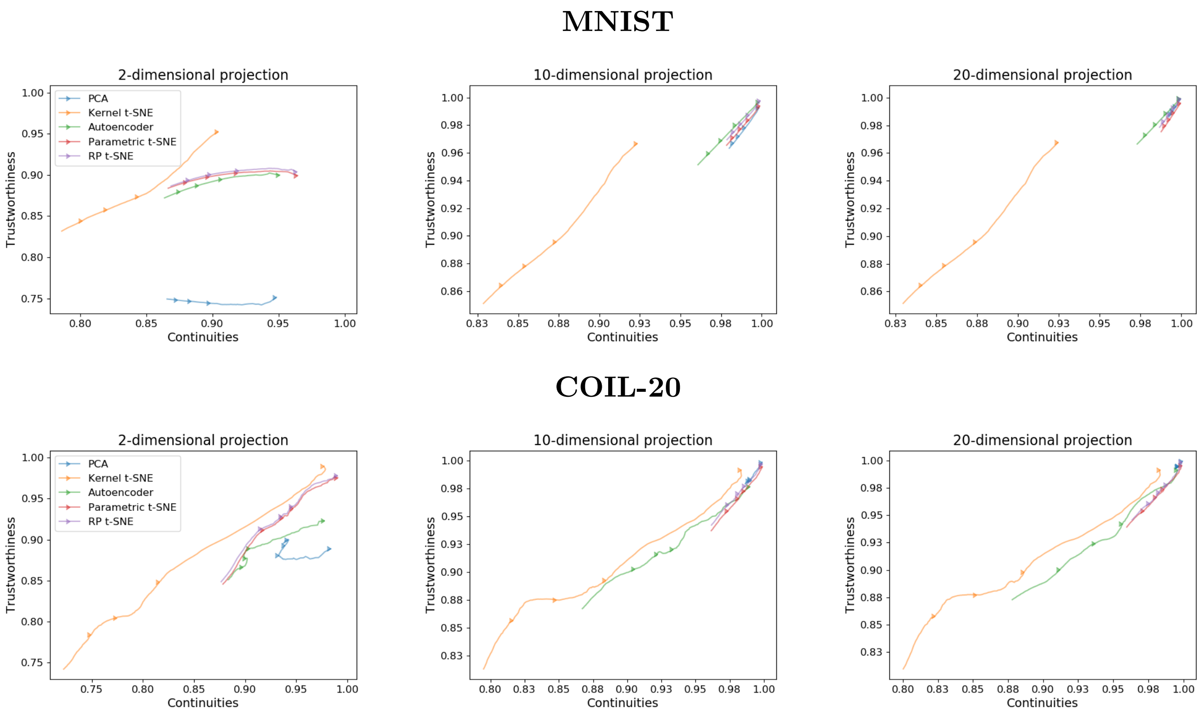


Figure 8: Trustworthiness $T(k)$ plotted against the continuity $M(k)$ of d-dimensional projections for the MNIST (upper row plots), and the COIL-20 (bottom row plots) test datasets. The number of neighbors k is varied from 1 to 200, and the markers on the curves are set to 50 point invervals.

In summary, the results reveal that parametric and RP t-SNE have similar visualizations, generalization errors and trustworthiness-continuity curves. It can thus be concluded that the local structure preservation ability of parametric t-SNE is retained in RP t-SNE. Moreover, the parametric mappings from these techniques provide overall the best generalization performances. Although, kernel t-SNE might obtain the lowest generalization errors for 2-dimensional projections, in terms of generalization performance we still give the edge to parametric and RP t-SNE due to the more robust trustworthiness-continuity curves. However, for higher dimensionalities (i.e. 10 and 20) of the projections space, we recognize that a linear benchmark such as PCA is able to give generalization errors and trustworthiness-continuity curves on par with that from parametric and RP t-SNE. This suggests that parametric and RP t-SNE only have a strong generalization performance for relatively low-dimensional projections. An explanation for this lies in a key characteristic of t-SNE, which is to solve the crowding problem. This problem is less severe for higher projection dimensionalities as mentioned before in Section 2.1, hence at some point the crowding problem becomes negligible such that t-SNE eventually stops improving the generalization performance, if the projection dimensionality keeps increasing.

5.4 Reconstruction performance

In Table 3 we present the reconstruction losses of the autoencoder and RP t-SNE networks with the lowest loss in bold. This table shows that RP t-SNE provides similar reconstruction losses as the autoencoder for the MNIST test dataset, while RP t-SNE does seem to clearly improve the reconstruction losses for the COIL-20 test dataset. These results can be explained on the basis of the following two observations. First, the sample size of the COIL-20 training dataset is relatively small making the networks susceptible to overfitting. Second, RP t-SNE obtains extra information from the t-SNE cost due to the combined backpropagation. These observations imply that both networks might learn the noise of the data for small sample sizes, but RP t-SNE also learns to preserve the local structure in the bottleneck layer making it less susceptible to overfit on the reconstruction loss. Hence, the t-SNE cost can also be seen as a kind of regularization on the reconstruction loss.

To illustrate the improved reconstruction performance by RP t-SNE, we present in Figure 9 the original and reconstructed images of four random samples from the COIL-20 test dataset. These images clearly show that RP t-SNE is able to distinguish more details of the images than the autoencoder for the 2- and 10-dimensional projections. However, from the results we also observe that reconstructions of these images by RP t-SNE might worsen, if the projection dimensionality becomes too high. This is likely because of the similar generalization performances for high projection dimensionalities (due to the less severe crowding problem). Implying that the marginal increase in information from the t-SNE cost decreases with its projection dimensionality, and thus the regularization on the reconstruction loss becomes less effective for higher projection dimensionalities.

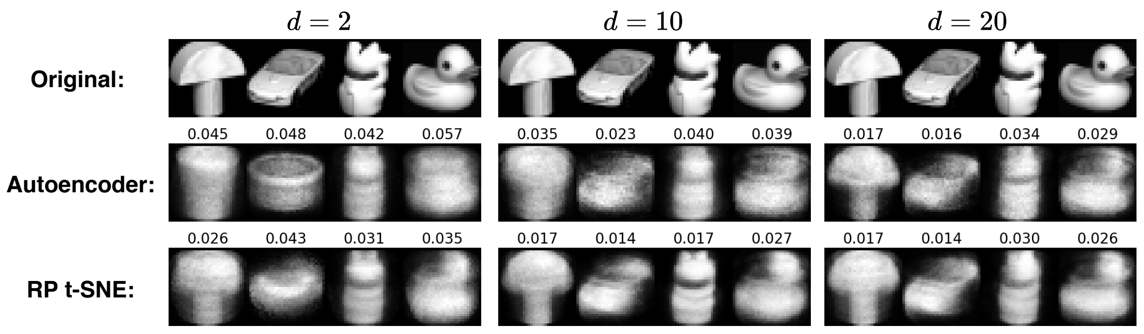


Figure 9: Four random sampled images from the COIL-20 test dataset. The middle and lower rows contain the reconstructed images by the autoencoder and RP t-SNE respectively with its reconstruction loss above the image, where d denotes the dimensionality of the projection in the bottleneck layer.

Table 3: Reconstruction losses of the autoencoder, and RP t-SNE on the MNIST and COIL-20 test datasets.

	Reconstruction losses					
	MNIST			**COIL-20**		
	2D	**10D**	**20D**	**2D**	**10D**	**20D**
Autoencoder	**0.039**	**0.017**	**0.010**	0.031	0.028	0.021
RP t-SNE	**0.039**	0.019	0.011	**0.026**	**0.018**	**0.020**

6 Conclusion

In this paper we investigated whether it is possible to combine the reconstruction ability from autoencoders and the local structure preservation ability from parametric t-SNE into one autoencoder network. To investigate whether the performances of these abilities are able to be retained or improved in a single autoencoder network, we formulated the following research question: *"To what extent are the abilities of autoencoders and parametric t-SNE retained or improved in an autoencoder network that optimizes both the reconstruction loss and t-SNE cost?"*.

From the results, we find that RP t-SNE has similar generalization performances as parametric t-SNE. Moreover, these techniques provide overall the best generalization performances. On the other hand, RP t-SNE provides similar reconstruction performances as autoencoders for large sample sizes, and for smaller sample sizes the results show a clear improvement of the reconstruction ability. However, simple benchmarks as kernel t-SNE and PCA provide generalization performances on par with RP t-SNE for low and high projection dimensionalities respectively due to the less severe crowding problem for higher projection dimensionalities.

In conclusion, to answer the research question, RP t-SNE retains the abilities from autoencoders and parametric t-SNE fairly well, and even improves the reconstruction ability for smaller sample sizes. Furthermore, RP t-SNE has a similar or stronger performance than the implemented benchmarks, but it has to be noted that this stronger performance is only noticeable for relatively low projection dimensionalities. Nonetheless, we have introduced a technique that trains a single autoencoder network able to provide us with a parametric mapping for projections that preserve the local structure of high-dimensional input data, and simultaneously able to reconstruct these projections with equal or higher precision than regular autoencoders.

A limitation of this paper is the setting of the trade-off parameter for RP t-SNE. We opted for the generalization error as the criterion for cross-validation, but this means that the choice is in favor of improving generalization rather than reconstruction performance. Ideally, the magnitudes of the reconstruction loss and t-SNE cost are similar such that the regularized t-SNE cost could act as a criterion measuring both costs. In this research, we measured the reconstruction loss with the mean squared error. However, since we normalize our data to be in the range of [0, 1], we can also measure the reconstruction loss with the cross-entropy error as done by Salakhutdinov and Hinton (2007), which is an log-based entropy measure just as the t-SNE cost. We thus propose to further investigate whether the regularized t-SNE cost could act as a valid criterion for cross-validation, if the reconstruction loss is measured by the cross-entropy error.

Acknowledgements

I would like to thank our supervisor, prof. dr. Patrick Groenen, for the weekly meetings, helpful discussions and suggestions during this research.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

Bertsekas, D. P. (2014). *Constrained optimization and Lagrange multiplier methods*. Academic press.

Chollet, F. et al. (2015). Keras.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741.

Gisbrecht, A., Lueks, W., Mokbel, B., and Hammer, B. (2012). Out-of-sample kernel extensions for nonparametric dimensionality reduction. In *ESANN*.

Gower, J. C. (1966). Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3-4):325–338.

Groenen, P. J. and van de Velden, M. (2005). Multidimensional scaling. *Encyclopedia of Statistics in Behavioral Science*.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

Hinton, G. E. (2009). Deep belief networks. *Scholarpedia*, 4(5):5947.

Hinton, G. E. and Roweis, S. T. (2003). Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, pages 857–864.

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

Hofmann, T., Schölkopf, B., and Smola, A. J. (2008). Kernel methods in machine learning. *The Annals of Statistics*, pages 1171–1220.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4):295–307.

Kaski, S., Nikkilä, J., Oja, M., Venna, J., Törönen, P., and Castrén, E. (2003). Trustworthiness and metrics in visualizing similarity of gene expression. *BMC Bioinformatics*, 4(1):48.

Kindermann, R. (1980). Markov random fields and their applications. *American Mathematical Society*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv Preprint arXiv:1412.6980*.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243.

Lee, J. A. and Verleysen, M. (2009). Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72(7-9):1431–1443.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

Penrose, R. (1956). On best approximate solutions of linear matrix equations. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 52, pages 17–19. Cambridge University Press.

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

Salakhutdinov, R. (2015). Learning deep generative models. *Annual Review of Statistics and Its Application*, 2:361–385.

Salakhutdinov, R. and Hinton, G. (2007). Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics*, pages 412–419.

Sammon, J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 100(5):401–409.

Schubert, E. and Gertz, M. (2017). Intrinsic t-stochastic neighbor embedding for visualization and outlier detection. In *International Conference on Similarity Search and Applications*, pages 188–203. Springer.

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

Van Der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. In *Artificial Intelligence and Statistics*, pages 384–391.

Venna, J. et al. (2007). *Dimensionality reduction for visual exploration of similarity structures.* Helsinki University of Technology.

Welling, M., Rosen-Zvi, M., and Hinton, G. E. (2005). Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems*, pages 1481–1488.

A Additional information

In this appendix we present some additional information of our techniques. Specifically, Appendix A.1 describes how to obtain the kernel widths of t-SNE with binary search, Appendix A.2 gives more details about RBMs, and Appendix A.3 describes the computation of the trustworthiness, and continuity quality measures.

A.1 Binary search to obtain kernel widths for t-SNE

In Section 2.1 we mentioned binary search to find the kernel width σ_i, which given a datapoint $\mathbf{x}_i$ induces a conditional probability distribution P_i, corresponding to the conditional probabilities $p_{j|i}$ as given in Equation (1), over all of the other datapoints. Finding the appropriate σ_i is important, since it is unlikely that a single value of σ_i would be optimal for all datapoints. This is because the density of the data is likely to vary. This means that a smaller value of σ_i is more appropriate in dense regions than for sparse regions and vice versa.

Furthermore, the probability distribution P_i has a Shannon entropy $H(P_i)$ measured in bits, which is defined as

$$H(P_i) = -\sum_{j}^{n} p_{j|i} \log_2 p_{j|i}. \tag{23}$$

The Shannon entropy monotonically increases as σ_i increases. Therefore, binary search can be used to find the value of σ_i which produces a P_i with a user-specified perplexity $Perp(P_i)$. Hinton and Roweis (2003) defines the perplexity as

$$Perp(P_i) = 2^{H(P_i)}. \tag{24}$$

Notice that given $\mathbf{x}_i$ and σ_i we can compute the probabilities of P_i by using equation (1). Let $P_i = P(\mathbf{x}_i, \sigma_i)$, then we can write $H(P_i)$ as $H(P(\mathbf{x}_i, \sigma_i))$, whereby $H(P(\mathbf{x}_i, \sigma_i))$ increases as σ_i increases. Moreover, given an user-specified perplexity $Perp$ the entropy $H_0 = \log_2(Perp)$, can be seen as the desired Shannon entropy for a datapoint $\mathbf{x}_i$. We thus optimize σ_i with respect to H_0 by using binary search. An algorithm for this procedure is given in Algorithm 1.

Algorithm 1 Binary search to obtain kernel widths with an user-specified perplexity

Input: User-specified perplexity $Perp$, tolerance ϵ, maximum number of tries T, and input data $\mathbf{X}$.

Output: The kernel widths corresponding to an user-specified perplexity, $\boldsymbol{\sigma}$.

```
$H_0 \leftarrow \log_2(Perp)$                                    ▷ Desired Shannon entropy
$\boldsymbol{\sigma} \leftarrow \emptyset$                        ▷ Initialisation set of widths $\sigma_i$
for $i \in \{1, 2, ..., |\mathbf{X}|\}$ do
    $\sigma_i \leftarrow 1$; $\sigma_{min} \leftarrow -\infty$; $\sigma_{max} \leftarrow +\infty$; $tries \leftarrow 0$    ▷ Initialisation for each datapoint
    $H_i \leftarrow H(P(\mathbf{x}_i, \sigma_i))$                 ▷ Note: $H(P(\mathbf{x}_i, \sigma_i))$ is increasing in $\sigma_i$
    while $|H_i - H_0| > \epsilon$ AND $tries < T$ do
        if $H_i - H_0 < 0$ then
            $\sigma_{min} \leftarrow \sigma_i$                    ▷ Increase $\sigma_i$ if $H_i - H_0 < 0$
            if $\sigma_{max} = +\infty$ then
                $\sigma_i \leftarrow 2\sigma_i$
            else
                $\sigma_i \leftarrow (\sigma_i + \sigma_{max})/2$
            end
        else
            $\sigma_{max} \leftarrow \sigma_i$                    ▷ Decrease $\sigma_i$ if $H_i - H_0 \geq 0$
            if $\sigma_{min} = -\infty$ then
                $\sigma_i \leftarrow \sigma_i/2$
            else
                $\sigma_i \leftarrow (\sigma_{min} + \sigma_i)/2$
            end
        end
        $H_i \leftarrow H(P(\mathbf{x}_i, \sigma_i))$
        $tries \leftarrow tries + 1$
    end
    $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} \cup \sigma_i$
end
return $\boldsymbol{\sigma}$
```

A.2 In depth overview of Restricted Boltzmann Machines

A RBM is a special case of a Markov random Field, hence the joint distribution over all nodes is given by a Boltzmann distribution that corresponds to an energy function $E(\mathbf{v}, \mathbf{h})$. As stated by Van Der Maaten (2009), the most common choice of $E(\mathbf{v}, \mathbf{h})$ is a linear function over the states of the visual nodes $\mathbf{v}$ and hidden nodes $\mathbf{h}$ given by

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i,j} W_{ij} v_i h_j - \sum_i b_i v_i - \sum_j c_j h_j, \tag{25}$$

where W_{ij} denotes the weights of the connection between the nodes v_i and h_j, b_i the bias on node v_i, and c_j the bias on node h_j. Moreover, the states of the visual and hidden notes are conditionally independent from each other given one-another. Combining this with equation (25) we can derive $p(v_i = 1|\mathbf{h})$, and in a similar way $p(h_j = 1|\mathbf{v})$ as the following conditional probabilities:

$$p(v_i = 1|\mathbf{h}) = \sigma(b_i + \sum_j W_{ij} h_j), \text{ and} \tag{26}$$

$$p(h_j = 1|\mathbf{v}) = \sigma(c_j + \sum_i W_{ij} v_i). \tag{27}$$

In these equations $\sigma(z) = 1/(1 + e^{-z})$ is defined as the sigmoid function.

The objective of RBMs is to learn the weights W, and the biases $\mathbf{b}$ and $\mathbf{c}$ in such a way that the probability distribution over the visual nodes under the model $P_{\text{model}}(\mathbf{v})$, is close to the observed data distribution $P_{\text{data}}(\mathbf{v})$. Thus, we would like to minimize the KL-divergence $KL(P_{\text{data}}(\mathbf{v})||P_{\text{model}(\mathbf{v})})$ between these two distributions. The gradient with respect to W of this KL-divergence is given by

$$\frac{\partial KL(P_{\text{data}}(\mathbf{v})||P_{\text{model}(\mathbf{v})})}{\partial W_{ij}} = \mathbb{E}_{P_{\text{data}}}[v_i h_j] - \mathbb{E}_{P_{\text{model}}}[v_i h_j], \tag{28}$$

where $\mathbb{E}_{P_{\text{data}}}[\cdot]$ and $\mathbb{E}_{P_{\text{model}}}[\cdot]$ denote the expectation under the data and model distribution respectively. However, this gradient can not be computed, because $\mathbb{E}_{P_{\text{model}}}[v_i h_j]$ is intractable (Hinton and Salakhutdinov, 2006). So instead, we minimize the contrastive divergence (Hinton, 2002) which measures the tendency of the model distribution to diverge from the data distribution:

$$KL(P_{\text{data}}||P_{\text{model}}) - KL(P_{\text{recon}}||P_{\text{model}}), \tag{29}$$

where P_{recon} represents the probability distribution of the reconstructed visual nodes after one iteration of the RBM. Combining equations (26) and (27), we can estimate $\mathbb{E}_{P_{\text{recon}}}[v_i h_j]$ from samples obtained by Gibbs sampling (Geman and Geman, 1984). Thus, the approximate gradient of the contrastive divergence with respect to the weights W_{ij} can be computed, since it is given by

$$\mathbb{E}_{P_{\text{data}}}[v_i h_j] - \mathbb{E}_{P_{\text{recon}}}[v_i h_j]. \tag{30}$$

Furthermore, instead of modelling binary input data, real-valued input data can also be modelled by using Gaussian distributed visible units as described by Salakhutdinov (2015). In particular, we consider modelling visible real-valued nodes $\mathbf{v} \in \mathbb{R}^D$ whose mean is determined by the hidden units with the following conditional probabilities:

$$p(v_i = x|\mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp(-\frac{(x - b_i - \sigma_i \sum_j h_j w_{ij})^2}{2\sigma_i^2}), \text{ and} \tag{31}$$

$$p(h_j = 1|\mathbf{v}) = \sigma(c_j + \sum_i W_{ij}\frac{v_i}{\sigma_i}), \tag{32}$$

where $x \in \mathbb{R}$ is a value from the input data. The corresponding energy term for these conditional probablities is given by

$$E(\mathbf{v}, \mathbf{h}) = \sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_{i,j} h_j w_{ij}\frac{v_i}{\sigma_i}. \tag{33}$$

The gradient for the contrastive divergence with respect to the weights W_{ij} using these conditional probabilities and energy term takes the following form:

$$\mathbb{E}_{P_{\text{data}}}[\frac{1}{\sigma_i}v_i h_j] - \mathbb{E}_{P_{\text{recon}}}[\frac{1}{\sigma_i}v_i h_j], \tag{34}$$

where σ_i^2 is set to one as done by Salakhutdinov and Hinton (2007) to obtain the same approximate gradient given in Equation (30).

A.3 Computing trustworthiness and continuity

- Trustworthiness $T(k) \in [0, 1]$: measures the "precision" of the projections $\mathbf{y}_i$, that is, to what extent do the k nearest neighbors of $\mathbf{y}_i$ correspond to the k nearest neighbors of $\mathbf{x}_i$. Mathematically, Kaski et al. (2003) defines $T(k)$ as

$$T(k) = 1 - \frac{2}{nk(2n - 3k - 1)} \sum_{i=1}^{n} \sum_{j \in U_i^{(k)}} (r(i, j) - k), \tag{35}$$

  where $r(i, j)$ represents the rank of datapoint $\mathbf{x}_j$ in the ordering from lowest to highest Euclidean distance from $\mathbf{x}_i$ and $U_i^{(k)}$ represents the set of points that are among the k nearest neighbors of $\mathbf{y}_i$, but not of $\mathbf{x}_i$. Notice that the term $\frac{2}{nk(2n-3k-1)}$ scales the measure to be in the range $[0, 1]$ if $k < \frac{n}{2}$ (Venna et al., 2007).

- Continuity $M(k) \in [0, 1]$: measures the "recall" of the projections $\mathbf{y}_i$, that is, to what extend are the k nearest neighbors of $\mathbf{x}_i$ retrieved in the k nearest neighbors of $\mathbf{y}_i$. The continuity $M(k)$ can be computed analogous to the trustworthiness by replacing $r(i, j)$ and $U_i^{(k)}$ in equation (35) with $\hat{r}(i, j)$ and $V_i^{(k)}$ respectively, resulting in the following expression:

$$M(k) = 1 - \frac{2}{nk(2n - 3k - 1)} \sum_{i=1}^{n} \sum_{j \in V_i^{(k)}} (\hat{r}(i, j) - k). \tag{36}$$

  In this case, $\hat{r}(i, j)$ represents the rank of datapoint $\mathbf{y}_j$ in the ordering from lowest to highest Euclidean distance from $\mathbf{y}_i$ and $V_i^{(k)}$ represents the set of points that are among the k nearest neighbors of $\mathbf{x}_i$, but not of $\mathbf{y}_i$, where again the term $\frac{2}{nk(2n-3k-1)}$ scales the measure to be in the range $[0, 1]$ if $k < \frac{n}{2}$.

B Additional Results

Due to space constraints we present in this appendix some relevant additional results of our research. Appendix B.1 shows the plots of the t-SNE cost of the three gradient descent methods, and Appendix B.2 the out-of-sample visualization of the COIL-20 test dataset by our parametric mappings.

B.1 Plots: t-SNE cost

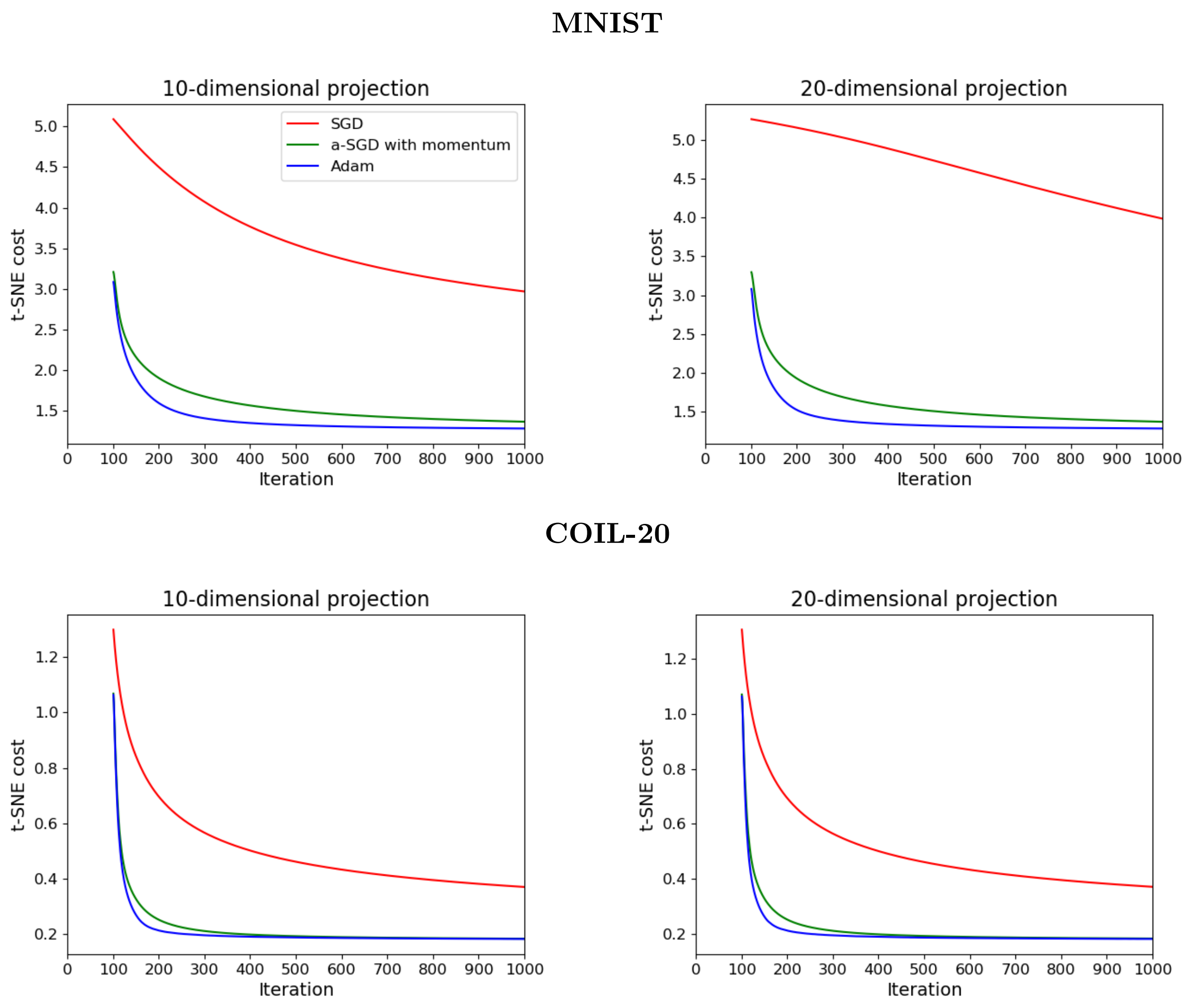


Figure 10: Plots of the t-SNE cost of the three gradient descent methods when applying t-SNE on the MNIST and COIL-20 datasets. The plots start at the 100th iteration which is after the early exaggeration

B.2 Visualizations: COIL-20 test dataset

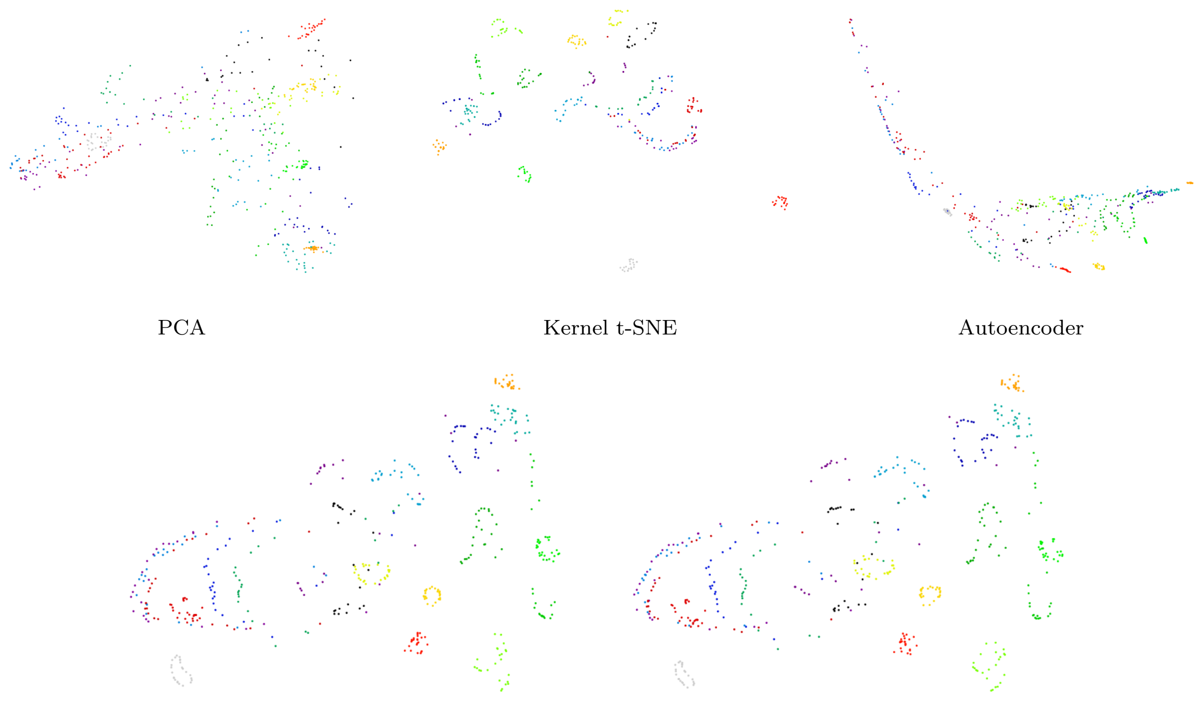


Figure 11: Visualizations of the COIL-20 test dataset by PCA, Kernel t-SNE, an autoencoder, Parametric t-SNE and RP t-SNE.

C Code description

In this appendix we present a brief description of the Python files used in this paper, which can be found at `https://github.com/StefanLam99/OOS__tSNE`. The descriptions of the classes for our implemented techniques, the classes to pretrain an autoencoder network, and the main and utility classes are given in Tables 4, 5, and 6 respectively.

Table 4: Classes for our implemented techniques.

Python file	**Description**
tSNE.py	Class which implements the t-SNE algorithm with three different gradient descent methods: SGD, a-SGD with momentum and Adam. Note that a-SGD with momentum is the same as the implementation by Maaten and Hinton (2008).
reg_tSNE.py	Class to make an object for the RP t-SNE model, it is able to finetune the autoencoder network, predict projections from input data and reconstruct the input data from projections. Note that depending on the value of the trade-off parameter θ, this class can also be used for parametric t-SNE, and regular autoencoders.
par_tSNE.py	Class to make an object for the parametric t-SNE model, it is able to finetune an encoder network and to predict projections from input data.
kernel_tSNE.py	Class to make an object for the kernel t-SNE model, it is able to train a parametric mapping with kernels and predict projections from input data.
pca_tSNE	Class to make an object for the PCA model, it is able to predict projections from input data.
kNN.py	Class to train a k-nearest neighbor classifier on given input data.

Table 5: Classes to pretrain an autoencoder network.

Python file	Description
RBM.py	Class that trains a RBM with Bernoulli distributed binary visual and hidden nodes. Used to pretrain the weights of a layer in the autoencoder network.
RBM_linear_hidden.py	Identical to **RBM.py**, but with Gaussian distributed hidden nodes.
RBM_linear_visible.py	Identical to **RBM.py**, but with Gaussian distributed visual nodes.
pretrain_autoencoder.py	Class that pretrains an autoencoder network using the previous mentioned RBM classes.

Table 6: Main and utility classes

Python file	Description
kcrossfold.py	Main to perform k-fold cross-validation to find the optimal trade-off paramerter for RP t-SNE with respect to the lowest generalization error.
main_results.py	Main to obtain the generalization errors and trustworthiness-continuity curves of our techniques.
main_trainNN.py	Main to pretrain and finetune the autoencoder, parametric t-SNE and RP t-SNE networks.
datasets.py	Class to make an object for a Dataset, which is able to preprocess and load several datasets.
utils.py	Contains different utility functions used for our techniques.