# Overview of t-Distributed Stochastic Neighbour Embedding and Demonstration using Real and Simulated Data

**Abstract**
This paper provides a brief theoretical the t-distributed Stochastic Neighbour Embedding (t-SNE) technique for visualizing high-dimensional data by mapping each datapoint to a two/three-dimensional representation. Dimensionality reduction is an invaluable tool in exploratory data analysis, allowing for underlying structure to be found in data. t-SNE represents similarity between datapoints by their Euclidean distance in the 2D/3D mapping. A description of the basics of t-SNE is discussed and some of its behaviour is explored using real and simulated data. t-SNE is used to map image data from the MNIST and COIL-20 datasets in order to evaluate the performance of the algorithm on real data. The technique is also applied to simulated data sets under varying hyper-parameters, to show why developing an intuition for its behaviour is important before exploring and interpreting real data.

## 1. Introduction

In exploratory data analysis, dimensionality reduction helps to gather useful information about the properties of the data by viewing it through lower-dimensional representations. Generally, data exploration is motivated by three types of tasks: (1) exploring local neighbourhoods, (2) viewing global geometry and finding clusters by inspection, (3) finding semantically significant patterns in the embedding to guide meaningful research directions [1].

Dimensionality reduction techniques transform a high dimensional dataset $X = \{x_1, \dots, x_n\}, x_i \in \mathbb{R}^p, p \geq 4$ into a two or three-dimensional representation $Y = \{y_1, \dots, y_n\}, y_i \in \mathbb{R}^q, q = 2 \text{ or } q = 3$, referred to as a map. The general aim of dimensionality reduction is to preserve as much of high-D structure as possible in the low-D map. With linear mappings seen in traditional dimensionality reduction techniques such as PCA [2] and MDS [3], the aim is to keep map points in the low-D representation far apart. However, this approach breaks down when the high-D data is distributed over a non-linear manifold [4]. t-SNE resolves data with this type of structural property by keeping very similar datapoints close together in the map, which linear mappings are typically incapable of achieving. In comparison to other non-linear dimensionality reduction methods, ie Sammon mapping, CCA, SNE, MVU, Laplacian Eigenmaps, etc [5], t-SNE claims to preserve both local and global structure of the data in a single map. In the following sections we review the theoretical basis for t-SNE, and explore its behaviour with real and simulated datasets.

### *1.1. Theory*

t-SNE is an extension of Stochastic Neighbour Embedding (SNE), which is described in the following section. The SNE algorithm starts by assigning similarities between datapoints in $x_i,\ x_j \in \mathbb{R}^p$ according to the conditional probability $p_{j|i}$ under the Gaussian distribution centred at $x_i$. The conditional $p_{j|i}$, with $\sigma_i$ centred at $x_i$, is given by

$$p_{j|i} = \frac{exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i} exp\,(-\|x_i - x_k\|^2/2\sigma_i^2)}, \quad p_{i|i} = 0 \tag{1}$$

whereas the conditional probability for determining the similarity pair-wise similarity of their corresponding map points $y_i,\ y_j \in \mathbb{R}^q$ in the map is computed as

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k\neq i} exp\,(-\|y_i - y_k\|^2)}, \quad q_{i|i} = 0 \tag{2}$$

Motivated by the observation that if $p_{j|i} = q_{j|i}$ then the map correctly models similarity between $x_i \text{ and } x_j$, SNE employs a gradient descent method to minimize the sum of some cost function over all data points, in order to minimize the mismatch between $p_{j|i} \text{ and } q_{j|i}$. The cost function is given by the Kullback-Leibler divergence

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}, \tag{3}$$

where $P_i$ is the distribution over all other datapoints in $\mathbb{R}^q$ centred at $x_i$, and $Q_i$ is the corresponding distribution over other map points centred at $y_i$. The gradient of the cost function has the form

$$\frac{\delta C}{\delta y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j). \tag{3}$$

To accelerate optimization and avoid poor local minima, an exponentially decaying sum of previous gradients, referred to as the momentum term $\alpha(t)$, is added to the gradient. $\eta$ indicates the learning rate. This modified gradient is defined as

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right). \tag{4}$$

The gradient update is initialized with randomly sampled Gaussians centred at the origin $Y$, and Gaussian noise is added to map points after each iteration to help find embeddings with better global organization. The penalty properties of the gradient is a reflection of the asymmetry in Kollback-Leibler divergence w.r.t. $p_{j|i}$ *and* $q_{j|i}$, which significantly penalizes distant data-points represented by close map-points, and a small penalty vice-versa. Consequently, SNE prioritizes embeddings which preserve local structure in the data, where locality varies according to the density of data in a particular region, given by the variance ${\sigma_i}^2$ of a high-D point $x_i$. The perplexity of P, is effectively a measure of the number of nearest neighbours of $x_i$, defined as

$$Perp(P_i) = 2^{H(P_i)}, \tag{5}$$

with $H(P_i)$ as the Shannon entropy of $P_i$ measured in bits

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}. \tag{6}$$

$Perp(P_i) = c$ is a user-defined constant, and the smooth and monotonic nature of the *Perp()* function means that the search for the mapping is robust to changes in the selected number of nearest neighbours within a certain range. The variance $\sigma_i$ is calculated for each $x_i$ through a binary search which yields $Perp(P_i) = c$.

### *1.1. t-distributed Stochastic Neighbour Embedding*

t-SNE modifies SNE in several ways. The Kullback-Leibler divergence is minimized based on the joint probability distributions $p_{ij}$ *and* $q_{ij}$ rather than $p_{j|i}\ q_{j|i}$,

$$C = \sum_i KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{7}$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{8}$$

Where $p_{j|i}$ and $p_{i|j}$ are defined the same as before. The symmetric joint probabilities are used so that the map location $y_i$ of an outlier point $x_i$ will have an impact on the organization of other points during gradient descent. The original SNE method tends to fail to show natural clusters in the data with data-points which are "moderately" spaced as gaps in the embedding. This is due to the fact that a low-D space has less "area" over which to distribute its elements compared to a higher-D space, generally referred to as the "crowding problem" [6]. An attempted remedy is with a uniform background model with a mixing proportion $\rho$, so that $q_{ij} > \frac{2\rho}{n(n-1)}$ [6]. However this fails if two parts of a cluster separate early on during gradient descent, since the attractive force between them is over-powered by the background term.

t-SNE uses the heavy tail of the single degree Student t-distribution (or the Cauchy) to model pair-wise distances in the map to alleviate this issue

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq i}\left(1 + \|y_i - y_j\|^2\right)^{-1}}, \quad q_{ii} = 0. \tag{9}$$

It reduces the unwanted attraction between map points that represent moderately distant points in the data space. Interestingly, this acts as an approximate inverse-square law which acts on distant clusters in much the same way as individual points, meaning that gradient descent is approximately scale invariant. Computationally, the gradient of the Student t-distrbution is less expensive than the Gaussian [7], described by

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1} \tag{10}$$

## 2. Methodology

We use the two datasets to verify the performance t-SNE. The MNIST dataset contains 60000 grayscale images of handwritten digits, each sample of size 28x28 pixels [8]. For our purposes, we consider a randomly selected subset of 1000 images in order to reduce the amount of required computation. The COIL-20 dataset contains images of 20 different objects viewed from 72 equally spaced orientations, producing 1440 images, each 64x64 pixels [9]. We use a subset of this, 72 images of the rubber ducky model, to evaluate t-SNE.

As a pre-processing step, we use PCA to reduce the dimensionality of the data to 45 in order to speed up pairwise distance calculations between data-points, with some noise filtering without severe distortion of inter-point distances. Each 45D representation is reduced to then reduced to 2D, shown in scatter-plot form. To assess how well each map preserves intra-class similarities, the class/label information of each datapoint is used to select a colour/symbol for the mapped points.

We also use simulated datasets consisting of 3D points clustered according to various patterns, in order to inspect how the low-D embeddings of the data vary with changes in hyper-parameters. For this we use an interactive visualization tool described in [1].

## 3. Results and Discussion

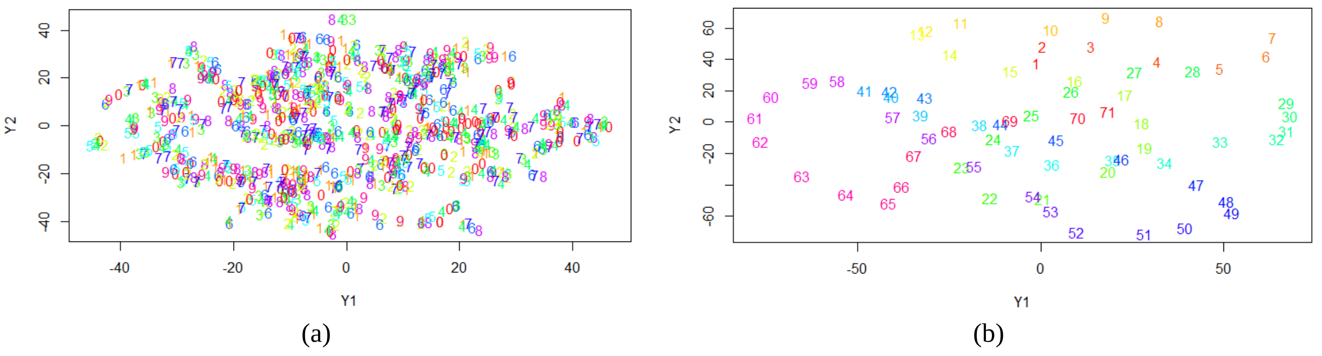
### *3.1. t-SNE performance on MNIST and COIL-20 datasets*

Attempts to reproduce results shown in [4] for the MNIST and COIL-20 benchmarks failed. Applying PCA to the data in the MNIST dataset resulted in 45 components which capture ~0.8 of the cumulative proportion of the variance. With a perplexity of 40, the Kallback-Leibler error for t-SNE was 0.755 after 10 epochs of 100 iterations each. Figure 1 features the resultant maps. t-SNE appears not to have returned any of the clusterings in contrast to those featured in [4] for the MNIST dataset.

Information lost due to PCA was ruled out as a reason, since the experiment was repeated without PCA with no improvement in results. The apparent noisiness of the data, as well as the large intra-class variance featured among digits in the MNIST dataset may also explain this issue, however, this would have been observed for [4] as well, since hyper-parameters from this experiment and the reference are similar. The only plausible explanations are the non-convexity of t-SNE means that the run from our experiment may have gotten stuck in bad local optima for this particular subset of data. However, several iterations over several different randomly selected subsets of the MNIST training dataset yielded similarly poor results.

The image data were also tested with normalized (by 255) and un-normalized values, without much difference. 3D embeddings of the data were also produced, and their various 2D axis-plane projects failed to show any improvement either. Lastly, two different R packages containing t-SNE (Rtsne and tsne) were tested and showed similarly poor results. The discrepancy in results is unclear, however, it may the case that the feature result in [4] was hand-selected from over thousands of iterations of t-SNE.

**Figure 1. (a)** 2D planar embedding of t-SNE map of MNIST data (b) 2D planar embedding of t-SNE map of COIL-20 rubber duck data



The same PCA parameters and t-SNE hyper-parameters were used for the test COIL-20 dataset, with much more encouraging results. Although the results do not look as good as those contained in [4], the embedding structure is similar enough to warrant comparison. Since the COIL-20 rubber-ducky data consists of a camera rotations about a fixed axis about a static object, the “loop” manifold result is expected. The discrepancy between the reference and experimental results may be explained by the effect that a much larger dataset would have on the organizing behavior of small clusters during gradient descent. In this case, seeing as the camera rotations in the data occur in consistent increments, the loop-iness of the embedding does not fully capture the topology of the higher dimensional data.

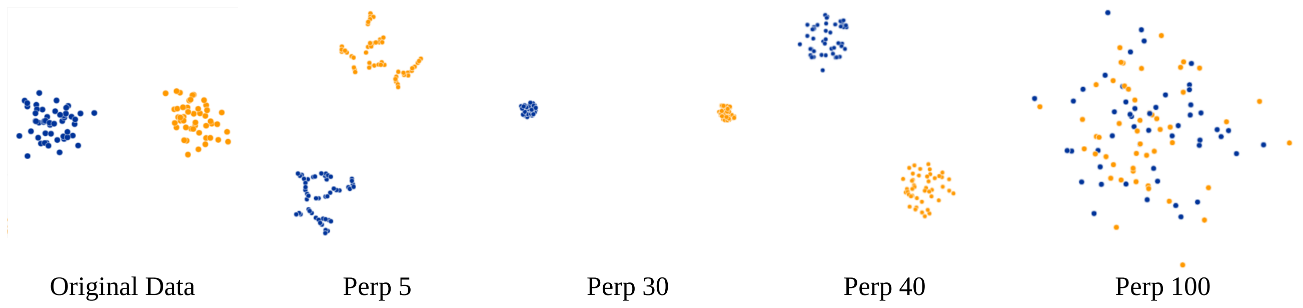
As an aside, it may be generally conjectured that data which contains apparent loops or circuits in its 2D/3D embedding may actually be describing a single mathematical object under various transformations (ie images from rotated views about a fixed object).

### *3.2. Demonstrating t-SNE behaviour using simulated experiment*

In this section we look at the organizing behaviour of the map points under varying perplexities and dataset distributions. Six simulated datasets are considered, each showing the performance of the algorithm under different conditions.

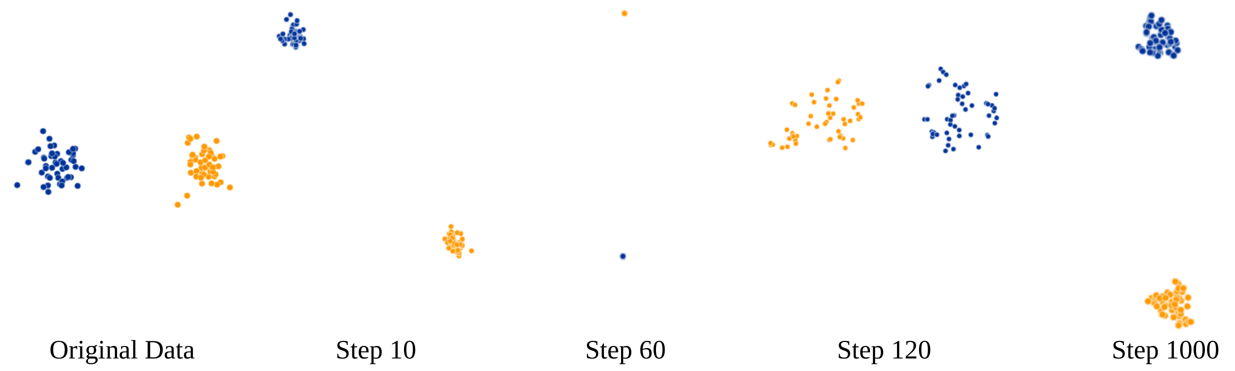
Although the recommended perplexity range is 5-50 [4], a wider range of perplexities were chosen to explore the organizing behavior of the algorithm. Figure 2 shows how map point dispersion is sensitive to perplexity. With a perplexity that is near the number of data points (ie Perp 100), t-SNE treats the entire dataset as nearest neighbours, resulting in an approximately uniformly dispersed map cluster. With perplexities between 5 and 50 more reasonably clustering is observed.

**Figure 2.** 2D t-SNE embedding after 5000 steps, with perplexities 5 – 100



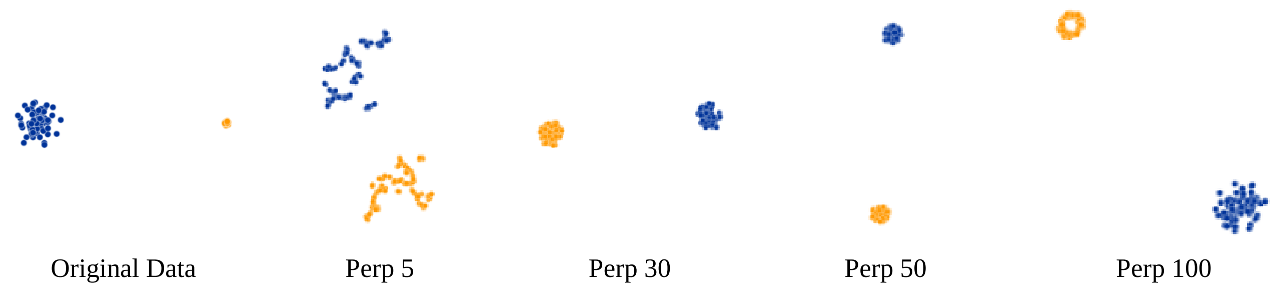
Fixing the perplexity at 30, the distribution of the map points are monitored during gradient descent iterations. The behaviour of the map points can be explained by the point-wise gradient for $y_i$. Interpreting the gradient through the analogy of springs acting between $y_i$ and all other map points $y_j$. The stiffness of the springs are proportional to the difference in conditional probabilities of the data-points and map-points (equation 10), and the "spring force" acts proportionally to the distance between the points in the map. The resultant force from each spring is attractive if the pair-wise similarity between $y_j$ and $y_i$ is larger in the map space than in the data space, and repulsive vice-versa.

**Figure 3.** 2D t-SNE embedding with perplexity 30 at various step times



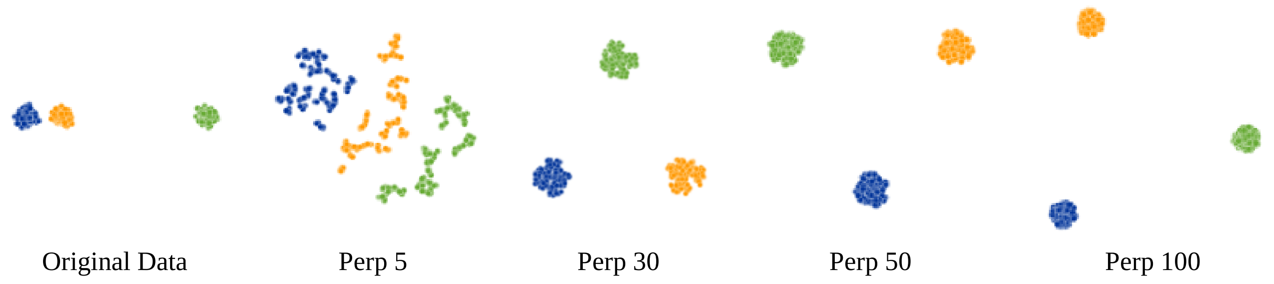
At step 60 in Figure 3, the clusters mutually collapse into singularities, however the "repulsive force" of the gradient update pushes the points apart until they settle on a map distribution which preserves the cluster distribution of the original data. However, Figure 4 shows how clusters with different densities can appear the same in the t-SNE map, even though they have different variances. Since $p_{j|i}$ is calculated with locally centred $\sigma_i$, t-SNE adjusts the notion of pair-wise distance according to regional density variations in the data. Therefore, relative cluster sizes cannot be accurately assessed with t-SNE.

**Figure 4.** Comparison of cluster density of incongruently dispersed clusters after 2D t-SNE mapping, blue cluster has standard deviation 10X that of orange



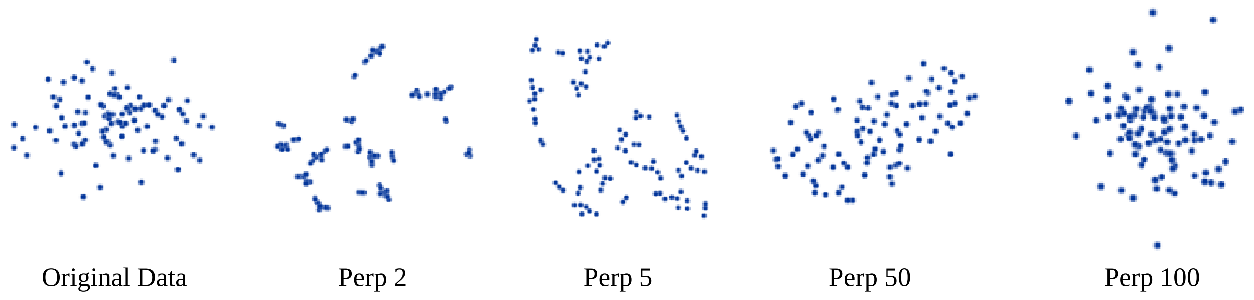
Further, as seen in Figure 5, relative inter-cluster distances do not readily become more apparent with a t-SNE embedding, which is sensitive to the perplexity value. Since perplexity is a global parameter, however, it cannot be fine-tuned to capture distances across all clusters, so adjusting perplexity to see the global distribution of clusters may not necessarily work.

**Figure 5.** Behaviour of t-SNE with respect to global structure of clusters



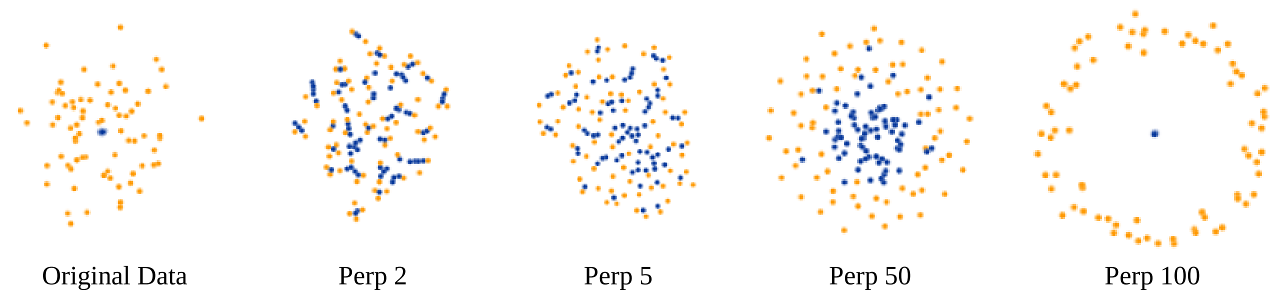
A common hazard of exploratory data analysis is seeing a signal where there's only noise. Figure 6 shows how at low perplexities, meaningless clumps tend to form in the map as local effects take over. Given that with a low perplexity a smaller number of nearest neighbours are considered, this means that smaller point-wise variances are calculated, giving a lower distance measure between neighbours and thus higher attraction in the embedding during gradient descent.

**Figure 6.** Effect of t-SNE transform on randomly generated data at varying perplexities



Lastly, Figure 7 shows how t-SNE can fail to capture the topology of the high-D data. With low perplexities, the tendency of t-SNE to expand highly dense regions in the data means the blue data-points expand initially due to repulsion from gradient descent, then begin to mix with the orange data points due to the clumping action described for low perplexities in the previous demonstration.

**Figure 7.** Effect of t-SNE transform on topological representation of higher-dimensional data



Interestingly, an outer ring of orange map points form for higher perplexities, as t-SNE tries to represent the approximate equidistance of the orange points from the inner blue points. In light of the many short-comings of t-SNE, its limitations can serve as launch points for future development. An example of this is allowing the perplexity value to be fine-tuned locally to account for the organization of clusters globally in the dataset.

## 4. Conclusion

Generally, the flexibility which makes t-SNE a powerful dimensionality-reduction technique is also the source of its failings. Although the non-convexity of the algorithm may be a reason for some data explorers to reject it, the local optima of t-SNE can still outperform other methods. As this review has shown, it is important to develop an intuition for how t-SNE behaves before applying it to real-world high-dimensional data, due to its non-convexity and sensitivity to hyper-parameters. Experimentation with the MNIST and COIL-20 datasets demonstrates that t-SNE can show no structure when there is one, as with digit classes in the MNIST embedding, and may not properly represent structure even if it finds it, as with COIL-20 embedding. Hopefully, the brief overview provided by this paper captures the essentials of the theoretical strengths of the t-SNE algorithm, as well as its practical limitations.

## 5. References

[1] D. Smilkov, N. Thorat, C. Nicholson, E. Reif, F. B. Viégas, and M. Wattenberg, “Embedding Projector: Interactive Visualization and Interpretation of Embeddings,” 2016.

[2] K. V. Mardia, J. T. Kent, and J. M. Bibby., *Multivariate Analysis*, vol. 9. 1979.

[3] W. S. Torgerson, “Multidimensional Scaling: I. Theory and Method,” *Psychometrica*, vol. 17, pp. 401–419, 1952.

[4] L. Van Der Maaten and G. Hinton, “Visualizing Data using t-SNE,” *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.

[5] J. A. Lee and M. Verleysen, “Nonlinear dimensionality reduction of data manifolds with essential loops,” *Neurocomputing*, vol. 67, pp. 29–53, 2005.

[6] J. Cook, I. Sutskever, A. Mnih, and G. Hinton, “Visualizing Similarity Data with a Mixture of Maps,” *Int. Conf. Artif. Intell. Stat.*, pp. 67–74, 2007.

[7] L. van der Maaten, “Accelerating t-SNE using Tree-Based Algorithms,” *J. Mach. Learn. Res.*, vol. 15, pp. 3221–3245, 2014.

[8] LeCun Yann, Cortes Corinna, and Burges Christopher, “THE MNIST DATABASE of handwritten digits,” *Courant Inst. Math. Sci.*, pp. 1–10, 1998.

[9] S. Nene, S. Nayar, and H. Murase, “Columbia Object Image Library (COIL-20),” *Tech. Rep.*, vol. 95, pp. 223–303, 1996.

**Appendix**

```
library(caret)

library(doParallel)
library(tsne)
#library(Rtsne)
library(stats)
library(RnavGraphImageData)
library(R.matlab)
library(png)
library(plot3D)

# Enable parallel processing.
cl <- makeCluster(detectCores())
registerDoParallel(cl)

# Load the MNIST digit recognition dataset into R
# http://yann.lecun.com/exdb/mnist/
# assume you have all 4 files and gunzip'd them
# creates train$n, train$x, train$y  and test$n, test$x, test$y
# e.g. train$x is a 60000 x 784 matrix, each row is one digit (28x28)
# call:  show_digit(train$x[5,])   to see a digit.
# brendan o'connor - gist.github.com/39760 - anyall.org
load_mnist <- function() {
  load_image_file <- function(filename) {
    ret = list()
    f = file(filename,'rb')
    readBin(f,'integer',n=1,size=4,endian='big')
    ret$n = readBin(f,'integer',n=1,size=4,endian='big')
    nrow = readBin(f,'integer',n=1,size=4,endian='big')
    ncol = readBin(f,'integer',n=1,size=4,endian='big')
    x = readBin(f,'integer',n=ret$n*nrow*ncol,size=1,signed=F)
    ret$x = matrix(x, ncol=nrow*ncol, byrow=T)
    close(f)
    ret
  }
  load_label_file <- function(filename) {
    f = file(filename,'rb')
    readBin(f,'integer',n=1,size=4,endian='big')
    n = readBin(f,'integer',n=1,size=4,endian='big')
    y = readBin(f,'integer',n=n,size=1,signed=F)
    close(f)
    y
  }
  train <<- load_image_file('train-images-idx3-ubyte')
  test <<- load_image_file('t10k-images-idx3-ubyte')

  train$y <<- load_label_file('train-labels-idx1-ubyte')
  test$y <<- load_label_file('t10k-labels-idx1-ubyte')
}

show_digit <- function(arr784, col=gray(12:1/12), ...) {
  image(matrix(arr784, nrow=28)[,28:1], col=col, ...)
}

train <- data.frame()
test <- data.frame()

# Load data.
load_mnist()

# Normalize: X = (X - min) / (max - min) => X = (X - 0) / (255 - 0) => X = X / 255.
train$x <- train$x
```

```
# Setup training data with digit and pixel values with 60/40 split for train/cv.
inTrain = data.frame(y=train$y, train$x)
inTrain$y <- as.factor(inTrain$y)
trainIndex = createDataPartition(inTrain$y, p = 0.60,list=FALSE)
training = inTrain[trainIndex,]
cv = inTrain[-trainIndex,]

# Try t-SNE
idx = floor(runif(6000, 0, 30000))
X = training[idx,-1]
pca = prcomp(X, center=TRUE)
summary(pca) #45 principal components explains 80% of total variance
pc.use <- 45 # explains 80% of variance
#trunc <- pca$x[,1:pc.use] %*% t(pca$rotation[,1:pc.use])
#trunc <- scale(trunc, center = -1 * pca$center, scale=FALSE)
Xtrunc <- pca$x[,1:pc.use]

Xtsne = tsne(Xtrunc[1:1000,], perplexity=40, max_iter=500, k=3)
Ytsne = training[1:1000,1]
color = rainbow(10)
plot(Xtsne[,1],Xtsne[,2], type = "n", xlab = "Y1", ylab="Y2")
text(Xtsne[,1],Xtsne[,2], labels = Ytsne, col = color[Ytsne])

##
#image(x=1:64, y=1:64, z=matrix(as.numeric(faces$faces[,2]), ncol=64, nrow=64),
col= gray(seq(1,0,length.out=256) ),  xaxt="n", yaxt="n")
#box()
#image(x=1:28, y=1:28, z=matrix(as.numeric(trunc[2000,]), ncol=28, nrow=28), col=
gray(seq(1,0,length.out=256) ),  xaxt="n", yaxt="n")
#box()

DAT = matrix(nrow=71, ncol = 128*128)

for(i in 1:71){
  filename = paste("coil-20-proc\\obj1__",as.character(i), sep = "")
  filename = paste(filename, ".png", sep = "")
  img = readPNG(filename)
  DAT[i,] = img
}

pca.d = prcomp(DAT, centre=TRUE)
X2 = tsne(pca.d$x[,1:50], perplexity = 40, k = 3)
Y2 = 1:71
color = rainbow(71)
scatterplot3d(X2[,1],X2[,2], X2[,3])
text3D(X2[,1], X2[,2], X2[,3],labels = Y2, col = color[Y2])
par(mar=c(5,4,4,4))
plot(X2[,1],X2[,2], type="n", ylab="Y2", xlab = "Y1")
text(X2[,1], X2[,2], labels = Y2, col = color[Y2])
```