

## User's Guide for t-SNE Software

### Laurens van der Maaten

*Pattern Recognition and Bioinformatics Group  
Delft University of Technology  
Mekelweg 4, 2628 CD Delft, The Netherlands*

LVDMAATEN@GMAIL.COM

### Geoffrey Hinton

*Department of Computer Science  
University of Toronto  
6 King's College Road, M5S 3G4 Toronto, ON, Canada*

HINTON@CS.TORONTO.EDU

### Editor:

## 1. Introduction

In this document, we describe the use of the t-SNE software that is publicly available online from <http://lvdmaaten.github.io/lvdmaaten/tsne>. Please note that this document does not describe the t-SNE technique itself; for more information on how t-SNE works, we refer to (van der Maaten and Hinton, 2008). The user's guide assumes that the Matlab environment is used, however, the discussion in section 3 is also of relevance to the use of the t-SNE software in programs that are written in, e.g., R, Java, or C++.

This document describes two implementations of t-SNE that are available online: (1) a simple Matlab implementation, and (2) a fast binary Barnes-Hut implementation with wrappers in various languages. Implementations of t-SNE in other languages work very similarly. The use of the Matlab implementation is described in Section 2. Section 3 covers the use of the fast t-SNE implementation.

## 2. Simple Matlab implementation

The main purpose of the Matlab implementation of t-SNE is to illustrate how the technique works. The code contains a lot of comments, making it a useful resource in the study of the technique. Matlab's poor memory management probably makes the codes less appropriate for application on large real-world datasets. The code requires a working installation of Matlab. No additional toolboxes are required.

The syntax of the Matlab implementation of t-SNE is the following:

```
mappedX = tsne(X, labels, no_dims, init_dims, perplexity)
```

Herein,  $X$  denotes the  $N \times D$  data matrix, in which rows correspond to the  $N$  instances and columns correspond to the  $D$  dimensions. In case the labels are specified, the code plots the intermediate solution every ten iterations. The labels are *only* used in the visualization of the intermediate solutions: t-SNE is an unsupervised dimensionality reduction technique. If an empty matrix (`[]`) is specified instead of a `labels` vector, no intermediate solutions are shown. The dimensionality of the visualization constructed by t-SNE can be specified through `no_dims` (the default value for `no_dims` is 2).

Before running t-SNE, the Matlab code preprocesses the data using PCA, reducing its dimensionality to `init_dims` dimensions (the default value is 30). The perplexity of the Gaussian distributions that are employed in the high-dimensional space can be specified through `perplexity` (the default value is 30). The function returns a  $N \times \text{no\_dims}$  matrix `mappedX` that specifies the coordinates of the  $N$  low-dimensional datapoints.

We illustrate the use of the simple Matlab implementation with an example. The following example requires the file `mnist_train.mat` as available from the t-SNE website. Notice that the example takes approximately an hour to complete.

```
% Load data
load 'mnist_train.mat'
ind = randperm(size(train_X, 1));
train_X = train_X(ind(1:5000),:);
train_labels = train_labels(ind(1:5000));

% Set parameters
no_dims = 2;
initial_dims = 50;
perplexity = 30;

% Run t-SNE
mappedX = tsne(train_X, [], no_dims, initial_dims, perplexity);

% Plot results
gscatter(mappedX(:,1), mappedX(:,2), train_labels, 'o');
```

### 3. Barnes-Hut Implementation

The fast Barnes-Hut implementation of t-SNE that is available online was implemented in C++. This implementation uses a different algorithm, which is described in detail by van der Maaten (2014). To compile the code, please execute the following command in the terminal:

```
cd /path/to/bhtsne
g++ sptree.cpp tsne.cpp -o bh_tsne -O2
```

The code comes with a Matlab script is available that illustrates how the fast implementation of t-SNE can be used. The syntax of the Matlab script (which is called `fast_tsne.m`) is roughly similar to that of the `tsne` function. It is given by:

```
mappedX = fast_tsne(X, no_dims, initial_dims, perplexity, theta)
```

Most of the parameters are identical to those discussed in Section 2, which is why we do not discuss them in detail here. The parameter `theta` specified how coarse the Barnes-Hut approximation is: setting `theta` to 0 runs the original  $\mathcal{O}(N^2)$  t-SNE algorithm, whereas using higher values runs the  $\mathcal{O}(N \log N)$  with increasingly better constant. The value of `theta` should be between 0 and 1, and its default value is 0.5.

An example of the use of the fast (landmark) version of t-SNE is given below. This example may also take up to an twenty minutes to complete.

```

% Load data
load 'mnist_train.mat'

% Set parameters
no_dims = 2;
initial_dims = 50;
perplexity = 30;

% Run t-SNE
mappedX = tsne(train_X, [], no_dims, initial_dims, perplexity);

% Plot results
gscatter(mappedX(:,1), mappedX(:,2), train_labels, 'o');
    
```

The example embeds all 60,000 datapoints in the MNIST training set!

There are wrappers available of the C++ code in Matlab, Python, Torch, and R. If you want to use another language, it should be fairly easy to code one up yourself. The fast implementation of t-SNE assumes the data and the parameters to be specified in a file called `data.dat`. The file `data.dat` is a binary file of which the structure is outlined in Table 1. The data matrix  $X$  should be specified as a concatenation of the instances (i.e.,  $\{instance_1, instance_2, \dots, instance_N\}$ ). After executing the binary, the result of the algorithm is written in a file called `result.dat`, the structure of which is outlined in Table 2. The low-dimensional embedding `mappedX` is returned as a concatenation of instances.

<i>Value</i>	<i>Number of elements</i>	<i>Type</i>
$N$	1	int32
$D$	1	int32
<code>theta</code>	1	double
<code>perplexity</code>	1	double
<code>no_dims</code>	1	int32
$X$	$N \times D$	double

Table 1: Structure of the `data.dat` file.

<i>Value</i>	<i>Number of elements</i>	<i>Type</i>
$N$	1	int32
<code>no_dims</code>	1	int32
<code>mappedX</code>	$N \times no\_dims$	double
<code>landmarks</code>	$N$	int32
<code>costs</code>	$N$	double

Table 2: Structure of the `result.dat` file. The last two variables are there for legacy reasons, and need never be used.

## References

- L.J.P. van der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 15(Oct):3221–3245, 2014.
- L.J.P. van der Maaten and G.E. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.