User's Guide for t-SNE Software

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

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

Editor:

1. Introduction

In this document, we describe the use of the t-SNE software that is publicly available online from `http://ticc.uvt.nl/~lvdrmaaten/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++.
There are two implementations of t-SNE available online: (1) a simple Matlab implementation, and (2) a fast binary implementation. 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 7.4 (which is also known as Matlab 2007A). 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 dimensional-

ity 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 $N \times$ `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(1:5000);

% Set parameters
no_dims = 2;
init_dims = 30;
perplexity = 30;

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

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

3. Fast implementation

The fast version of t-SNE that is available online was implemented in C++. For large datasets, the fast version employs the random-walk version of t-SNE. The fast version of t-SNE employs Intel's Primitive Performance Libraries in order to optimize the computational performance of the implementation. As a result, the implementation will generally run faster on Intel CPUs than on other x86 CPUs (although the performance on, e.g., AMD CPUs is not bad either). Binaries for all major platforms (Windows, Linux, and Mac OS X) are available for download. Also, 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, landmarks, costs] = fast_tsne(X, no_dims, init_dims, landmarks, ...
                                         perplexity)
```

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 `landmarks` can specify either the ratio of landmark points that should be employed (between 0 and 1; default value is $\min(6000/N, 1)$), or the indices of the datapoints that should be employed as landmark points. The function returns the low-dimensional representation of the landmark points in `mappedX`. The function also returns a vector `landmarks` that contains the indices of the datapoints that were employed as landmark points[1]; the labels of the landmark points can thus be obtained through `labels(landmarks)`. Moreover, the function returns

1. Note that if you set the ratio of landmarks to 1, the points in `mappedX` are also perturbed.

a vector `costs` that contains the contribution to the cost function per datapoint (hence, `sum(costs)` is the Kullback-Leibler divergence between P and Q), and can be used to evaluate which datapoints are modeled poorly in the map.
An example of the use of the fast (landmark) version of t-SNE is given below. This example may also take up to an hour to complete.

```
% Load data
load 'mnist_train.mat'

% Set parameters
no_dims = 2;
init_dims = 30;
ratio_landmarks = .1;     % use 6,000 points
perplexity = 30;

% Run t−SNE
[mappedX, landmarks] = fast_tsne(train_X, no_dims, init_dims, ...
                                        ratio_landmarks, perplexity);
train_labels = train_labels(landmarks);  % select the right labels

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

The example uses all $60,000$ datapoints in the MNIST training set to compute the P-values, but only returns an embedding for $0.1 \times 60,000 = 6,000$ datapoints. The indices of these $6,000$ datapoints are returned in `landmarks`.
Although we do not provide wrappers in other languages than Matlab, 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. If you want to specify landmark points yourself, specify the number of landmarks N_l (an integer value larger than 1) as well a list of indices of landmark points at the end of the file. Otherwise, specify the ratio of landmark points `ratio_landmarks` as a real-valued number between 0 and 1. The data matrix `X` should be specified as a concatenation of the instances (i.e., $\{\texttt{instance}_1, \texttt{instance}_2, ..., \texttt{instance}_\texttt{N}\}$). 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.

Value	*Number of elements*	*Type*
N	1	int32
D	1	int32
`no_dims`	1	int32
`perplexity`	1	double
N_l or `ratio_landmarks`	1	double
`X`	$N \times D$	double
if N_l was specified: `landmarks`	N_l	double

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

Value	*Number of elements*	*Type*
N_l	1	int32
`no_dims`	1	int32
`mappedX`	$N_l \times$ `no_dims`	double
`landmarks`	N_l	int32
`costs`	N_l	double

Table 2: Structure of the `result.dat` file.

References

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.