Machine Learning HW07

Siffi Singh – 0660819

Program to modify implemented t-SNE code to symmetric SNE and analyze their differences.

OUTLINE

In this report, I will be explaining,

1. ***Explanation of Code Logic***
   - ***a. Converting t – SNE to Symmetric SNE.***
   - ***b. Plot embedding of t-SNE for different perplexity values.***
   - ***c. Plot embedding of Symmetric SNE for different perplexity values.***
   - ***d. Plot distribution of high and low-dimensional space for t-SNE and symmetric-SNE.***
2. ***Testing Performance and comparison of t-SNE and symmetric SNE***
3. ***Inferences from Result***
4. ***Conclusion***

1. Explanation of Code Logic

The goal is to take a set of points in a high-dimensional space and find a faithful representation of those points in a lower-dimensional space, typically the 2D plane. For given input data, which contains 2500 feature vectors with vector of length 784 for describing MNIST images, we have the code for t-SNE that could be used for dimensionally reducing these 784 features to two features for visualization. Using t-SNE code provided by Laurens van der Maaten, our main task in this homework has been to convert the t-SNE code to symmetric SNE. We prefer t-SNE because it is best suited for visualization of high-dimensional data. As we know that t-SNE preserves the local structure by keeping data from high dimension, the close point's closer and distant points further when transformed to low dimension.

However, before we begin with conversion of code from t-SNE to symmetric SNE we need to understand what is done in the code already implemented by Laurens van der Maaten in order to figure out the specific code piece in MATLAB to modify for conversion.

In the given code, there are three functions, **main.m, tsne.m** and **tsne_p.m.** Briefly, main.m consists of the reading of input data files and conversion of it to array, and calling of required functions for dimensionality reduction.

```
clear;
close all;
clc;
X_data = dlmread('mnist2500_X.txt');
labels = dlmread('mnist2500_labels.txt');

%%Dimensionality Reduction using T-SNE
ydata = tsne(X_data, labels, 2, 784, 5);
```

As we see, the main function reads input data files and labels file and calls **tsne()** function to perform dimensionality reduction. Next in **tsne.m,** we first normalize the data:

```
% Normalize input data
   X = X - min(X(:));
   X = X / max(X(:));
   X = bsxfun(@minus, X, mean(X, 1));
```

Then, we perform preprocessing using PCA:

```
% Perform preprocessing using PCA
   if ~initial_solution
       disp('Preprocessing data using PCA...');
       if size(X, 2) < size(X, 1)
           C = X' * X;
       else
           C = (1 / size(X, 1)) * (X * X');
       end
       [M, lambda] = eig(C);
       [lambda, ind] = sort(diag(lambda), 'descend');
       M = M(:,ind(1:initial_dims));
       lambda = lambda(1:initial_dims);
       if ~(size(X, 2) < size(X, 1))
           M = bsxfun(@times, X' * M, (1 ./ sqrt(size(X, 1) .* lambda))');
       end
       X = bsxfun(@minus, X, mean(X, 1)) * M;
       clear M lambda ind
   end
```

Next, we compute the pairwise distance in High-dimension and compute the joint probabilities in the following manner:

```
% Compute pairwise distance matrix
   sum_X = sum(X .^ 2, 2);
   D = bsxfun(@plus, sum_X, bsxfun(@plus, sum_X', -2 * (X * X')));

% Compute joint probabilities
   P = d2p(D, perplexity, 1e-5); compute affinities using fixed perplexity
```

Further, we make a call to the function **tsne_p()** where we will compute the probabilities in low dimension and plot the visualization which is done through the following lines of codes:

```
  ydata = tsne_p(P, labels, no_dims);
  % The function performs symmetric t-SNE on pairwise similarity matrix P
  % to create a low-dimensional map of no_dims dimensions
```

So, we run then algorithm for a thousand iterations and modify our data in every step, resulting in different and more separated plots.

```
  % Run the iterations
    for iter=1:max_iter

        % Compute joint probability that point i and j are neighbors
        sum_ydata = sum(ydata .^ 2, 2);
       num = 1 ./ (1 + bsxfun(@plus, sum_ydata, bsxfun(@plus, sum_ydata', -2
* (ydata * ydata')))); % Student-t distribution
        num(1:n+1:end) = 0; % set diagonal to zero
        Q = max(num ./ sum(num(:)), realmin); %normalize to get probabilities
  % Compute the gradients (faster implementation)
        L = (P - Q) .* num;
        y_grads = 4 * (diag(sum(L, 1)) - L) * ydata;
```

Hence, in every iteration we compute the pairwise distance in low-dimension and also compute the gradient, update the solution and momentum and print the progress.

```
  % Print out progress
        if ~rem(iter, 10)
            cost = const - sum(P(:) .* log(Q(:)));
            disp(['Iteration ' num2str(iter) ': error is ' num2str(cost)]);
        end
```

From the above computations, the values obtained are plotted based on labels and the results have been showed and compared in the sections to come.

1. a. Converting t – SNE to Symmetric SNE

So, now that we have the full explanation of t-SNE code given to us, we are prepared to make changes in the part of code that will convert the t-SNE to symmetric SNE.

1. There are different ways of computation for pairwise similarities in high and low dimension for t-SNE and symmetric SNE (look figure 1). For symmetric SNE, we keep the **sigma constant** , while in asymmetric SNE, the value of sigma changes, but because in the code given to us, the implementation is done for symmetric t-SNE. Hence, **for high dimensions we do not need to make any change,** as the code has been implemented for symmetric t-SNE which takes care of keeping the sigma constant in every calculation.

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

Figure1: On left, pairwise similarities for symmetric SNE in high dimension and on right, pairwise similarities for asymmetric t-SNE in low dimension.

2. So, the major difference here is in the computation of pairwise similarities in lower

dimensions, where the formula for both are as follows:

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

Figure 2: On left is, the pairwise similarities for symmetric sne in lower dimensions while on right, pairwise similarities for t-sne in lower dimensions.

This is where the change is needed in the code as well, as we can see from above explanation, the computation of pairwise similarities for t-SNE has been done like this:

```
% Compute joint probability that point i and j are neighbors
        sum_ydata = sum(ydata .^ 2, 2);
        num = 1 ./ (1 + bsxfun(@plus, sum_ydata, bsxfun(@plus, sum_ydata', -2
* (ydata * ydata')))); % Student-t distribution
```

3. In lower dimensions, we are using a student t-distribution to represent the pairwise distance as it ensures that the points that are closer to each other in high dimensions will stay closer to each other in low dimensions too, and the points that are far away in higher dimensions will stay further apart in lower dimensions after dimensionality reduction. We make the following change to convert this to symmetric SNE code.

```
% Compute joint probability that point i and j are neighbors
num = exp(-(1 + bsxfun(@plus, sum_ydata, bsxfun(@plus, sum_ydata', -2 * (ydata
* ydata'))))); % Normal distribution
```

★ Therefore, to convert t-SNE to symmetric SNE, we **change** the representation from **student t-distribution** in lower dimensions to **Normal distribution** in lower dimensions.

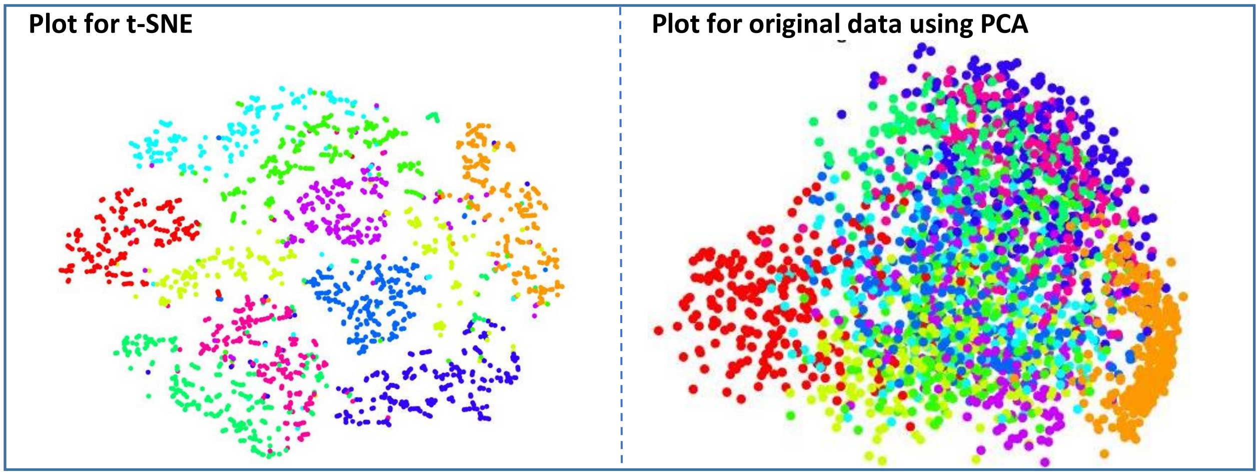
4. In addition, for asymmetric SNE and symmetric SNE, the gradient calculation changes, but here we do not need to take care of that as the code implemented is already for symmetric t-SNE. However, for stating the differences, we should know that gradient calculation changes in the following manner, when we convert and asymmetric SNE to symmetric SNE.

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

Figure 3: On left is the gradient calculation for symmetric SNE while on right, gradient calculation for symmetric SNE.

5. From t-SNE we get the plot that shows each of the classes separated clearly and because different perplexity values give different results, the execution of code has been done for ten different values 5,10, 15, 20...,50, and it has been concluded that the best separation could be seen for perplexity = 5 and in symmetric SNE the best plot could be seen at perplexity = 20.

Here is the plot for ten different classes of MNIST images, dimensionally recued from 784 feautres to two features.


Plot for t-SNE

Plot for original data using PCA

Figure 4: The left plot is for ten classes for handwritten digits 0-9, dimensionally reduced **using t-SNE** and the right plot is for the same data dimensionally reduced **using PCA**.

Hence, we can see in the left figure that the neighbouring structure has been preserved very well as we are transforming the data from higher dimension to lower dimension using t-SNE, while using PCA, we cannot preserve the local structure, instead we get a plot that rarely makes much sense with no clear seperation. PCA is more useful than t-SNE for compressing data to create a smaller number of features for input to predictive algorithms while t-SNE is a user-friendly method for visualizing high dimensional space.

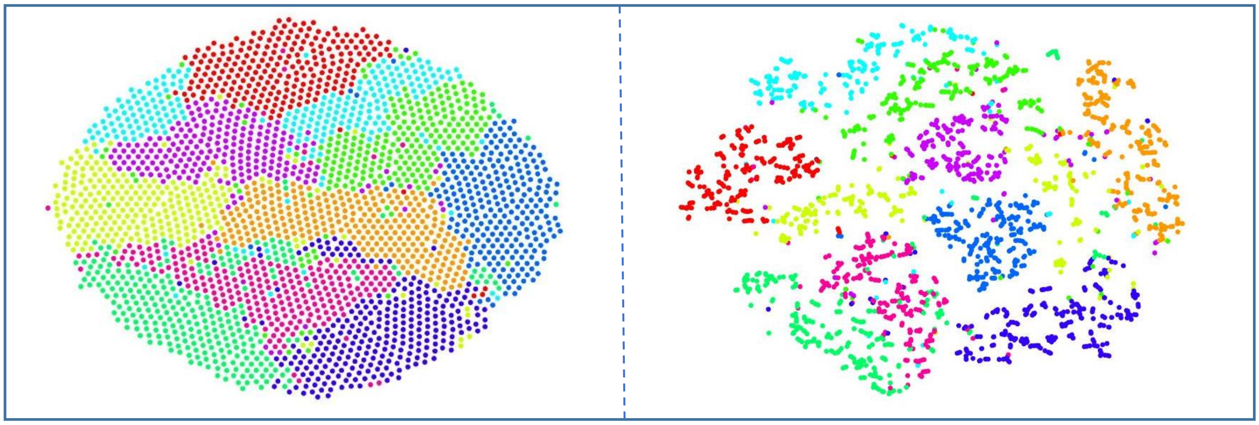


Figure 5: The left plot is for ten classes for handwritten digits 0-9, dimensionally reduced **using symmetric-SNE** and the right plot is for the same data dimensionally reduced **using t-SNE**.

The left plot has been plotted for perplexity value = 20 using t-SNE algorithm, and the right plot has been plotted for perplexity value = 5 using symmetric SNE algorithm. As from comparision based on visual performance, the best results have been obatined for the above mention perplexity values and the plots have been shown above.

t-SNE uses a Student-t distribution rather than a Gaussian to compute the similarity between two points in the low-dimensional space. t-SNE employs a heavy-tailed distribution in the low-dimensional space to alleviate both the crowding problem and the optimization problems of SNE. In our execution, it is observed that symmetric SNE seems to produce maps that are just as good as t-SNE, and sometimes even a little better.

1. b. Plot embedding of t-SNE for different perplexity values

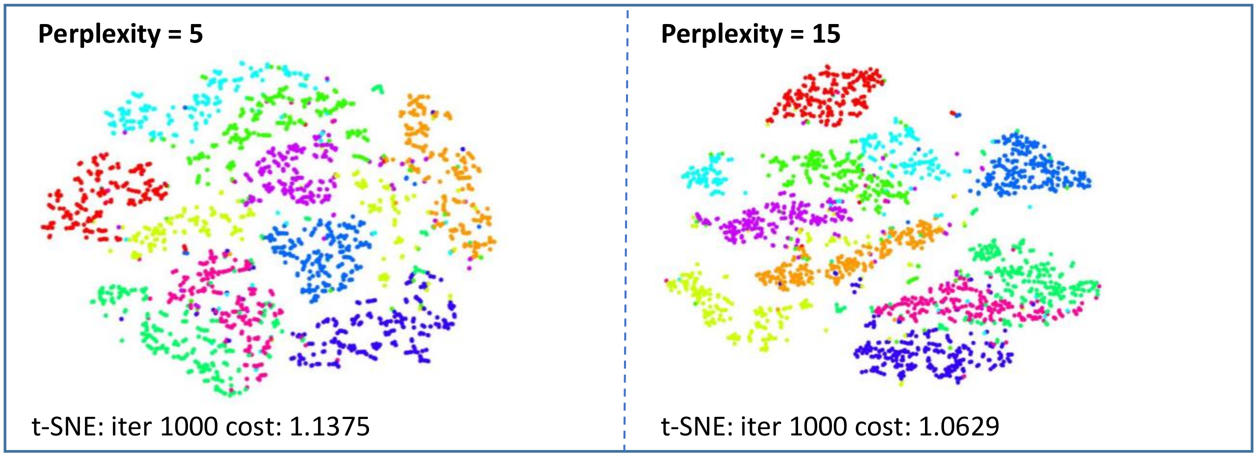
Before, we discuss the impact of perplexity on our plot, we must know, how we are using the perplexity and what it means. The perplexity is defined as a measure of the effective number of neighbors, defined as the two to the power of Shannon entropy, given by:

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

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

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

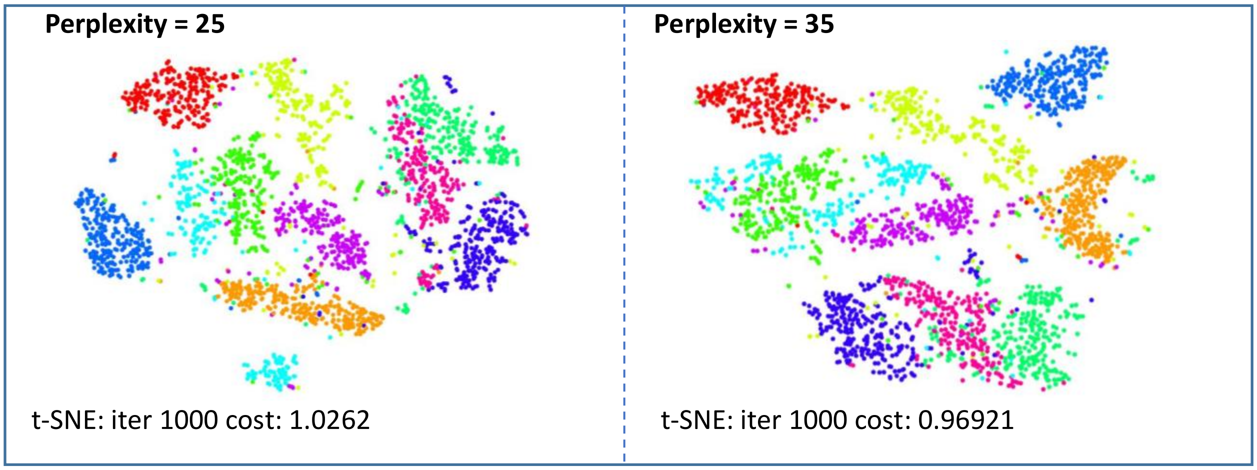
The perplexity can be interpreted as a smooth measure of the effective number of neighbors. The performance of SNE is fairly robust to changes in the perplexity, and typical values are between 5 and 50.


Perplexity = 5

t-SNE: iter 1000 cost: 1.1375

Perplexity = 15

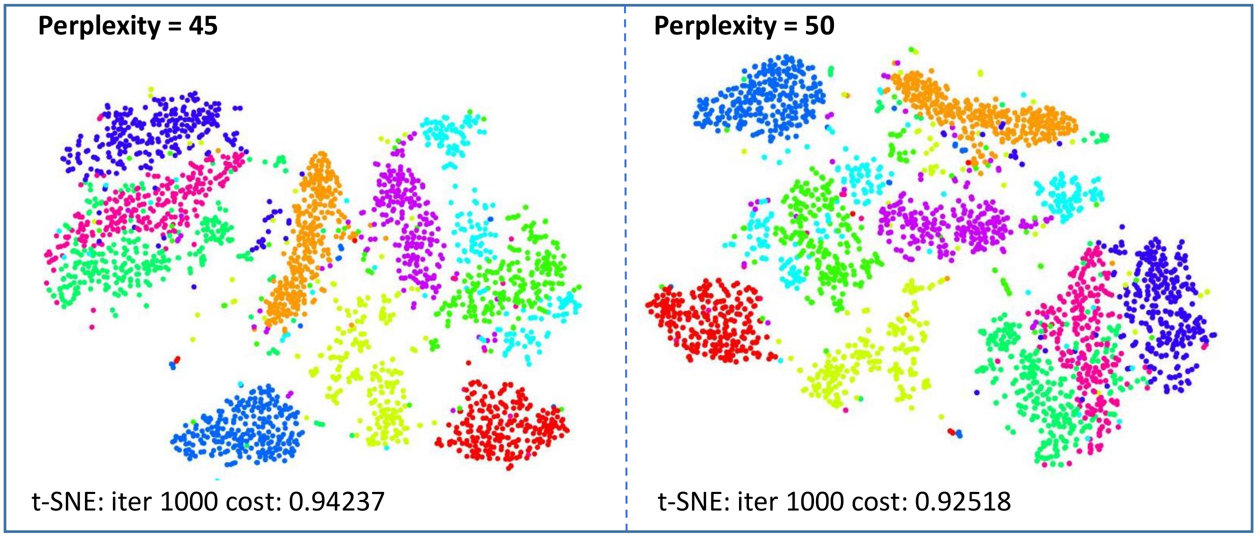
t-SNE: iter 1000 cost: 1.0629


Perplexity = 25

t-SNE: iter 1000 cost: 1.0262

Perplexity = 35

t-SNE: iter 1000 cost: 0.96921



Perplexity = 45

Perplexity = 50

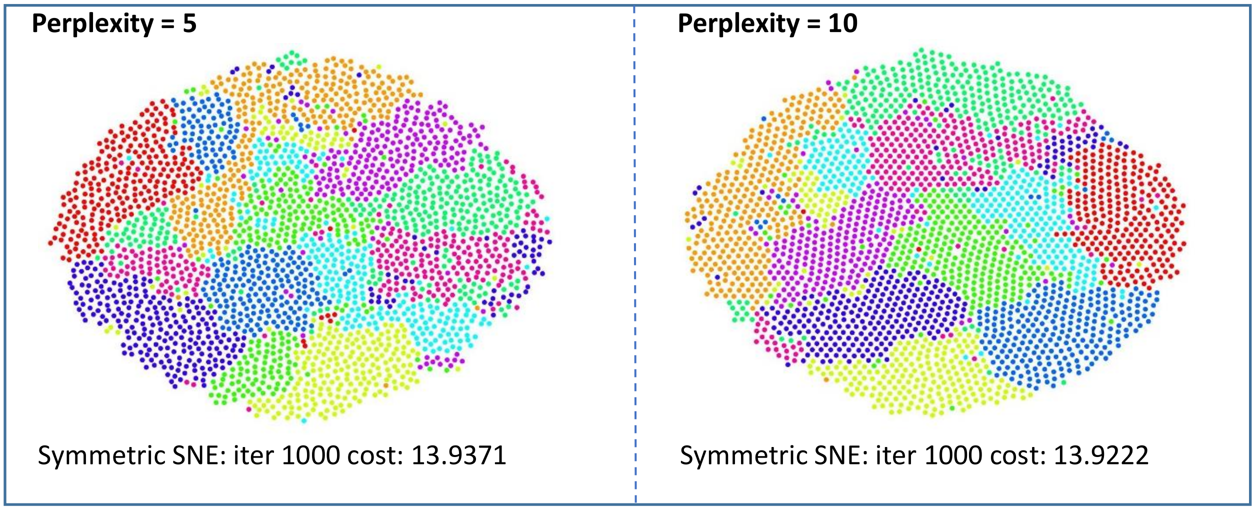
t-SNE: iter 1000 cost: 0.94237

t-SNE: iter 1000 cost: 0.92518

Figure 6: The images above show six different runs at perplexity 5, 15, 25, 35, 45, 50 for t-SNE. These runs were stopped after 1000 iterations.

1. c. Plot embedding of Symmetric SNE for different perplexity values

A second feature of t-SNE is a tunable parameter, "perplexity," which says (loosely) how to balance attention between local and global aspects of your data. The parameter is, in a sense, a guess about the number of close neighbors each point has. Getting the most from symmetric-SNE may mean analyzing multiple plots with different perplexities. There is also a possibility that the result might change for different runs.



Perplexity = 5

Perplexity = 10

Symmetric SNE: iter 1000 cost: 13.9371

Symmetric SNE: iter 1000 cost: 13.9222

Thus, to find a reliable value of perplexity, we need to rerun the code for different values and compare results until we find the best solution.

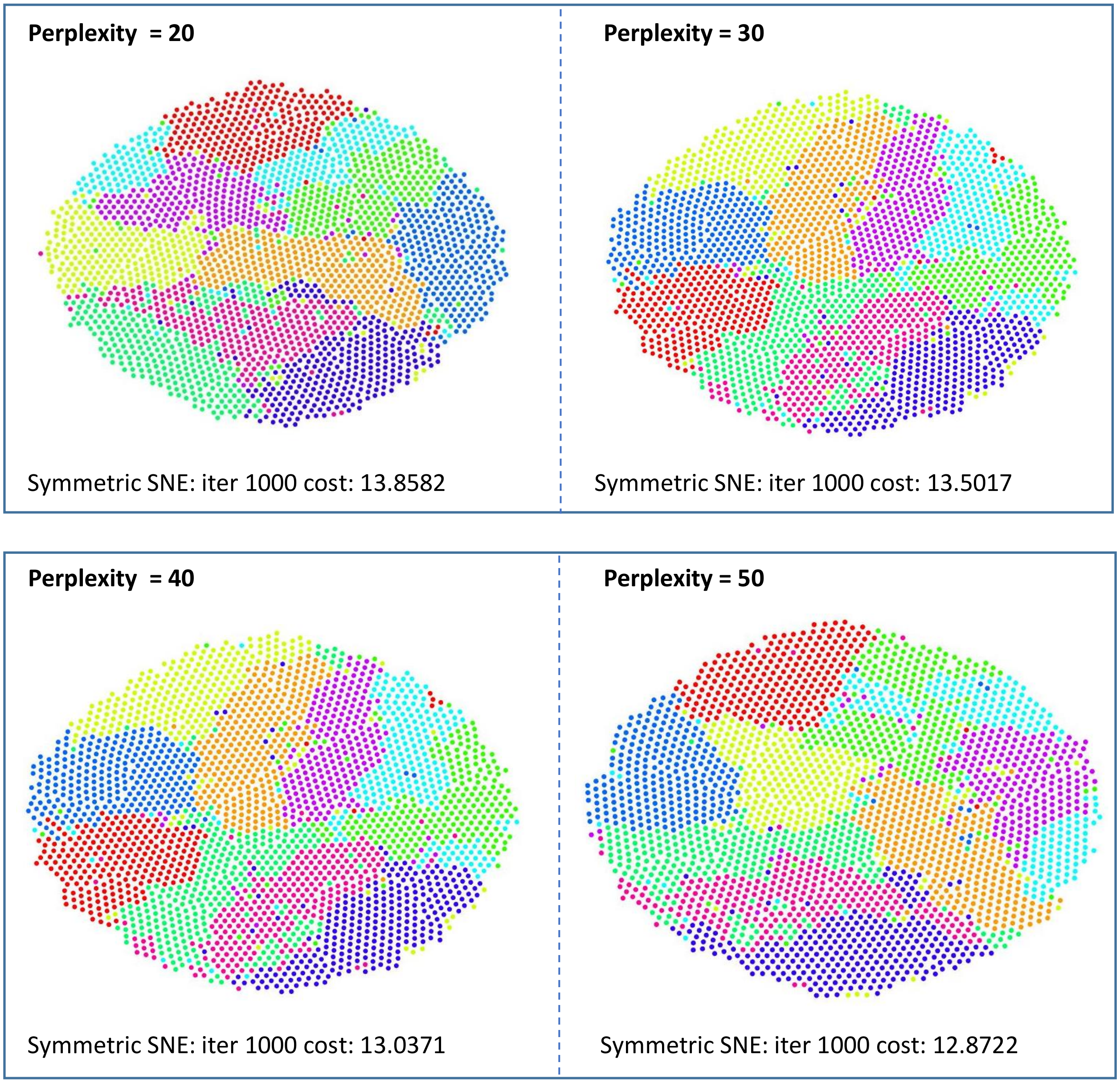


Figure 7: The t-SNE and symmetric SNE algorithm doesn't always produce similar output on successive runs, and there are additional hyper parameters related to the optimization process. The images above show six different runs at perplexity 5, 10, 20, 30, 40, 50. These runs were stopped after 1000 iterations.

Unfortunately, **there's no fixed number of steps that yields a stable result**. **Different data sets can require different numbers of iterations to converge.** SNE algorithm for dimensionality reduction depends a lot on parameters, especially on perplexity.

1. d. Plot distribution of high and low-dimensional space for t-SNE and symmetric-SNE

We need to visualize the distribution of pairwise similarities in both high-dimensional space and low-dimensional space based on both t-SNE and symmetric SNE.

```
%Compute mean and variance of distribution
  A = mean(D, 2);
  B = var(D);
  Mu_high = mean(mean(D));
  sigma_high = mean(var(D));
  x = [min(A):1:max(A)];
  norm = normpdf(x,Mu_high,sigma_high);
  plot(x,norm);
```

In case of **t-SNE** in high dimensional space, we have taken the similarity matrix ‘D’ and found the mean for the matrix and variance using the built-in function ‘mean’ and ‘var’ of MATLAB. So, below is the distribution for t-SNE of pairwise similarities in **High-Dimensional Space**, which is a **Gaussian distribution**.

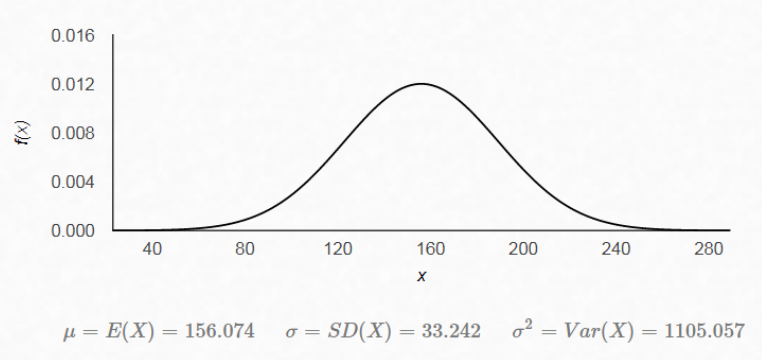


Figure 8: t-SNE in High Dimensional Space

In case of **t-SNE** in low dimensional space, we have taken the similarity matrix ‘num’ and calculated the mean for the matrix and variance using the built-in function ‘mean’ and ‘var’ of MATLAB. So, below is the distribution for pairwise similarities in **Low-Dimensional Space**, which is a **Student t-distribution**. t-SNE tends to expand denser regions of data.

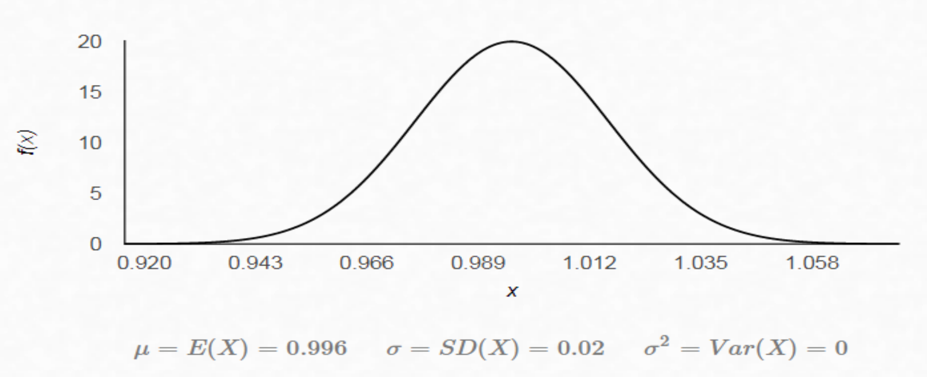


Figure 9: t-SNE in Low Dimensional Space

In case of **symmetric-SNE** in high dimensional space, we have taken the similarity matrix 'D' and found the mean for the matrix and variance using the built-in function 'mean' and 'var' of MATLAB. So, below is the distribution for symmetric-SNE of pairwise similarities in **High-Dimensional Space**, which is a **Gaussian distribution**.

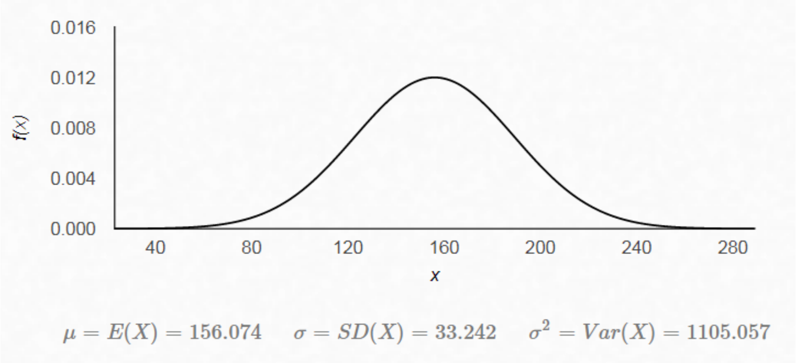


Figure 10: Symmetric SNE in High-Dimensional Space

In case of **symmetric-SNE** in low dimensional space, we have taken the similarity matrix 'num' and found the mean for the matrix and variance using the built-in function 'mean' and 'var' of MATLAB. So, below is the distribution for pairwise similarities in **Low-Dimensional Space**, which is a **Gaussian distribution**. Hence, there is a change specifically in the lower dimensions, where we are representing our data with Gaussian distribution in symmetric SNE while in t-SNE we represent it with Student t-distribution.

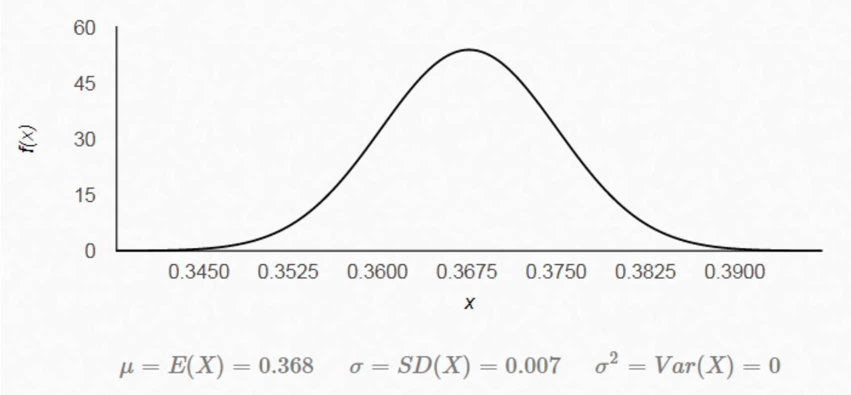


Figure 11: Symmetric SNE in Low-Dimensional Space

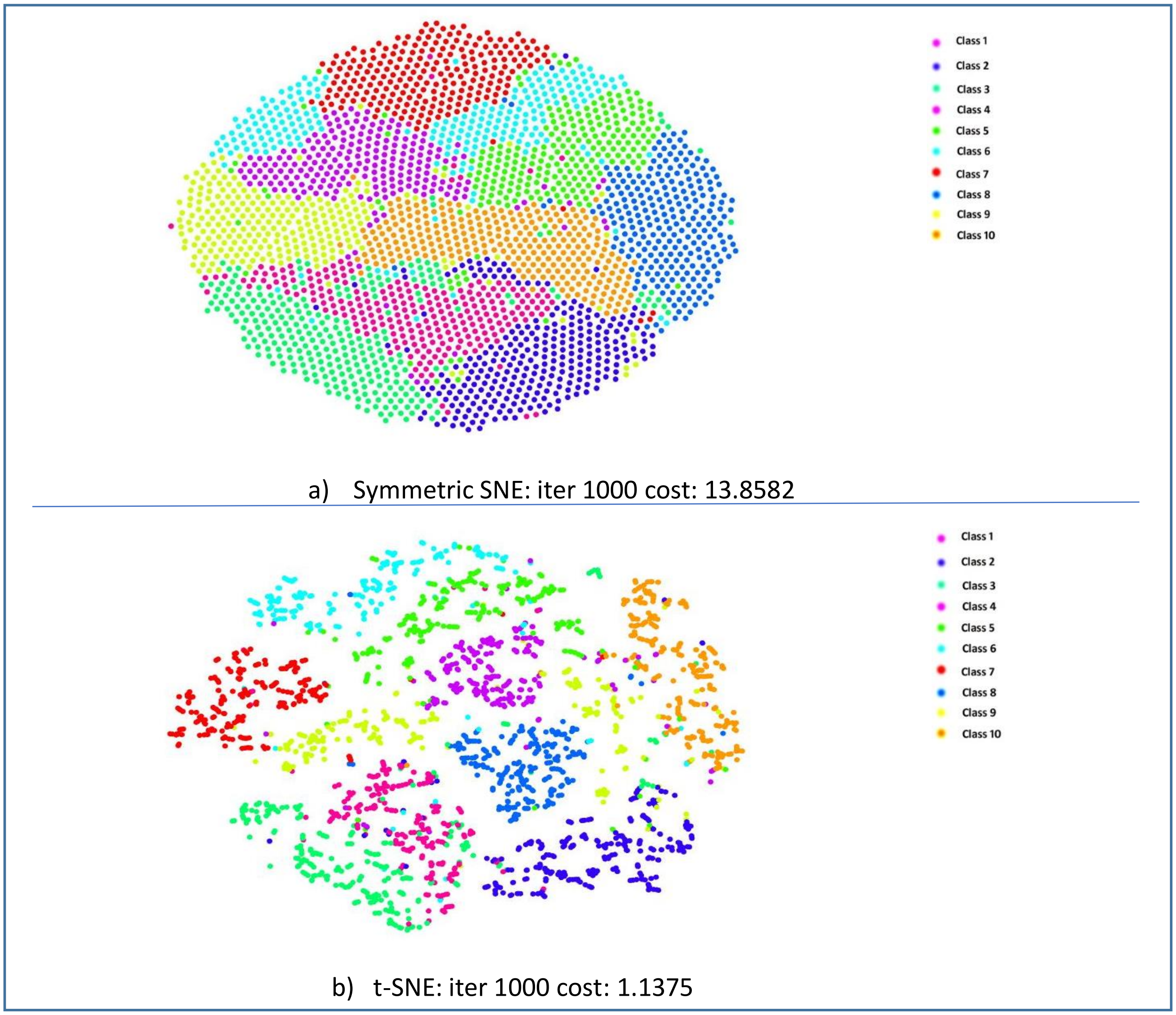
Hence, above is the visualization of distribution of pairwise similarities for both t-SNE and symmetric SNE.

2. Testing performance and comparison of t-SNE and symmetric SNE

1. The testing performance is measured with respect to the cost which is computed in the following manner:

```
 % Print out cost and progress
        if ~rem(iter, 10)
            cost = const - sum(P(:) .* log(Q(:)));
            disp(['Iteration ' num2str(iter) ': error is ' num2str(cost)]);
        end
cmap = colormap(hsv(10));
        % Display scatter plot (maximally first three dimensions)
        if ~rem(iter, 10) && ~isempty(labels)
            if no_dims == 1
                scatter(ydata, ydata, 28, cmap(labels+1,:), 'filled');
            elseif no_dims == 2
                scatter(ydata(:,1), ydata(:,2), 28, cmap(labels+1,:),
'filled');   else
                scatter3(ydata(:,1), ydata(:,2), ydata(:,3), 40,
cmap(labels+1,:), 'filled');            end
            axis tight
            axis off
            drawnow
        end
```

2. We also can see from the plot below that in t-SNE the plot is much more dispersed and clearly separated, thus preserving the neighborhood structure to a much greater extent.



a) Symmetric SNE: iter 1000 cost: 13.8582

b) t-SNE: iter 1000 cost: 1.1375

Figure 12: Plot for t-SNE, perplexity = 5 and for symmetric SNE, perplexity = 20.

```
 Q = max(num ./ sum(num(:)), realmin); % normalize to get probabilities
 P = max(P ./ sum(P(:)), realmin);    % make sure P-values sum to one
```

The cost error in case of best representation for t-SNE and symmetric SNE shows:

Dimensionality Reduction	Perplexity	Cost
Symmetric-SNE	5	13.8582
Symmetric t-SNE	20	1.1375

Table 1: Results for *best figure* with chosen perplexity and computed cost value.

While symmetric SNE shows why t-SNE shows a better performance, as just by making a small change of changing the student t-distribution to normal distribution, we get a change in the structure.

3. To demonstrate the impact of perplexity, I have started with low values as 5 and then moved to 10, 15, 20, …, 50. On visual comparison, I could conclude that in case of t-SNE, the best result is found at perplexity = 5 whereas in case of symmetric SNE, the best result is found at perplexity = 20. It can also be seen from the different results shown in 1.b and 1.c where the plots for t-SNE and symmetric SNE has been shown for different values of perplexity has been shown. There have been a few observances, one of which being that in t-SNE we see the cost decreases with increase in perplexity values, also in symmetric SNE, the cost decreases with increase in perplexity values.

4. We also come to know that in **high-dimensional space** both t-SNE and symmetric SNE, the pairwise similarities are represented by a **Gaussian Distribution** and in **low-dimensional space**, **t-SNE has student t-distribution** and **symmetric SNE** has **Gaussian distribution**.

3. Inferences from Result

1. The t-SNE algorithm adapts its notion of "distance" to regional density variations in the data set. As a result, it naturally expands dense clusters, and contracts sparse ones, evening out cluster sizes. To be clear, this is a different effect than the fact that any dimensionality reduction technique will distort distances. Rather, density equalization happens by design and is a predictable feature of t-SNE. Hence, local neighboring structure is efficiently preserved using t-SNE.

2. To see the effect of perplexity in action, I started by setting it to a low value of 5. The mapping of each point considers only its very closest neighbors. We tend to see many small groups of a few points. Now I'll rerun the t-SNE with a high perplexity of 50 and with this change we see the points are more evenly spread out, as though they are less-strongly attracted to each other.

3. The plot for distribution of pairwise similarities in both t-SNE and symmetric SNE shows that the probability distribution for t-SNE in low dimensional space has more variance, and as from class noes we know that in Low-D, small probability can be achieved by using "not-so-far" distance, therefore will be crowded (points do not need to be too far to achieve low probability).

→ use distribution with longer-tail, such that data should be further away in Low-D in order to achieve low probability.

→ Student t-distribution in Low-D (Gaussian distribution in High-D).

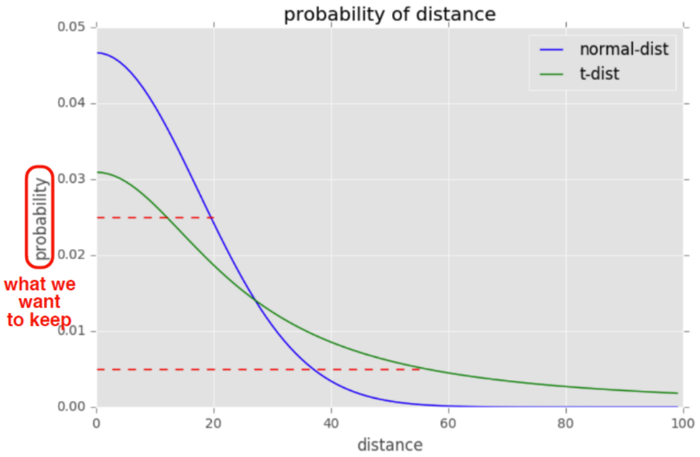


Figure 13: Comparison of student-t and normal distribution from class notes

4. Furthermore, t-SNE extends SNE with symmetric similarities and by using student's t-distribution in low-dimensional space. **Symmetric SNE explains why the heavy-tailed distribution and the symmetric similarity form in t-SNE lead to better performance.**

5. Symmetric SNE works best if we use different procedures for computing the p’s and the q’s. This destroys the property that if we embed in a space of the same dimension as the data, the data itself is the optimal solution.

4. Conclusion

1. symmetric t-SNE and symmetric SNE is a technique to turn a list high-dimensional vectors into a list of lower dimensional ones while keeping the relative similarity of things as close to the original (in high dimensional space) as possible. The only difference is in the representation in low dimensions.

2. The reason that t-SNE has become so popular is because it’s incredibly flexible, and can often find structure where other dimensionality-reduction algorithms cannot and this very flexibility makes it tricky to interpret. The algorithm makes all sorts of adjustments that tidy up its visualizations.

3. By studying how symmetric SNE and t-SNE behaves in simple cases, it’s possible to develop an **intuition** for what’s going on. Symmetric SNE has made me understand more in depth about t-SNE and s-SNE and has also motivated me to know in which cases is using t-SNE more suitable to use and which case is s-SNE more suitable. Symmetric SNE gives higher error as compared to t-SNE and the error seems to increase as we increase the perplexity values.