t-SNE Final Project

Michael Butler

December 2020

1 Introduction

Proper visualization of high dimensional data (i.e. more than 3 dimensions, in this context) is a ubiquitous and important problem across many domains. In higher dimensional space, it's harder for the practitioner to intuit whether two observations have some inherent similarity relative to other observations because it's challenging to grasp the underlying structure of the data in high dimensional space. As an example, consider a set of text documents represented by their word count vectors living in R^D. To assess similarity between documents, one could calculate euclidian distance in R^D, but this naive measure doesn't weigh salient words or deal with the sparcity of the vector. If the practitioner could reduce the dimensionality of the data, in such a way that preserves the high dimensional structure, they could visualize the relationship between text documents and between clusters of text documents.

The t-sne algorithm attempts to produce visualizations of high dimensional data in a low dimensional setting in such a way that preserves local structure (i.e. distances between neighbors) and global structure (distances across neighborhoods) (Van der Maaten and Hinton, 2008). t-SNE is a nonlinear dimension reduction technique, so unlike linear techniques (e.g. PCA or Classical MDS), t-SNE can keep neighbors on a high dimensional nonlinear manifold close together in the low dimensional representation. Compared to other non-linear dimension reduction techniques (e.g. Sammon Mapping, Isomap, LLE), t-SNE explicitly prevents the 'crowding problem', where different neighborhoods on the high dimensional manifold 'collapse' onto each other in the low dimensional embedding, i.e. they appear as one neighborhood (Linderman and Steinerberger, 2017).

At a high level, here's the t-SNE algorithm:

1. Calculate euclidean distances between raw each point.
2. Calculate the similarity matrix P in R^{NxN}, where p_{ji} roughly equals the probability that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a mean-field gaussian centered at x_i.

3. Initialize some low dimension embedding of the points (dimension is 1,2,3) and construct an analogous similarity matrix Q, except instead using a gaussian, we use a student-t distribution.

4. Minimize the following cost function using a penalized gradient descent method, updating Q after each step:

$$KL(P||Q)$$

where KL is the Kullback-Leibler divergence.

It's worth noting that the t-SNE algorithm builds upon the Stochastic Neighborhood Embedding algorithm by tweaking the cost function and computing Q with a t-distribution instead of a Gaussian (Hinton and Roweis 2002). As I'll show, these updates address crowding problem and simplify the numerical optimization.

t-SNE has a few notable weaknesses. While t-SNE has clear advantages for visualization and early stage data exploration, it is less interpretable than other nonlinear dimension reduction techniques, which potentially make it less preferable for more general dimensional reduction tasks (like reducing the data dimensionality, to a dimension greater than 3, to speed up a machine learning pipeline, for example). Like LLE and ISOMAP, t-SNE relies on a local linearity assumption, so when the data that is intrinsically high dimensional, i.e. where neighbors data don't approximately live on a linear plane, t-SNE struggles. Lastly, t-SNE cost function is non-convex, so finding the low dimensional embedding requires numerical optimization.

2 Method

In this section, I outline the steps t-SNE takes to produce a low dimensional embedding of high dimensional data. Suppose we're given a data matrix $X \in R^{NxD}$, where D is greater than 3. The goal of t-SNE is to construct a low dimensional representation $Y \in R^{NxK}$, where k is 1,2 or 3. To explain how t-SNE finds Y, I first review the Stochastic Neighborhood Embedding approach, as t-SNE merely builds upon this simpler algorithm (Hinton and Roweis, 2002).

2.1 Review of Stochastic Neighborhood Embedding

2.1.1 Similarity Matrix Construction

Given the raw data X, SNE constructs a similarity matrix $P \in R^{NxN}$ where p_{ij} represents the conditional probability, $p_{j|i}$, that x_i picks x_j as its neighbor if neighbors were picked in proportion to their probability density under a gaussian centered at x_i, or:

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\Sigma_{k \neq l} exp(-||x_k - x_l||^2/2\sigma_i^2)} \tag{1}$$

where σ_i equals the standard deviation of the gaussian, a hyperparameter the practitioner tunes. Note that $p_{i|i}$ gets set to zero. To clarify, the denominator sums all pairwise densities between distinct points.

SNE holds P fixed during the optimization, which means that σ_i is set before optimization. At a high level, σ_i^2 represents the variance of a gaussian centered at x_i, fit over x_i's neighbors. So if x_i belongs in a locally data dense region (i.e. lots of data points near x_i), we would expect σ_i^2 to be smaller. The practitioner explicitly pre specifies all the σ_i's through the perplexity parameter, which is essentially a smooth measure of effective number of neighbors:

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

where $H(P_i)$ is the Shannon Entropy, defined as:

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

So given that σ_i^2 is the only free variable in $p_{j|i}$ as all the x_i's are fixed, $Perp$ is essentially a function σ_i^2. Thus, when the user tells SNE to create a distribution P_i with perplexity equal to 5, for example, SNE must solve an inverse problem, i.e. find a σ_i that produces the desired perplexity. SNE solves this inverse problem by conducting a binary search over values of σ_i^2.

When perplexity is lower, σ_i is lower, causing the distances between points to have lower $P_{j|i}$. As I show with examples later, this tends to cause a higher number of smaller clusters to form in the low dimensional embedding.

The Q matrix is created in an identical fashion, though the variance in Q is set to $\frac{1}{\sqrt{2}}$. Conceptually, row i, in either P or Q, represents a bunch of likelihoods of samples generated by the conditional distribution centered around observation i.

2.1.2 Implicit Local Linearity Assumption

By constructing likelihoods with euclidean distances between points, SNE (and t-SNE as well) implicitly assumes that all high dimensional points live on a linear plane! So does SNE fall apart if the data live on a nonlinear manifold? Didn't I say this is a nonlinear dimension reduction technique? This implicit assumption only affects the performance of t-SNE and SNE if neighbors don't live on a plane. If x_i is far away from x_j and their geodesic distance (the true measure of similarity) is much larger than their euclidean distance (our measure of similarity), it doesn't really matter because $p_{i|j}$ will be really low regardless, as x_j lives in the tail of x_i's gaussian distribution. But if x_i and x_j are closer, then $p_{i|j}$ could be significantly different if geodesic distance were used. Notice also that the practioner can lower the perplexity hyper-parameter to tighten the gaussian's fit around each x_i if they are worried about violating the local linearity assumption.

2.1.3 SNE Optimization

SNE aims to maximize the similarity of the distributions described by Q_i and P_i, so minimizing the Kullback-Leibler divergence, an analytically tractable measure of the distance between distributions, is a reasonable function to minimize. Since we're dealing with N observations, or N different distribution comparisons, we minimize the sum of KL's from each comparison. Hence, SNE minimizes:

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

This cost function prioritizes pereserving local structure, or close pairwise similarities. Why? If $p_{i|j} = q_{i|j}$, then the log term becomes 0. Because of the $p_{i|j}$ term outside of the log, we get more bang for our buck by minimizing the discrepancy of pairs with high $p_{i|j}$.

To minimize this function, SNE considers a gradient descent algorithm, where we update our low dimensional embedding Y in the following manner:

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

The gradient, $\frac{\delta C}{\delta Y}$, comes out to:

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (p_{i|j} - q_{i|j} + p_{j|i} - q_{j|i})(y_i - y_j) \tag{5}$$

See Hinton and Roweis (2002) for details of gradient derivation. As for the other terms, η is the step size, $\alpha(t)(Y^{(t-1)} - Y^{(t-2)})$ adds momentum to an individual step, $\epsilon(t)$ is gaussian noise. So even if the gradient of the cost function approaches 0 (at some local minimum), the optimization algorithm may continue exploring the function if momentum is high. Momentum is higher if the algorithm recently descended a "steep hill" via $(Y^{(t-1)} - Y^{(t-2)})$, or if the hyper parameter $\alpha(t)$ is large. Typically, the practitioner tunes $\alpha(t)$ to be large at the beginning of the optimization and decreases it as time moves on, in the spirit of Simulated Annealing. Further the practitioner only adds gaussian noise, $\epsilon(t)$ early on in the optimization, also to avoid getting trapped at a local minimum. Lastly, the optimization is run several times, again to increase the probability that we find the global min of the objective function. As this optimization routine suggests, this objective function can be highly nonconvex, requiring many tuning knobs.

2.2 t-SNE Similarity Matrices

t-SNE tweak's the P and Q matrices in different ways and for different reasons. In P, the entry $p_{ij} = (p_{i|j} + p_{j|i})/2N$ where $p_{i|j}$ and $p_{j|i}$ are calculated using equation 1. This implies that t-SNE continues to rely on the local linearity assumption. By reframing each entry as a joint probability, we can view all

the entries of P as likelihoods of samples generated by a single joint distribution. This reformulation makes P symmetric, i.e. $P_{ij} = P_{ji}$. This adjustment ensures that high dimensional outliers have a well determined location in the low dimensional embedding. Why? Using the traditional matrix construction in equation 1, the outlier x_i will have large pairwise distances $||x_i - x_j||^2$, leading to uniformly small $p_{i|j}$'s for all x_j. This means the location of point i in the low dimensional embedding has little effect on the SNE cost function. Why? Recall that KL is minimized when two distributions look similar; thus, samples with very low likelihoods, like outliers, have little effect on the shape of the distribution, and consequently, little effect on the KL score. In t-SNE, $p_{ij} = (p_{i|j} + p_{j|i})/2N$ leads to a lower bound on the sum of joint probabilities for outlier x_i: $\sum_i p_{ij} > \frac{1}{2n}$. Since $\sum_j \sum_i p_{ij} = 1$, by definition of a probability distribution, this lower bound implies that each data point contributes meaningfully to the joint distribution P, and therefore the cost function.

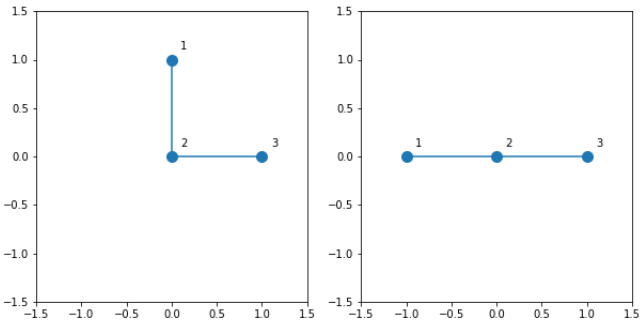
As for the Q matrix, t-SNE instead assumes the euclidean distance between two points is generated by a student t distribution with one degree of freedom. Thus, the pdf is given by:

$$P(x = ||y_i - y_j||) = c(1 + x^2)^{-1}$$

where c is a constant. Similar to SNE, we assume q_{ij} is the probability y_i picks y_j as its neighbor (or vice versa), proportional to the pair's t-distribution density:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}} \tag{6}$$

Note that this reformulation makes Q symmetric, and like our new P, we can treat Q as containing likelihoods from a joint distribution. The main rationale behind using the t distribution is that it addresses the crowding problem faced by SNE. Consider the following toy example:



In the left most photo, three points make up the vertices of a right triangle, pairs (1,2) and (2,3) are closer together than (1,3). Suppose we wanted to find 1D representation of these points that preserves the distances between the points. As shown in the second figure, if we maintained the distance between the

closer pairs, then the distance between (1,3) would significantly increase. Here's the source of the crowding problem: there's simply less area in lower dimensions to faithfully maintain all high dimensional pairwise distances. SNE reacts to the crowding problem by collapsing moderately distance points closer together. Why? SNE aims for $p_{i|j}$ to be as close as possible to $q_{i|j}$, so in the toy example above, $q_{1|3}$ would become too small. Even though SNE prioritizes preserving similarities between closer points, SNE could still decrease the distance between (1,2) and (2,3) to ensure the distance between (2,3) doesn't blow up. In other words, SNE would crowd the data points together.

t-SNE, on the other hand, would tolerate this initial low dimensional representation! Why? The t distribution has a similar shape to the gaussian but with heavier tails. This implies that points that are moderately far apart in the high dimensional space can be faithfully modelled further apart in the low dimensional setting. More explicitly, if a euclidean distance of 5 maps to a p_{ij} equal to 0.2, a euclidean distance of 6 could map to a q_{ij} also equal to 0.2.

2.3 t-SNE Optimization

Given that we interpret P and Q as containing likelihoods from two joint distributions, we can reformulate the cost function as evaluating the KL between two joint distributions:

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

To clarify, reframing P and Q as joint distributions instead of a bunch of conditional distributions is in part a way to build some intuition that connects the contents of P and Q to the use of the KL distance metric. Indeed, when comparing equation 7 and 3, you'll notice we do the same number of computations using the same indices of P and Q. However, the reformulation of P and Q do make evaluating a single gradient speedier, as we'll see.

So to find the low dimensional embedding, Y, t-SNE minimizes C over each y_i using a gradient descent method with step updates similar to the SNE update in equation 4, except without the noise term, $\epsilon(t)$, and with a different gradient.

$$Y^{(t)} = Y^{(t-1)} + \eta \frac{\delta C}{\delta Y} + \alpha(t)(Y^{(t-1)} - Y^{(t-2)}) \tag{8}$$

where the gradient is:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \tag{9}$$

See the full gradient derivation in the appendix 10. When we compare t-SNE's gradient to SNE's (equation 5), we notice that t-SNE's gradient doesn't involve an expensive exponential evaluation, making t-SNE's gradient quicker to compute. In addition, we gain another persepective on why t-SNE addresses SNE's crowding problem. In both optimizations, we seek to push the gradient to 0.

Thus, within the sum of the SNE gradient, we generally want the $(y_i - y_j)$ term to be as close to zero as possible, while ensuring that we faithfully conserve similarities between the high and low dimensional probability distributions by pushing $(p_{i|j} - q_{i|j} + p_{j|i} - q_{j|i})$ to 0. So if $x_i - x_j$ is large, the gradient will generally attempt to push y_i and y_j closer to together, if it can maintain a small value for $(p_{i|j} - q_{i|j} + p_{j|i} - q_{j|i})$. In the t-SNE gradient, however, larger distances between y_i and y_j can also help minimize the t-SNE gradient because of the $(1 + ||y_i - y_j||^2)^{-1}$ term.

t-SNE further improves upon the general performance of the optimization algorithm with two additional tricks. By performance, I mean increasing the probability of finding the global minimum of the cost function holding step size constant. First, t-SNE uses "early compression" which pushes low dimensional points to stay closer together at the start of the optimization. When the points are closer together, it's easier for them to move amongst each other, allowing points to shift between clusters of data and allowing for clusters to arrange themselves differently. To implement this, t-SNE adds a penalty term to the cost function:

$$C = KL(P||Q) + d(t) \sum_{i,j} ||y_i - y_j||^2$$

where d(t) tuning parameter that decreases to 0, after the first few iterations. In other words, we penalize low dimensional embeddings where the sum of the squared distances is large in the first few iterations.

t-SNE also implements an "early exaggeration" parameter, $e(t)$ which multiplies all the values of the p_{ij}'s by a constant, like 4, in the initial stages of the optimization. So while all the q_{ij}'s add up to 1, they are too small to match the corresponding p_{ij}'s as the KL minimization seeks to do. This compels the optimization to prioritize modelling points with large p_{ij}'s, or the immediately clear clusters in the data. Consequently, early exaggeration forces the optimization to construct a low dimensional representation of the immediately obvious clusters in the data, so during later steps, the optimization works on less clearly clustered data. The sklearn python package implements t-sne with a default of 1000 iterations, where the first 250 steps add early exaggeration and compression (Pendagrosa 2011). These tricks make the t-SNE optimization more stable than SNE's, making it unnecessary to run t-SNE several times with the same hyperparameters (van der Maaten and Hinton, 2008).

To put everything together, the algorithm becomes:

Algorithm 1: t-SNE Algorithm

Objective Function Hyperparameter: perplexity, *perp*;
Optimization Hyperparameters: $e(t)$, early exaggeration; $d(t)$, early compression; η, step size; $\alpha(t)$, momentum.;
Compute pairwise affinities $p_{j|i}$ using equation 1 with perplexity $Perp$ specified in equation 2 ;
Set $p_{ij} = e(t)\frac{p_{i|j}+p_{j|i}}{2n}$, where $e = 4$ for $t < 250$ and 0 otherwise;
Sample initial solution $Y^0 = y_1, y_2, \ldots y_n.$ from $N(0, 10^{-4}I)$;
for *t=1 to T* **do**
    Compute low-dimensional similarities q_{ij}, using equation 6;
    Compute the gradient $\frac{\delta C}{\delta Y}$ using equaiton 9;
    Set $Y(t)$ with equation 8;
end

Notice that I distinguish between the "Objective Function Hyperparameter", perplexity, which affects the objective function, and a slew of "Optimization Hyperparameters", which affect the nature of the optimization routine.

While all these optimization parameters seem quite ad hock, setting the early exaggeration parameter e roughly equal to $\frac{N}{10}$ and step size η roughly equal to 1 actually leads to exponential convergence (over the number of iterations) to a low dimensional representation with geometrically distinct clusters, provided that they exist and as long as the the number of points is roughly less than 20,000 (G. Linderman and S. Steinerberger, 2017). Further this paper claims t-SNE doesn't even need the momentum term $a(t)$ for fast convergence.

It's also worth noting that related papers have proposed alternative optimization techniques to minimize t-SNE's cost function, related to the dual tree algorithm and sampling via random walks (van der Maaten, 2014; van der Maaten and Hinton, 2008).

3 Discussion and Application

In this section, I will compare t-SNE's strengths and weaknesses relative to other leading dimension reduction techniques using numerical examples and theory. I will specifically consider PCA, ISOMAP, Sammon Mapping, and LLE.

3.1 MNIST Example

As a first pass to compare the performance of t-SNE relative to PCA, ISOMAP, Sammon Mapping, and LLE, I plot the 2D embeddings produced by each algorithm, using 5000 random samples from the MNIST hand drawn numbers dataset (Y. Lecun and C. Corets, 2010). See the Appendix in section 6.2. The MNIST data set represents each image as a 784 dimensional vector, where the value at an index represents the darkness of a pixel in the original 28 by 28 pixeled image. Before passing the 5000 x 784 raw data into each algorithm, I

pass the data through PCA to reduce the dimensionality to 30 features. The plot also reports the runtime for each algorithm.

Two important clarification points: 1) these are all unsupervised algorithms, which is to say, the data labels aren't passed into the algorithm, though I label the data points to evaluate the visualizations; 2) these are all feature based dimension reduction algorithms, not clustering algorithms– i.e. all these algorithms attempt to reduce the number of features, not find and label clusters in the data.

One way to evaluate these low dimensional visualizations is to consider the extent to which these algorithms preserve the labelled clusters. As an example, if we assume the 1s and 3s are far apart in high dimensional space, but mixed in the 2D representation, then we're witnessing the crowding problem! As the graphs suggest, t-SNE creates the most distinct low dimensional clusters. LLE is a lost cause, PCA is able to seperate the 1s and 0s but the rest of the digits are quite mixed. ISOMAP and Sammon Mapping are able to separate only a few digits as well. At least in this example, t-SNE is about 10x slower than ISOMAP, 100X slower than LLE, and 10000x slower than PCA. Sammon mapping was 10x slower t-SNE.

Obviously we cannot conclude t-SNE's superiority after only considering one data set and one set of hyper parameters for each algorithm. Indeed, t-SNE's performance can vary significantly when we change its one hyperparameter, perplexity, as the second figure in the appendix suggests (6.2). There could also exist a set of hyperparameters for LLE or ISOMAP that yields better visualizations.

Instead of relying on examples, we can make stronger statements around an algorithm's ability to produce better visualizations by comparing the theoretical properties of each algorithm.

3.2 Theoretical Strengths of t-SNE relative to other algorithms

In this subsection, I discuss the theoretical underpinnings behind t-SNE's ability to avoid the crowding problem, relative to other dimension reduction techniques. Like in Section 2, assume each method seeks to construct a low dimensional data matrix Y from high dimensional data matrix X.

Given that PCA accomplishes an identical dimension reduction to Classical Scaling, we can treat PCA as an optimization problem that returns a low dimensional embedding by finding a linear transformation on X that seeks to preserve all pairwise distances between each point (Mardia et al., 1979). i.e.:

$$\underset{Y}{\operatorname{argmin}}\, C = \sum_{i \neq j} (||x_i - x_j|| - ||y_i - y_j||)^2$$

PCA fails to address the crowding problem because it doesn't explicitly prioritize preserving distances between close points (neighbor structure) or relaxing distances between farther points (neighborhood structure) as it weighs all pairs

the same. And more generally, if data live on a high dimensional manifold, PCA fails because it assumes all pairwise distances are euclidean.

Sammon mapping attempts to directly address PCA's shortcomings by dividing the squared error by the high dimensional euclidean distance in the objective function. I.e.

$$\underset{Y}{\operatorname{argmin}} C = const * \sum_{i \neq j} \frac{(||x_i - x_j|| - ||y_i - y_j||)^2}{||x_i - x_j||}$$

This tweak allows the Sammon Mapping to prioritize keeping closer points together in low dimensional space, as farther pairs will have a larger denominator in their cost function term, and thus less influence on cost. This allows the Sammon mapping to preserve high dimensional local structure. While this weighting relaxes the need to retain the distance of farther apart observations, it doesn't explicitly address the crowding problem. Far away pairs, or whole neighborhoods, can either move further apart in the low dimensional embedding (good) or collapse onto each other (bad).

t-SNE tends to outperform ISOMAP and LLE because of its ability to reduce high dimensional data living on two or more widely seperated manifolds. ISOMAP and LLE rely upon building a strongly connected graph across all high dimensional points where the edges represents euclidean distance between points. As I discuss in my review of Stochastic Neighborhood Embedding (subsection 2.1), SNE and t-SNE model the high dimensional data in a manner similar to fitting many local gaussians, which can deal with widely seperated manifolds.

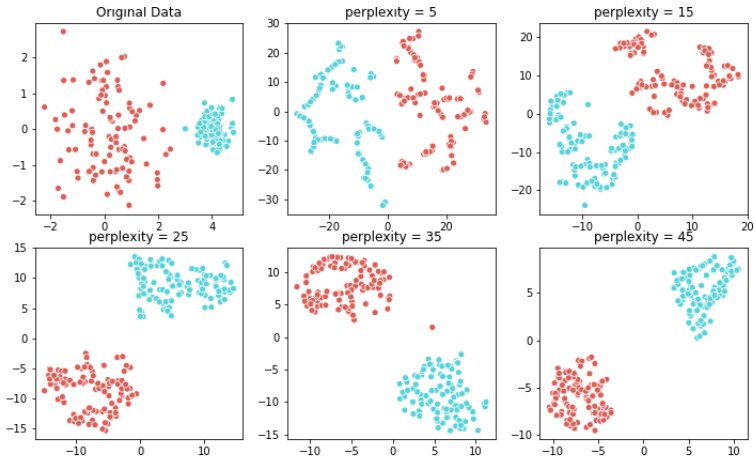
3.3 Limits of t-SNE

While I have described why t-SNE produces better vizualizations relative to other algorithms, t-SNE has some notable weaknesses

3.3.1 Interpretability and General Use

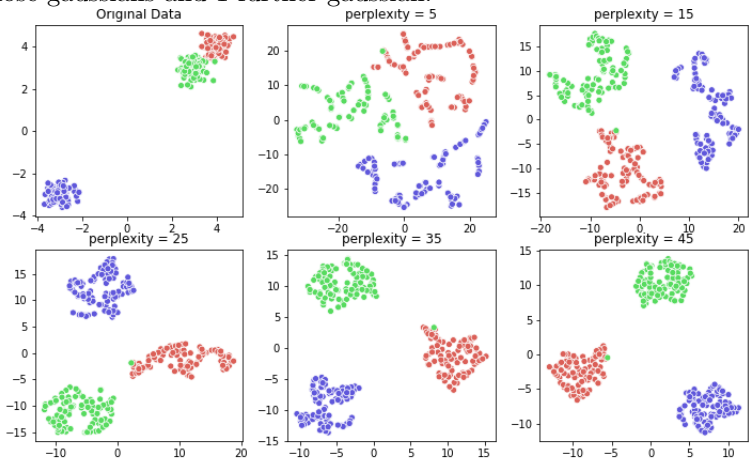
When the practitioner reduces the dimensionality of a dataset, regardless of the reason, they often seek to retain some specific and interpretable property of the high dimensional data. PCA and Classical MDS, for example, preserve pairwise euclidean distances; LLE and ISOMAP seek to preserve geodesic distances among points. With t-SNE, however, we seek to preserve some vague notion of local and global structure by minimizing the KL divergence of two joint distributions that required some gymnastics to intuitively connect to our raw data. In other words, a t-SNE dimension reduction is less interpretable. To further illustrate the limits of interpreting t-SNE mappings, I present two toy examples inspired by Wattenberg et al. 2016.

Consider the following example where the raw data is generated by two gaussian clusters of different density:



Clearly one cannot draw information from the relative sizes of the low dimensional clusters in a t-SNE plot, as t-SNE tends to balance densities across clusters in its low dimensional representation. This balancing behavior is a byproduct of using the t-distribution which spreads close points further apart. Thus, one can't make the following argument: "point A and B are closer in cluster 1 than points C and D and cluster two; therefore, A and B are closer to each other than C and D in the raw data."

Further, a t-sne visualization can't reliably show the distances between clusters either. Consider the following t-SNE map created from raw data generated by 2 close gaussians and 1 further gaussian.



As this example shows, t-SNE tends to equalize the distances between clusters. While there are cases where t-SNE vizualizations with higher perplexity can reveal global structure (like distances between clusters), larger data sets make it harder for t-sne to vizualize relative distances (Wattenberg et al. 2016).

Unless the practitioner is getting consistent visualisations across perplexities, it would be very challenging to rely on t-SNE to draw conclusions around relative

distances between clusters. As a more hopeful example, consider the MNIST t-SNE plot across multiple perplexities (appendix section 6.2). Regardless of the perplexities, the same clusters tended to neighbor each other! The loopy numbers (0, 5 and 6) tended to hang out near each other, and the more angular numbers (7 and 4) were always adjacent. So, given that similar adjacencies were maintained across several perplexities, it seems fine to argue that this t-sne picked up distances across clusters. Furthermore, as Hinton and van der Maaten (2008) show in their numerical examples, similar looking numbers are closer together within a cluster, validating that t-SNE preserves some local structure.

This example also shows that at lower perplexities, t-SNE may display too many small clusters. Suppose the data were unlabeled and you were presented with the graph produced by t-SNE with perplexity 5, how many clusters would you conclude were in the data? Too many. At larger perplexities, however, we are able to discern the actual number of clusters in the true data.

t-SNE also has some ability to retain high dimensional cluster shape, if those shapes can be represented in 2 or 3 dimensions. Again, we can only trust t-SNE's results if the algorithm returns consistent visualizations across perplexities.

3.4 Curse of Dimensionality

Like other nonlinear dimension reduction algorithms, t-SNE relies upon high dimensional neighbors to live on a linear plane. If this doesn't occur, t-SNE can't faithfully preserve the similarity between neighbors in the low dimensional embedding. I describe this problem in detail in section 2.1.2.

3.5 Non-Convex Optimization

Another weakness of t-SNE is the non-convexity of its cost function. Consequently, recovering the low dimensional embedding requires numerical methods that hinge on certain optimization hyper parameters. Further, because we randomly initialize Y, running t-SNE with the same hyperparameters can yield different results, though the original authors of the algorithm contend that the results don't vary significantly due to different initializations of Y. However, varying the optimization parameters can significantly affect the results, and can prevent convergence (G. Linderman and S. Steinerberger, 2017). By contrast, PCA, ISOMAP, and LLE all have convex objective functions, and thus, no optimization hyperparameters and a potentially speedier optimization routine. Sammon Mappings also require minimizing a non-convex optimization function.

4 Conclusion: Using t-SNE in the Wild

Given all these strengths and weaknesses, t-SNE should be primarily used for data exploration and model evaluation. Here are two different use cases:

- Suppose one has a high dimensional dataset without labelled clusters. A good first step to understanding the structure of the data would be to run t-SNE on it. After considering how many distinct neighborhoods t-SNE claims your data has, one can run a clustering algorithm on the data to test t-SNE's results and glean further within cluster structure.

- t-SNE can also be used to evaluate dimension reduction techniques. As an example, suppose you have designed algorithm that attempts to extract features from an image data set in R^{D*20} onto R^D where $D > 3$. As a concrete example, you designed an algorithm that clusters human portraits by facial expression. To evaluate this algorithm, map the presumably clustered data in R^D into R^2 with t-SNE, and detect if the images within clusters in R^2 retain similar facial expressions.

While running t-SNE, here are few pointers to consider(Van Der Maaten 2020):

- Implement t-SNE with the popular packages in R or Python, start with the default optimization routine and hyper parameters.

- If the data dimension is greater than 30, reduce its dimensionality down to 30 or 40 with PCA before applying t-SNE on the data to improve runtime.

- Run t-sne using a variety of perplexities, typically between 5 and 45. Lower perplexity preserves local structure while higher perplexity preserves global structure. In general, larger or denser datasets require a larger perplexity to create an interpretable vizualization.

- Run t-SNE with a maximum number of iterations of at least 1000. If the gradient in the last step isn't close to zero, consider increasing the number of iterations.

- It's worth noting that t-SNE can be applied to data sets that consist of pairwise similarites between observations, rather than the high dimensional vector representations of each object, only if the pairwise similarities encode probabilities that add up to 1. As an example, human word association data consist of the probability of producing a word from a corpus of words, in response to a given word (Van Der Maaten and Hinton 2008).

5 Bibliography

G.E. Hinton and S.T. Roweis. Stochastic Neighbor Embedding. In Advances in Neural Information Processing Systems, volume 15, pages 833–840, Cambridge, MA, USA, The MIT Press. 2002.

LeCun, Y. & Cortes, C. MNIST handwritten digit database. 2010.

G. C. Linderman, & S. Steinerberger. Clustering with t-SNE, provably. ArXiv:1706.02582 [Cs, Stat]. 2017. http://arxiv.org/abs/1706.02582

K.V. Mardia, J.T. Kent, and J.M. Bibby. Multivariate Analysis. Academic Press, 1979

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

Schubert, E., & Gertz, M. Intrinsic t-Stochastic Neighbor Embedding for Visualization and Outlier Detection. In C. Beecks, F. Borutta, P. Kröger, & T. Seidl (Eds.), Similarity Search and Applications (pp. 188–203). Springer International Publishing. 2017 https://doi.org/10.1007/978-3-319-68474-1_13

L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605, 2008

L.J.P van der Maaten. t-SNE FAQ. 2020.
https://lvdmaaten.github.io/tsne/faq

Wattenberg, et al., "How to Use t-SNE Effectively", Distill, 2016.
http://doi.org/10.23915/distill.00002

6 Appendix

6.1 Derivation of t-SNE gradient $\frac{\delta C}{\delta y}$

Recall that t-SNE attempts to minimize the following cost function

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

Where P has values:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

where

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\Sigma_{k \neq l} exp(-||x_k - x_l||^2/2\sigma_i^2)}$$

and Q has values

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

To reduce clutter, define two auxillary variables:

$$d_{ij} = ||y_i - y_j||$$

$$Z = \sum_{k \neq l} (1 + d_{kl}^2)^{-1}$$

allowing us to define:

$$q_{ij} = \frac{(1 - d_{ij})^{-1}}{Z}$$

If y_i changes, the only pairwise distances that change are $d_i j$ and $d_j i$ for all j, so by the chain rule:

$$\frac{\delta C}{y_i} = \sum_j (\frac{\delta C}{\delta d_{ij}} + \frac{\delta C}{\delta d_{ji}})(y_i - y_j)$$

$$\frac{\delta C}{y_i} = 2 \sum_j \frac{\delta C}{\delta d_{ij}} (y_i - y_j) \tag{10}$$

By the definition of Kullback-Leibler divergence in equation 7:

$$\frac{\delta C}{\delta d_{ij}} = -\sum_{k \neq l} p_{kl} \frac{\delta (log\, q_{kl})}{\delta d_{ij}} = -\sum_{k \neq l} p_{kl} \frac{\delta (log\, q_{kl} Z - log Z)}{\delta d_{ij}}$$

With more algebra:

$$\frac{\delta C}{\delta d_{ij}} = -\sum_{k \neq l} p_{kl} (\frac{1}{q_{kl} Z} \frac{\delta (1 + d_{kl}^2)^{-1}}{\delta d_{ij}} - \frac{1}{Z} \frac{\delta Z}{\delta d_{ij}})$$

The gradient $\frac{\delta (1 + d_{kl}^2)^{-1}}{\delta d_{ij}}$ is nonzero only when k = i and l = j, so:

$$\frac{\delta C}{\delta d_{ij}} = 2 \frac{p_{ij}}{q_{ij} Z} (1 + d_{kl}^2)^{-2} - \sum_{k \neq l} p_{kl} \frac{(1 + d_{kl}^2)^{-2})}{Z}$$

Given that $\sum_{k \neq l} p_{kl} = 1$, we can simplify to:

$$\frac{\delta C}{\delta d_{ij}} = 2 p_{ij} (1 + d_{kl}^2)^{-1} - 2 q_{ij} (1 + d_{kl}^2)^{-1}) = 2 (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1}$$

Substituting this expression into equation 10 from above yields the desired gradient:

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

6.2 MNIST Figures

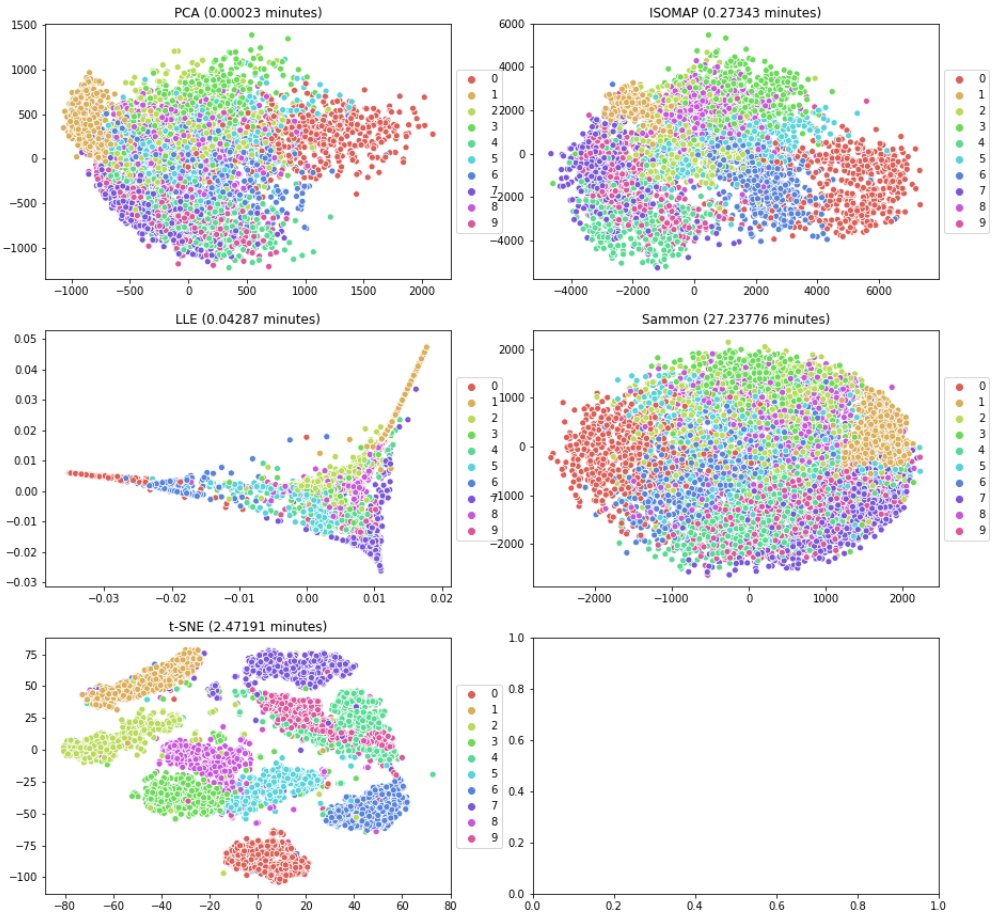


Figure 1: Visualises the 2D embedding of 5000 samples from MNIST data set created by a variety of algorithms, as well as the runtime required to produce each embedding. The t-SNE was created with a perplexity of 5, on 2000 iterations.

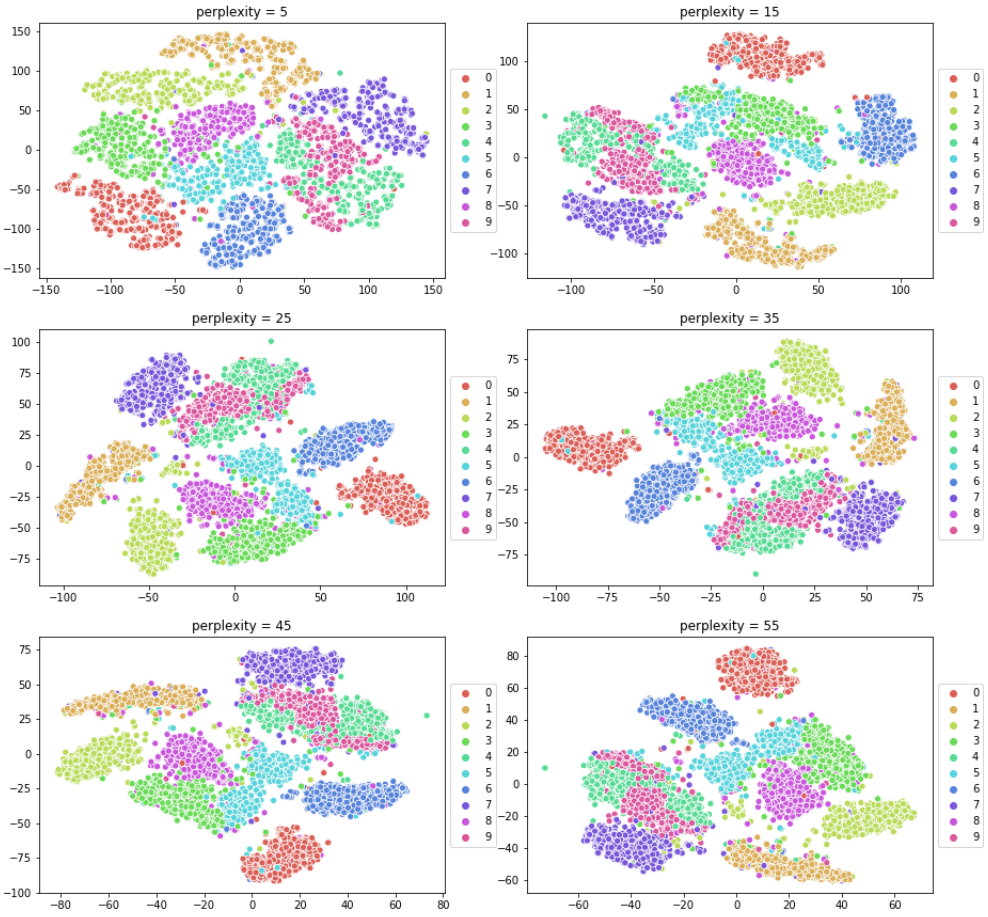


Figure 2: Visualizes the 2D embedding of 5000 samples from the MNIST data set created by t-SNE using 6 different values of perplexity and 2000