T-SNE Explained — Math and Intuition

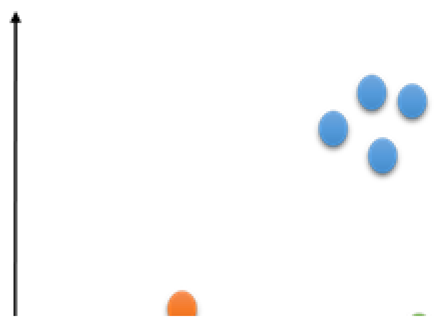
Achinoam Soroker Follow
Aug 4, 2020 · 7 min read ★

The method of t-distributed Stochastic Neighbor Embedding (t-SNE) is a method for dimensionality reduction, used mainly for visualization of data in 2D and 3D maps. This method can find non-linear connections in the data and therefore it is highly popular. In this post, I'll give an intuitive explanation for how t-SNE works and then describe the math behind it.

See your data in a lower dimension

So when and why would you want to visualize your data in a low dimension? When working on data with more than 2–3 features you might want to check if your data has clusters in it. This information can help you understand your data and, if needed, choose the number of clusters for clustering models such as k-means.

Now let's look at a short example that will help understand what we want to get. Let's say we have data in a 2D space and we want to reduce its dimension into 1D. Here's an example of data in 2D:



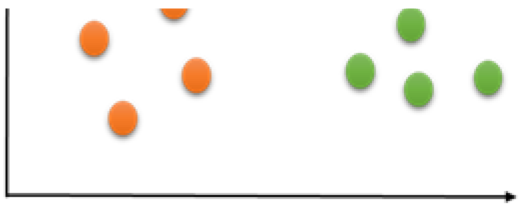


Figure 1: Original Data points in the high-dimensional space

In this example, each color represents a cluster. We can see that each cluster has a different density. We will see how the model deals with that in the dimensional reduction process.

Now, if we try to simply project the data onto just one of its dimensions, we see an overlap of at least two of the clusters:

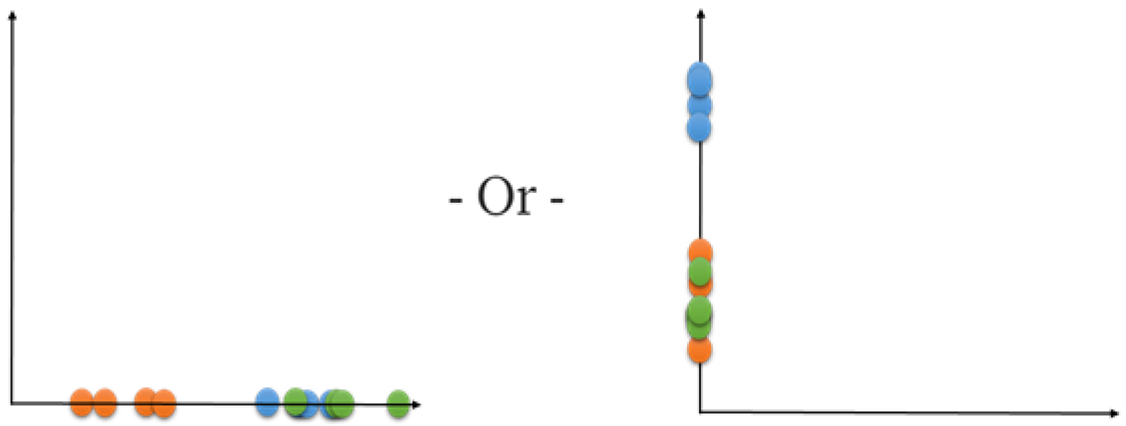


Figure 2: Data projections to one dimension

So we understand that we need to find a better way to do this dimension reduction.

T-SNE algorithm deals with this problem, and I'll explain its performance in three stages:

- Calculating a joint probability distribution that represents the similarities between the data points (don't worry, I'll explain that soon!).
- Creating a dataset of points in the target dimension and then calculating the joint probability distribution for them as well.
- Using gradient descent to change the dataset in the low-dimensional space so that the joint probability distribution representing it would be as similar as possible to

the one in the high dimension.

The Algorithm

First Stage — Dear points, how likely are you to be my neighbors?

The first stage of the algorithm is calculating the Euclidian distances of each point from all of the other points. Then, taking these distances and transforming them into conditional probabilities that represent the similarity between every two points. What does that mean? It means that we want to evaluate how similar every two points in the data are, or in other words, **how likely they are to be neighbors.**

The conditional probability of point x_j to be next to point x_i is represented by a Gaussian centered at x_i with a standard deviation of σ_i (I'll mention later on what influences σ_i). It is written mathematically in the following way:

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

The probability of point x_i to have x_j as it's neighbor

The reason for dividing by the sum of all the other points placed at the Gaussian centered at x_i is that we may need to deal with clusters with different densities. To explain that, let's go back to the example of Figure 1. As you can see the density of the orange cluster is lower than the density of the blue cluster. Therefore, if we calculate the similarities of each two points by a Gaussian only, we will see lower similarities between the orange points compared to the blue ones. In our final output we won't mind that some clusters had different densities, we will just want to see them as clusters, and therefore we do this normalization.

From the conditional distributions created we calculate the joint probability distribution, using the following equation:

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

Using the joint probability distribution rather than the conditional probability is one of the improvements in the method of t-SNE relative to the former SNE. The symmetric property of the pairwise similarities ($p_{ij} = p_{ji}$) helps simplify the calculation at the third stage of the algorithm.

Second Stage — Creating data in a low dimension

In this stage, we create a dataset of points in a low-dimensional space and calculate a joint probability distribution for them as well.

To do that, we build a random dataset of points with the same number of points as we had in the original dataset, and K features, where K is our target dimension. Usually, K will be 2 or 3 if we want to use the dimension reduction for visualization. If we go back to our example, at this stage the algorithm builds a random dataset of points in 1D:

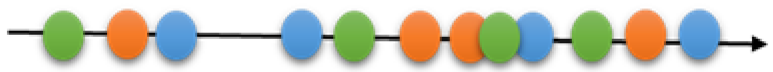


Figure 3: A random set of points in 1D

For this set of points, we will create their joint probability distribution but this time we will be using the t-distribution and not the Gaussian distribution, as we did for the original dataset. This is another advantage of t-SNE compared to the former SNE (t in t-SNE stands for t-distribution) that I will soon explain. We will mark the probabilities here by q, and the points by y.

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

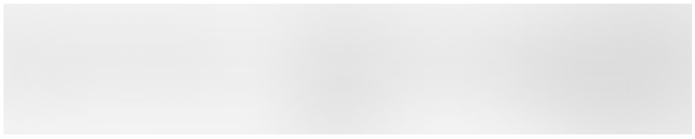
The reason for choosing t-distribution rather than the Gaussian distribution is the heavy tails property of the t-distribution. This quality causes moderate distances between points in the high-dimensional space to become extreme in the low-dimensional space, and that helps prevent "crowding" of the points in the lower dimension. Another advantage of using t-distribution is an improvement in the optimization process in the third part of the algorithm.

Third Stage — Let the magic happen!

Or in other words, change your dataset in the low-dimensional space so it will best visualize your data

Now we use the Kullback-Leiber divergence to make the joint probability distribution of the data points in the low dimension as similar as possible to the one from the original dataset. If this transformation succeeds we will get a good dimension reduction.

I'll briefly explain what Kullback-Leiber divergence (KL divergence) is. KL divergence is a measure of how much two distributions are different from one another. For distributions P and Q in the probability space of χ, the KL divergence is defined by:



The definition of KL divergence between the probability distributions P and Q

As much as the distributions are similar to each other, the value of the KL divergence is smaller, reaching zero when the distributions are identical.

Back to our algorithm — we try to change the lower dimension dataset such that its joint probability distribution will be as similar as possible to the one from the original data. This is done by using gradient descent. The cost function that the gradient descent tries to minimize is the KL divergence of the joint probability distribution P from the high-dimensional space and Q from the low-dimensional space.

The cost function for the gradient descent is the KL divergence between P and Q, the joint probability distributions of the high and low dimensions respectively

From this optimization, we get the values of the points in the low dimension dataset and use it for our visualization. In our example, we see the clusters in the low-dimensional space as follows:

Parameters in the model

There are several parameters in this model that you can adjust to your needs. Some of them relate to the process of gradient descent, where the most important ones are the learning rate and the number of iterations. If you are not familiar with gradient descent I recommend going through its explanation for better understanding.

Another parameter in t-SNE is **perplexity**. It is used for choosing the standard deviation σ_i of the Gaussian representing the conditional distribution in the high-dimensional space. I will not elaborate on the math behind it, but it can be interpreted as the number of effective neighbors for each point. The model is rather robust for perplexities between 5 to 50, but you can see some examples of how changes in perplexity affect t-SNE results in the following article.

Conclusion

That's it! I hope this post helped you better understand the operating algorithm behind t-SNE and will help you use it effectively. For more details on the math of the method, I recommend looking at the original paper of TSNE. Thank you for reading :)

Data Science Algorithms Math