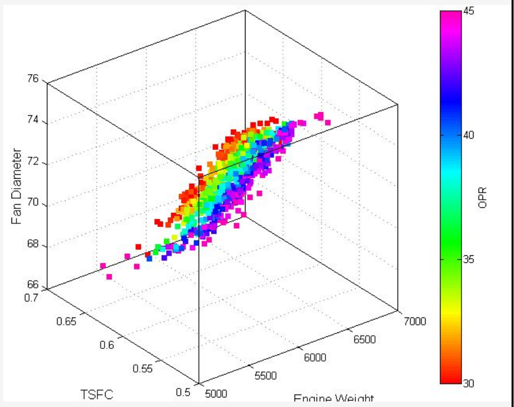
t-Distributed Stochastic Neighbor Embedding (t-SNE)

•••

Jimmy DeLano

Introduction & Motivating Examples

- Problem: predicting the prices of homes



https://stats.stackexchange.com/questions/123767/what-are-some-popular-choices-for-visualizing-4-dimensional-data

Let's start by considering this general problem: you're trying to predict the price of a home. You have found some sort of dataset containing lots of information that might be useful for predicting the price of the home:

- Square footage, # bathrooms, # bedrooms, Age, lot size, is it on a golf course
- These are all different metrics that we might find in a dataset about prices of homes.

These data can be represented in a data matrix X that has the shape n * p – n for the number of rows and p for the number of columns or features, as I'll refer to them from now on.
Each home, or data point, $\mathbf{x_1} = (x_{11}, \ldots x_{1p})T$ is a vector of length p.

Before we get into modeling the price of the house, we want to try and visualize the data.

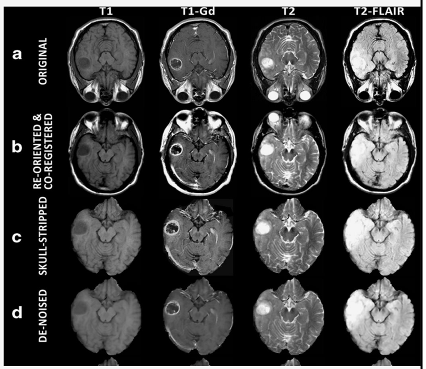
- Question is: How do we make a graph of this data? How do we visualize it to understand which features are most important for distinguishing expensive homes from cheap homes?

As humans, we can only visualize the relationship between 2 or 3 features at a time. We can't really understand plots in 4D or higher.

In our world of big data, there are lots of problems where this would come up – not just predicting home prices.

Introduction & Motivating Examples

- Medical imaging for cancer diagnoses
- Facial expression recognition
- Text comparison for semantic similarity



https://crimetechweekly.com/2017/07/20/big-data-investigation-analytics-a-difference-maker-for-law-enforcement/

https://www.nature.com/articles/sdata2017117

But what happens when the n and p increase by a lot.

Consider the problem of trying to analyze brain scans for cancer diagnoses. I can show you a picture here of 16 brain images but what if we have 10000 points? Each datapoint, each image, could be made up of hundreds of pixels. (p goes way up).

- How do we approach visualizing that data? Can we do so where we group cancerous brains together vs benign.

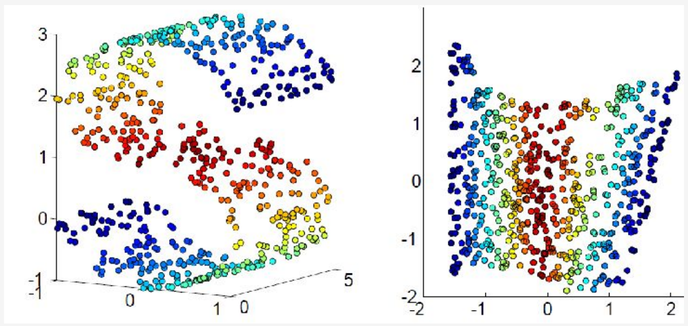
If we were looking at thousands of tweets and trying to do some sort of text analysis on the semantic similarity, we could try to visualize the data using a word cloud. This is one way to represent all the data but we lose a lot of information.

In both cases, the limiting factor seems to be the amount of data that we're trying to compress down into one plot.

A simple idea to solve this problem is to…

Solution: Dimensionality Reduction

- Build a *map* where distances between points reflect similarities in the data:

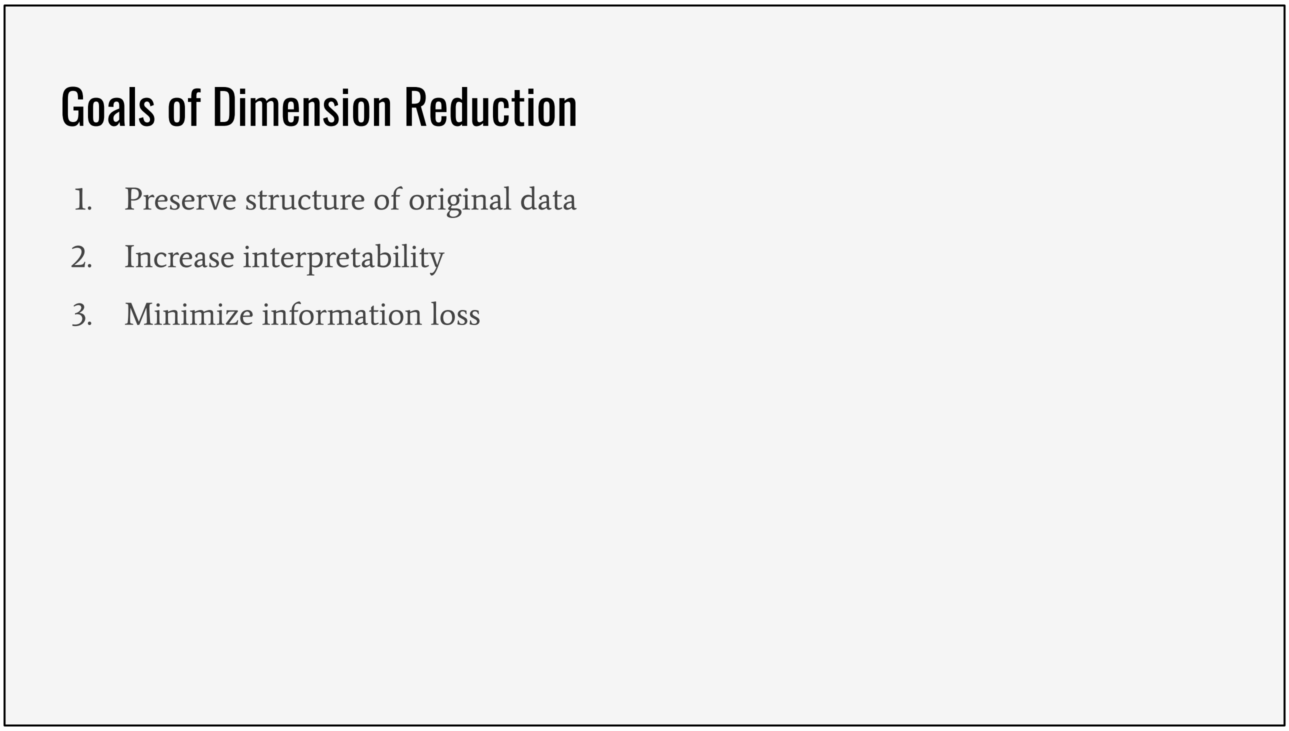


https://www.semanticscholar.org/paper/A-Survey-of-Dimension-Reduction-Methods-for-Data-Engel-H%C3%BCttenberger/8dc7a7af1685d6667d24f013ecc5fceeb2bcc689

Build a map, in which each point – as it exists in the data in high dimensional space – will be represented by a different point in lower dimensional space in such a way that that similar datapoints are represented by nearby points and dissimilar datapoints will be represented by far away points.

This is dimensionality reduction aka embedding. It's a technique for representing multi-dimensional data in 2 or 3 dimensions.
Hopefully this is where we'll get insight into the data in the lower dimensional space.

When we think about dimension reduction, we want to keep a few goals in mind. The main goals are



Goals of Dimension Reduction

1. Preserve structure of original data
2. Increase interpretability
3. Minimize information loss

1. Homes example: Similar homes should have similar representations in the map
2. Cancer image example: Cluster malignant vs benign tumors while visualizing thousands of images at once
3. Twitter example: don't want to lose the semantic structure

At its core, reducing the number of features naturally reduces the accuracy of the data, but the tradeoff is for simplicity: we are able to understand/interpret it better in lower dimensions.

These goals are very general and they apply to all applications using dimension reduction

They will also apply to one more example which I'll introduce now which will be used throughout the talk…

MNIST Dataset

- Popular machine learning database with 60,000 images
- Each $\mathbf{x}_i$ is a 28x28 grayscale image

https://towardsdatascience.com/handwritten-digit-recognition-with-mnist-on-ios-with-keras-e85e194f9fa5

Modified National Institute of Standards and Technology dataset = MNIST. Collected from high school students and people who worked at the US census.

It's a collection of 60k handwritten digits.
Each image is 28*28 pixel with one of 10 classes representing integer values from 0 to 9, inclusively. Each x_i is 1 image with 784 features, one for each pixel.

Widely considered to be "solved" by many different algorithms – achieving 99%+ accuracy, but it's a popular starting point for developing and testing new methodologies.

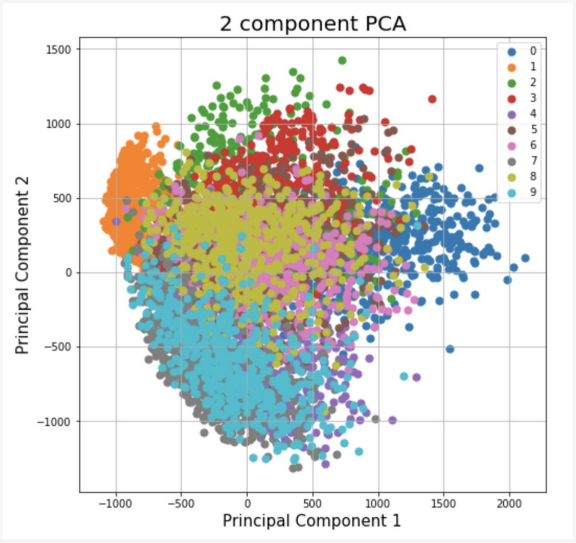
For our case here, we want to think about how to visualize all this data (60k x 784) in one plot.

We have 70 images here but how do we visualize thousands of images?

We're going to explore this question throughout my talk…

Outline

Principal Component Analysis (PCA)



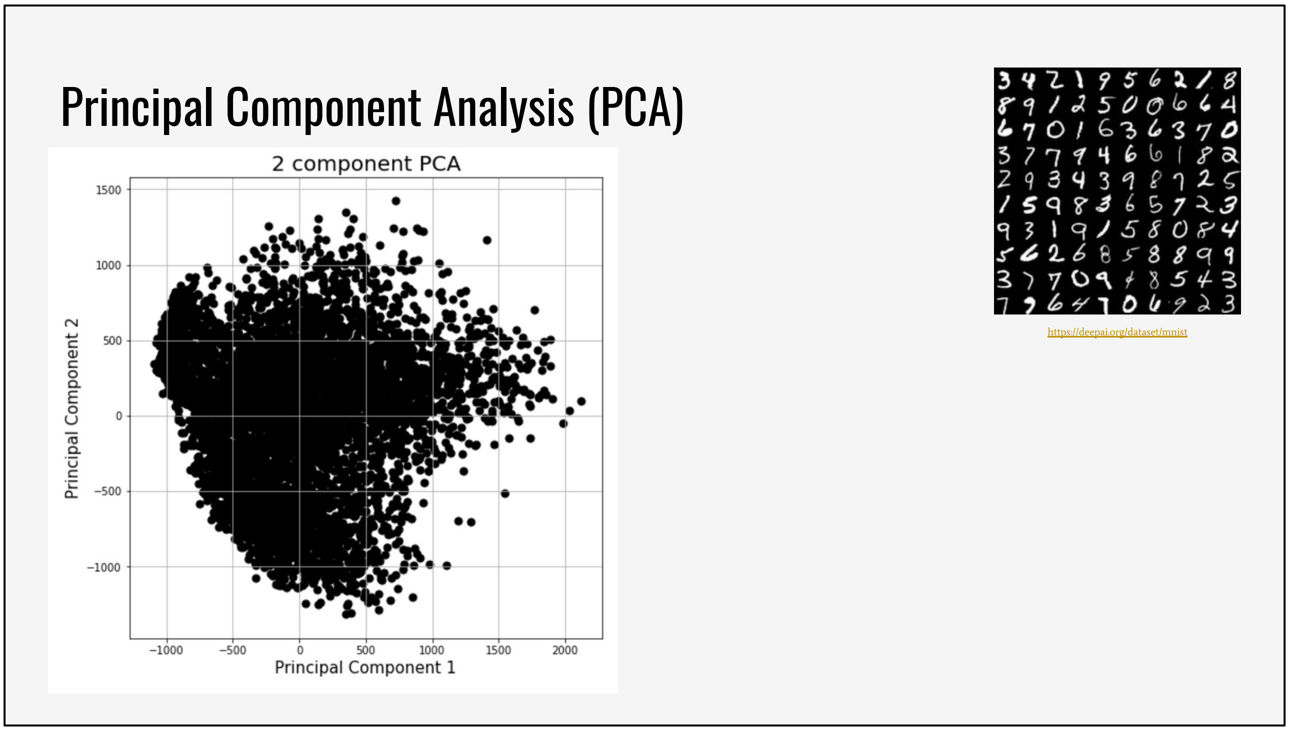
https://deepai.org/dataset/mnist

If you're in stat 442 with me you have heard about PCA because that's what we've talked about the last two classes. But for those who do not know,

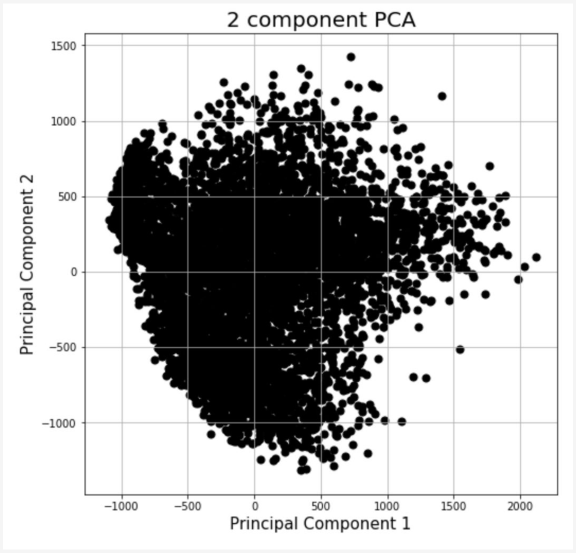
PCA is a dimensionality reduction technique. The output of the algorithm are PCs: new variables that are created as a linear combination of the existing features.
The 1st PC aims to capture/explain the maximum amount of variance in the data.
The 2nd PC does the same but in a direction that's uncorrelated with the 1st PC.
Geometrically speaking, a principle component represents the directions of the data that explain the maximum amount of variance. And there's a lot of linear algebra going on under the hood but I'll explain what's going on here geometrically…

These values along the axis don't mean anything to us humans, but we get an idea of what's happening visually:
1st PC: 0's and 1's are furthest apart. Makes sense when you think about them in terms of pixels – probably the most distinct digits.
2nd PC: 9's, 7's, 4's are at bottom. 2's, 3's 5's are at the top. Second source of variation.

This looks alright, but mainly because I added colors to the plot. If you were in the unlabeled case…



Principal Component Analysis (PCA)



https://deepai.org/dataset/mnist

We’d see something like this. Almost impossible to make out any clusters here

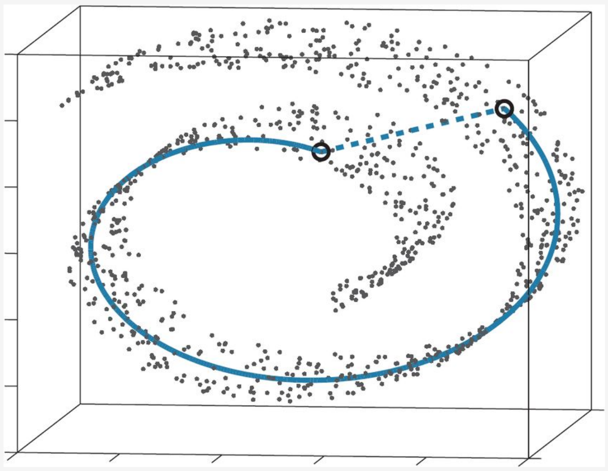
Maybe on the left but that’s a stretch.

Question is can we do better? Spoiler the answer is yes.

What Distances to Preserve?

PCA is mainly concerned with preserving *large* pairwise distances in the map

Are these distances reliable?



https://www.researchgate.net/figure/The-swiss-roll-dataset-The-distance-between-two-arbitrary-points-in-the-original_fig8_329075265

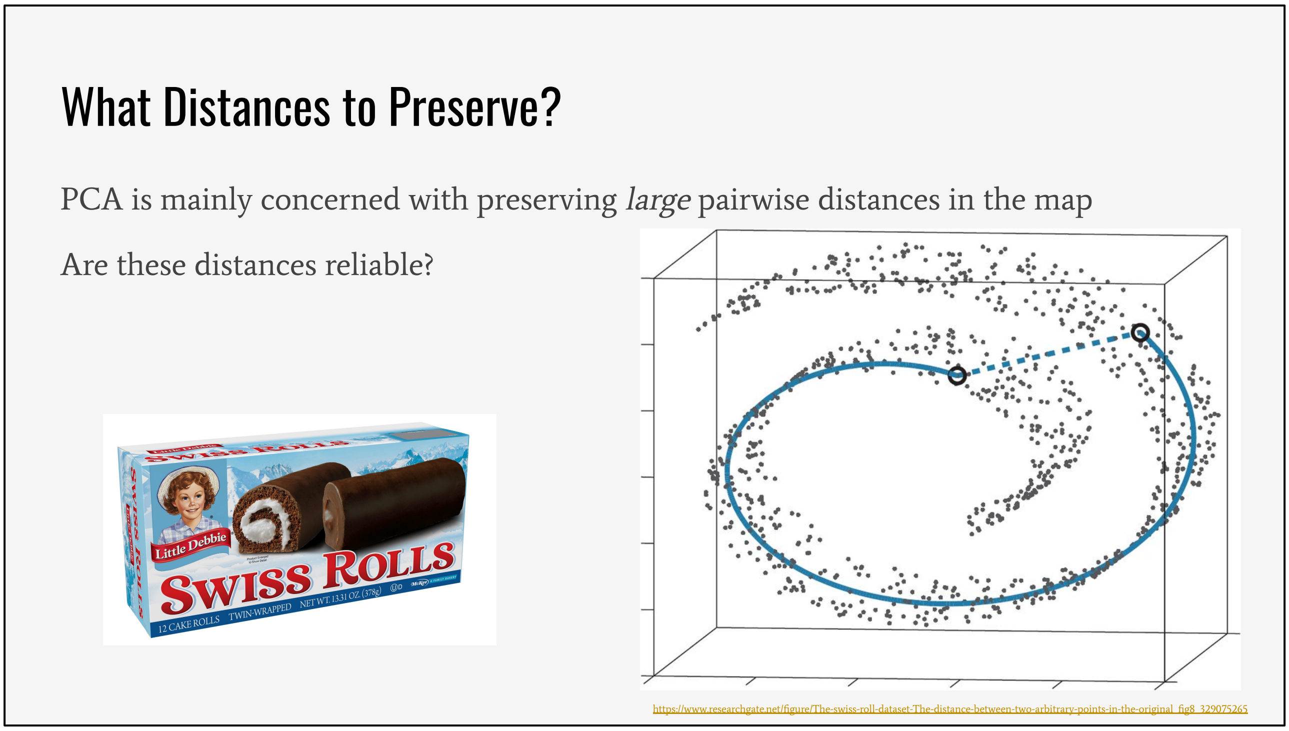
Question was is PCA minimizing the right objective function in this case?

PCA maximizes the variance which is the same as minimizing the squared error between distances in the original data.

Because you're looking at squared error, you're mainly concerned with preserving distances that are very large – meaning that stuff that's dissimilar should be very far apart (just like we saw with the 0's and the 1's) right.

And the question is: do we really want that in our visualization? Maaten and Hinton, the two people who came up with t-SNE answered no.

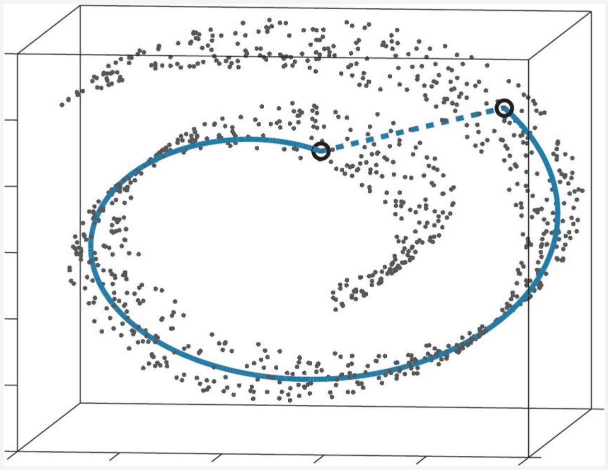
And their idea centers around an example like this:



What Distances to Preserve?

PCA is mainly concerned with preserving *large* pairwise distances in the map

Are these distances reliable?



https://www.researchgate.net/figure/The-swiss-roll-dataset-The-distance-between-two-arbitrary-points-in-the-original_fig8_329075265

when the original data is some non-linear manifold (meaning shape), here the swiss roll.

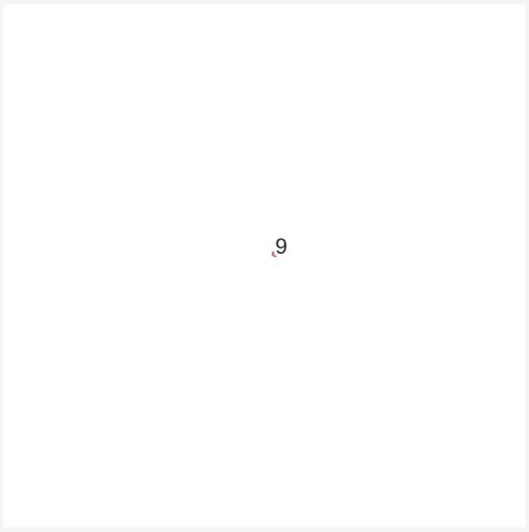
The euclidean distance between these two points in high dimensional space is relatively small. However, when we consider the entire structure of the manifold, these two points should be very far apart.

The key idea is that PCA doesn't work well for visualization always because it's mainly concerned with preserving large pairwise distances in the map and those are not always reliable. It's also doing a linear transformation which won't work for this example – it will require some non-linear projection.

What is reliable, even on this manifold, are very small pairwise distances – and this serves as the motivation for several other techniques for dimensionality reduction and visualization.

Outline

t-Distributed Stochastic Neighbor Embedding (t-SNE)



We’re going to talk about t-SNE and how it works.

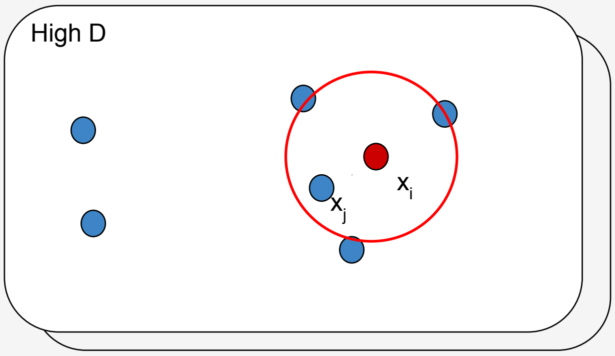
t-SNE is another dimensionality reduction technique but it differs from PCA in a few keys ways.
It is randomized and non-linear.

And rather than outputting PCs, the output of the algorithm is this map – a set of points clustered by the similarities between points.

I’ve included a gif here of the optimization phase for MNIST – let’s get into how it works …

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Measure pairwise similarities between points in high dimension



We're going to start in the high dimensions w/ the original data.

Take point xi and center a gaussian over this point. Then we're going to measure the density of all the other points under this gaussian.
This essentially gives us a set of probabilities p i j *write on board*
** define euclidean distance. **

So what does this mean? We're measuring the similarity between points xi and xj normalized by the sum of the distances between other points in high dimensional space.

You can think of this as a probability distribution over pairs of points where the probability of picking a particular point as its nearest neighbor is proportional to the similarity.

If 2 points are close together (similar) in high dimensional space, then P i j will be relatively large. [if we have 2 pics of same number, this will be relatively high]
If 2 points are dissimilar, P i j will be infinitesimal.
[if we have a 0 and a 1, we expect p ij to be almost 0]

I actually lied here. In practice …

t-Distributed Stochastic Neighbor Embedding (t-SNE)

In practice, we compute the similarities slightly differently:

$$p_{j|i} = \frac{\exp(-\parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2 / 2\sigma_i^2)}{\sum_{j' = i} \exp(-\parallel \mathbf{x}_i - \mathbf{x}_{j'} \parallel^2 / 2\sigma_i^2)}$$

σ_i is a hyperparameter that represents the *perplexity* for x_i

In practice, we actually going to compute the conditional distribution – the only thing that's changed here is the denominator. We don't normalize over all pairs of points, just over the pairs of points around x i

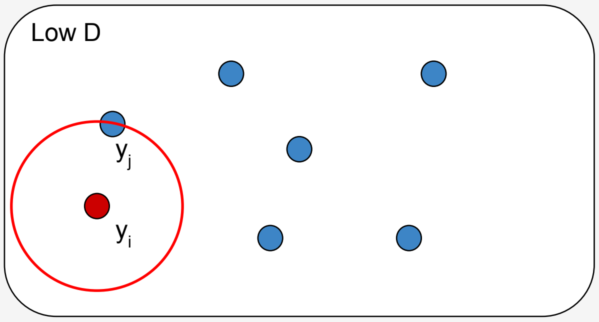
Sigma is a hyperparameter representing perplexity. We can think of this as scaling the size/bandwidth of the gaussian around x i such that a set number of points fall in the Gaussian. The reason for this is that different parts of the space may have different densities.

Also, because the distribution around x i is different than the distribution around x j, we average their p i js – where N is the number of dimensions.

Now we have our conditional probabilities for the high dimensional data. We're going to transition to talking about measuring similarities in the low dimension…

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Measure pairwise similarities in low-dimensional map points:



We're going to look at the low dimensional space – will be 2 or 3 dimensions. Lay out points in a map

Represent each data point in high dimensional space as a new point in the lower dimension.

In the low dimension, we define q i j *draw*

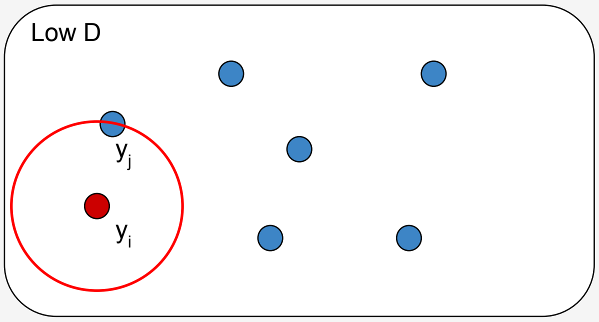
If we consider the red point, y i, we're basically going to do the same thing here. We're going to center a kernel around each point and measure the density of all the other points, y j, under that distribution. And again, we're going to re-normalize the distance dividing by the sum of distances between pairs of other points.

Thus, we get the probability, q i j, which gives us the similarity between two points in the low dimensional map.

We want these points, q i j, to reflect the similarities, p i j, that we computed in the high dimensional space as well as possible. The way we're going to measure the difference between these p i j and q i j is by using Kullback Leibler divergence (KL).

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Move points around to minimize cost function

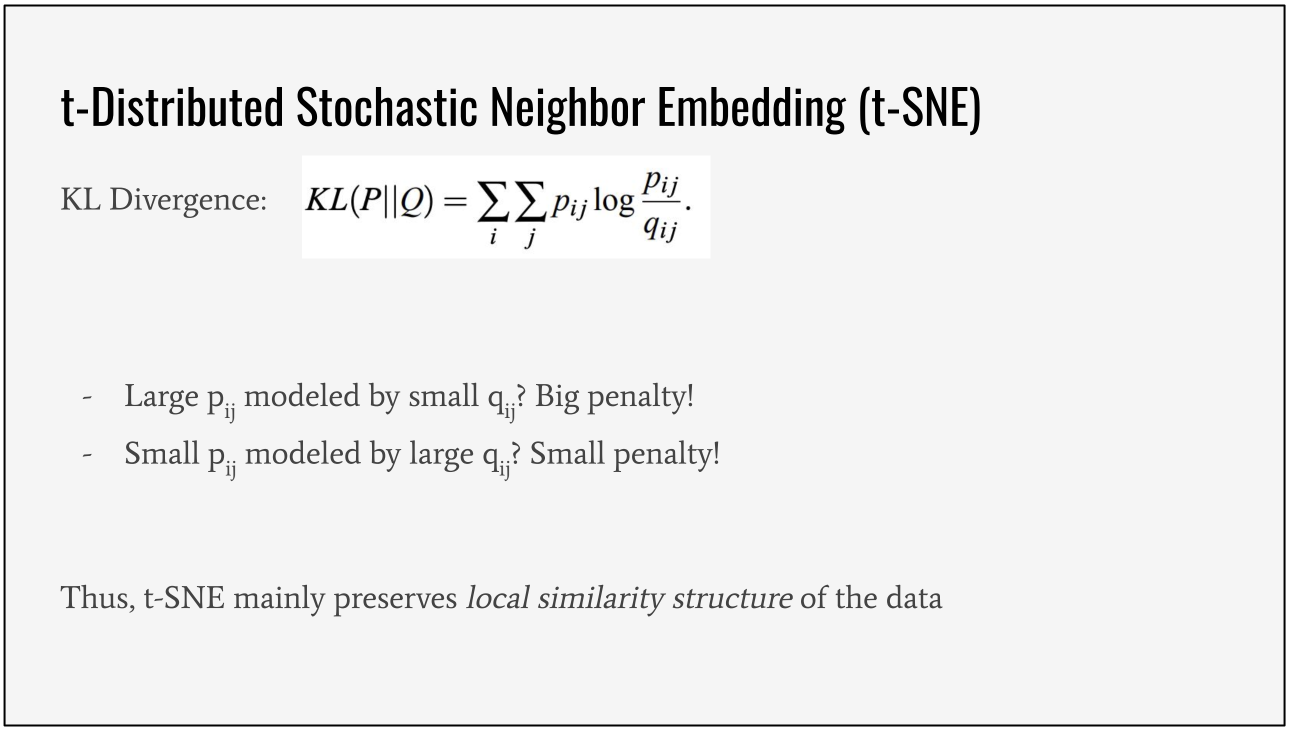


At a high level, KL divergence is essentially a way to measure the difference between probability distributions. It takes on the form *draw*

So it’s a sum, over all pairs points, with p i j multiplied by log(). You could also think of it as the expected value of the log difference between probabilities p ij and q ij .

Okay so we want to move around these points in the low dimensional space such that the KL divergence is minimized. The smaller the KL divergence the better our map is because p ij is very close to qij.

In order to do that, we use gradient descent. Before we get into that, I want to talk about why we use KL divergence….



t-Distributed Stochastic Neighbor Embedding (t-SNE)

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

- Large p_{ij} modeled by small q_{ij}? Big penalty!
- Small p_{ij} modeled by large q_{ij}? Small penalty!

Thus, t-SNE mainly preserves *local similarity structure* of the data

The reason we use KL divergence is because it has a nice asymmetry property.

Consider 2, similar points in high dimensional space (2 images of a 7). These will have a large p ij value. If you have a large p ij value then you better make sure these two points also get a large q ij value. Otherwise, you'll incur a large penalty. Right. Large p ij times log large value divided by small value is going to get big.

KL really tries to model large p ijs (similar points in high dimensions) with large q ijs (similar points in the map).

*It doesn't work the other way around though.

Consider 2 points that are dissimilar in high dimensional space (0 and a 1). You'll have a small p ij, then you don't really care about what the q ij value is.

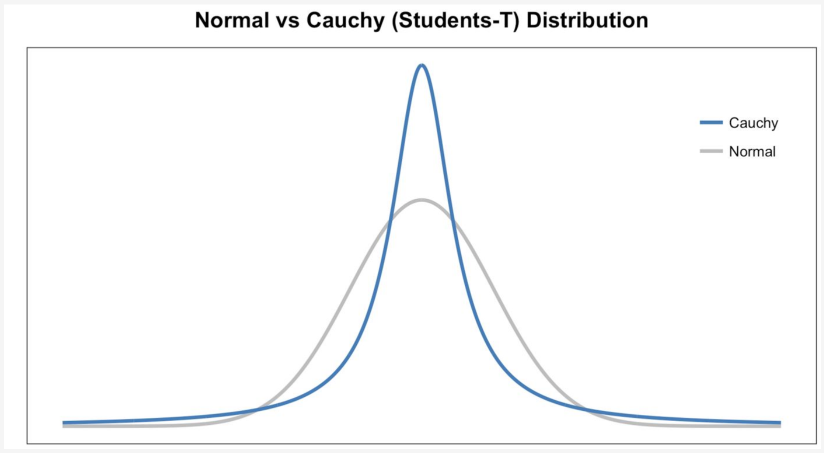
We're really only concerned with mapping the local structure of the data correctly.

Now the one thing that I skipped over earlier – if you noticed – was that when computing q ij, we did not use a Gaussian kernel

Outline

Why a Student t-Distribution?

What about the dissimilar points?



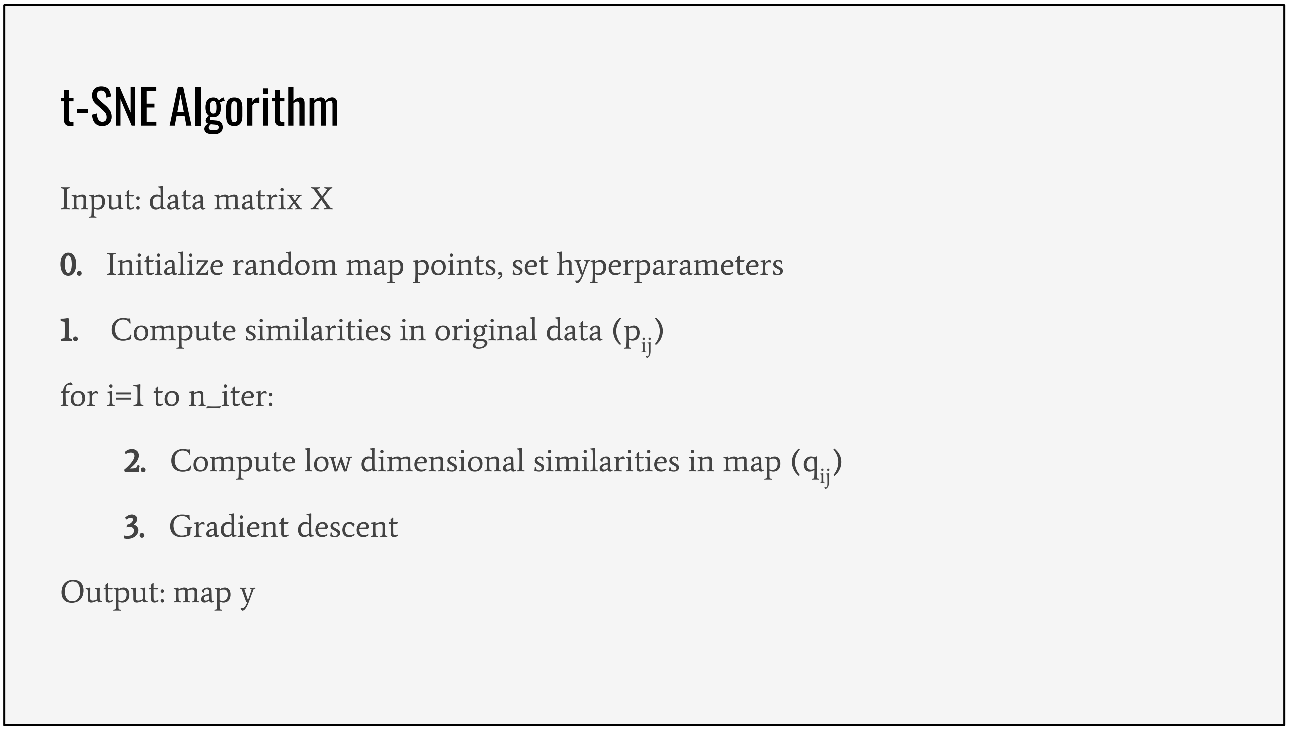
https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1

reference equation this e ^ -½ times squared euclidean distance between the points or whatever.

Instead, we used a student t-distribution with 1 degrees of freedom – also known as the Cauchy distribution. And the question is *why* do we define map similarities with this?

The reason is that the Gaussian distribution can cause crowding of data points in the map whereas the t is more flexible.

** as we can see from the pic, tails for t has more space for points that are far apart. Consider 2 points far away from x i. Under gaussian, these would be mapped to the same spot. Under the t, we have more space so these points don't overlap. This is the key intuition for t-SNE which improved on regular SNE that had problems with crowding data **



t-SNE Algorithm

```
Input: data matrix X
0. Initialize random map points, set hyperparameters
1. Compute similarities in original data ($p_{ij}$)
for i=1 to n_iter:
    2. Compute low dimensional similarities in map ($q_{ij}$)
    3. Gradient descent
Output: map y
```

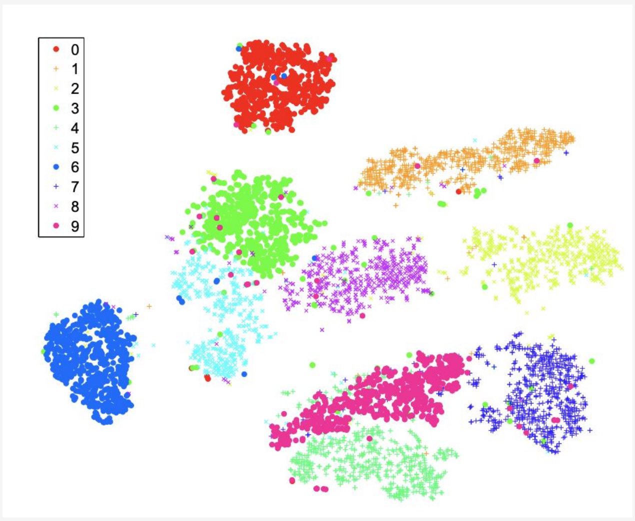
Summary of algorithm here

Vastly oversimplified the gradient descent aspect here. At a high level, this step involves calculating the derivative of our cost function C (KL) and using that as part of equation to update our map points, moving them in the direction of the steepest descent which is that derivative. (B/c goal is minimizing the KL divergence).

Each iteration, we move slightly closer to the minimum.

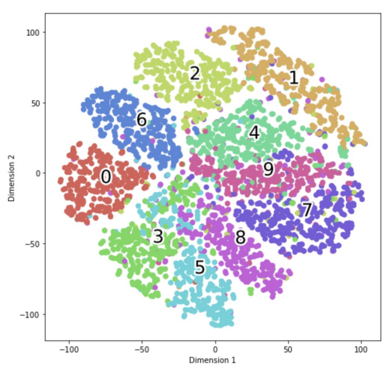
Outline

t-SNE on MNIST

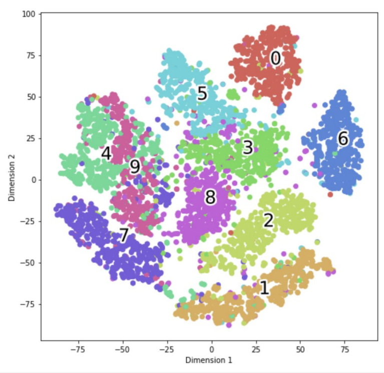


van der Maaten & Hinton (2008)

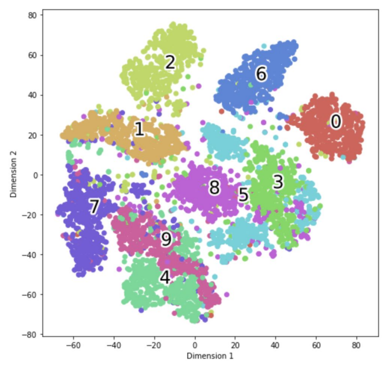
t-SNE on MNIST



perplexity = 5

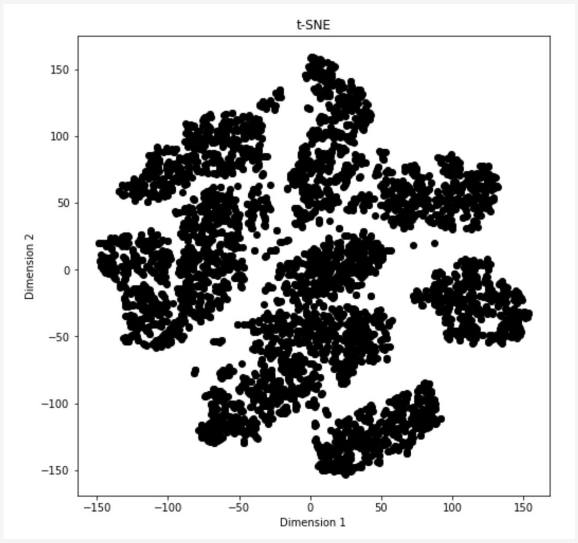


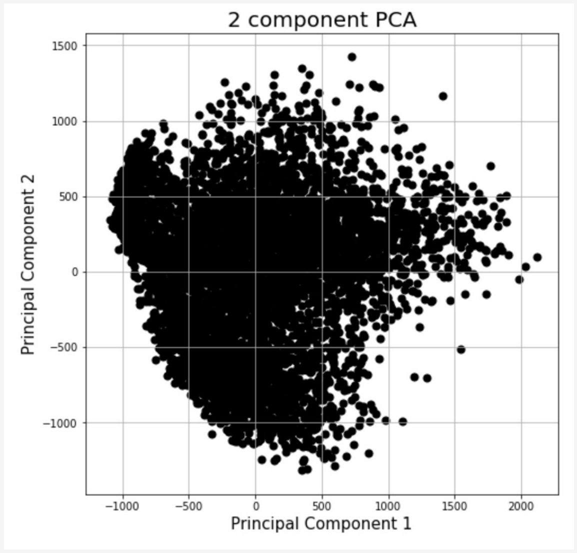
perplexity = 15



perplexity = 30

t-SNE vs PCA on MNIST





Black and white photos for t-SNE and PCA

Issues with t-SNE or interactive tool. Fun structures. Or other examples for ending.

References

https://jmlr.csail.mit.edu/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.441.8882&rep=rep1&type=pdf

https://www.youtube.com/watch?v=RJVL80Gg3lA&ab_channel=GoogleTechTalks

https://towardsdatascience.com/t-distributed-stochastic-neighbor-embedding-t-sne-bb60ff109561

https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a

https://www.oreilly.com/content/an-illustrated-introduction-to-the-t-sne-algorithm/

https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

Appendix

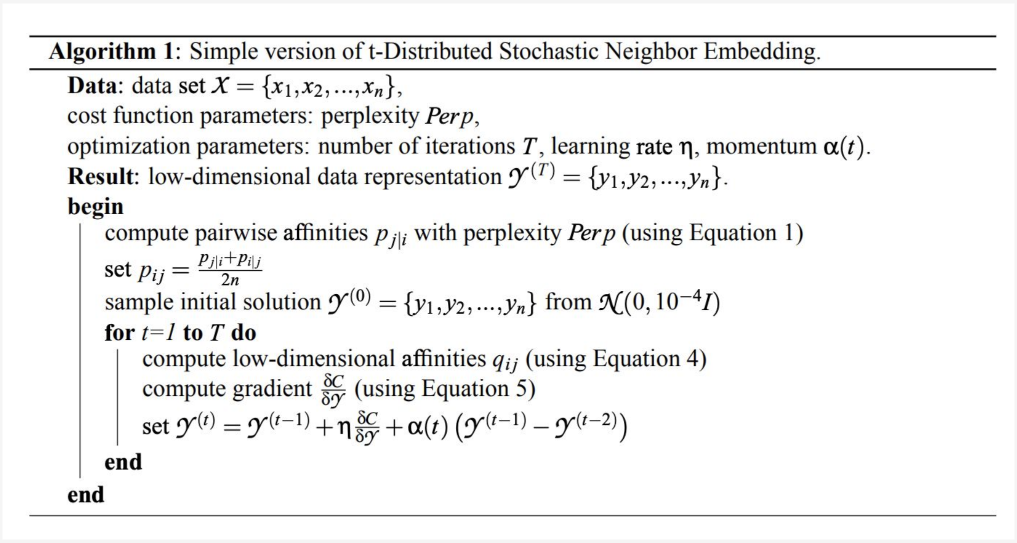
Gradient Descent

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

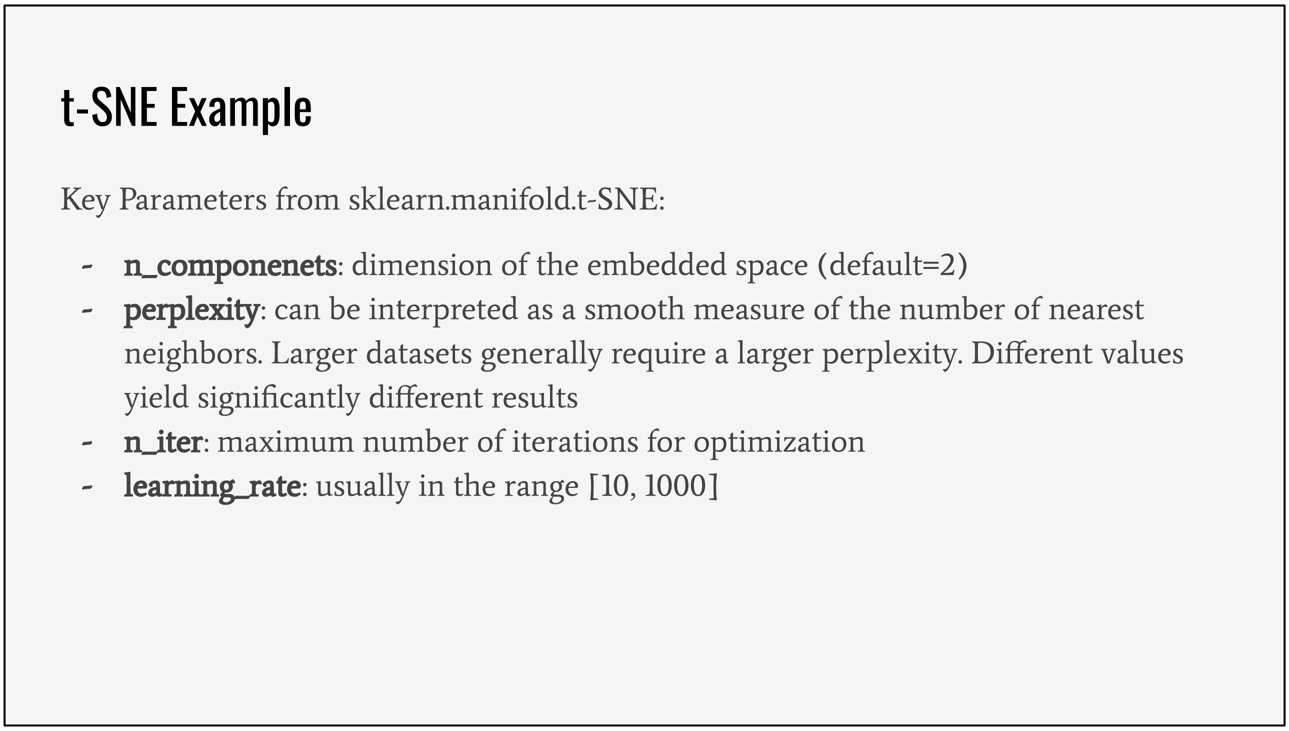
$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$$

Gradient descent gif

Algorithm Pseudocode



```
Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.
  Data: data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$,
  cost function parameters: perplexity $Perp$,
  optimization parameters: number of iterations $T$, learning rate $\eta$, momentum $\alpha(t)$.
  Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$.
  begin
      compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
      set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
      sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4} I)$
      for $t$=1 to $T$ do
          compute low-dimensional affinities $q_{ij}$ (using Equation 4)
          compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
          set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
      end
  end
```



t-SNE Example

Key Parameters from sklearn.manifold.t-SNE:

- **n_componenets**: dimension of the embedded space (default=2)
- **perplexity**: can be interpreted as a smooth measure of the number of nearest neighbors. Larger datasets generally require a larger perplexity. Different values yield significantly different results
- **n_iter**: maximum number of iterations for optimization
- **learning_rate**: usually in the range [10, 1000]

N_components – i.e, the lower dimension that we want the high dimensional data to be converted into

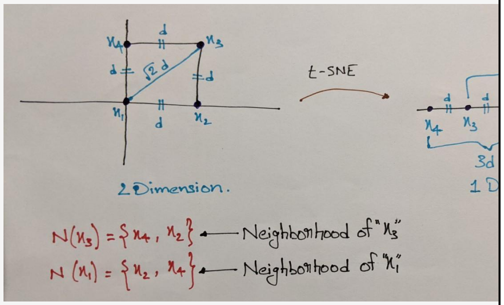
Perplexity:
Think of a manifold structure as any geometric shape like: cylinder, ball, curve, etc.

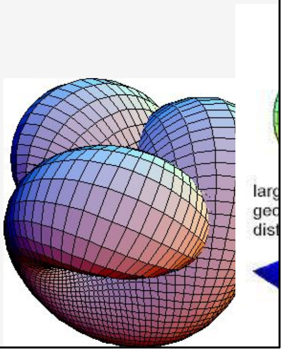
Challenges with SNE Algorithm

2 key issues issues that motivate t-SNE

- Cost function is difficult to optimize in practice

- The “crowding problem”
  - Draw 2D example


x_4
x_3
d
$\sqrt{2}d$
x_1
x_2
t-SNE
3d
2 Dimension.
1 D
$N(x_3) = \{x_4, x_2\}$ ← Neighborhood of "x_3"
$N(x_1) = \{x_2, x_4\}$ ← Neighborhood of "x_1"



- Cost function optimization is the process of finding the optimal values for the model parameters by finding the global minima of cost functions.
- Involves simulated annealing which I don’t know much about other than it’a a probabilistic technique for approximating the global optimum of a given function.

Not going to get into the weeds too much here but the main idea is that, in practice, one has to run the optimization several times on a data set to find appropriate values for the parameters

—-
Other problem …
NOT specific to SNE, but that it also occurs in other local techniques for multidimensional scaling such as Sammon mapping.

t-SNE doesn't guarantee to resolve the crowding problem all the time, but it gives its best to overcome the crowding problem and preserving the neighborhood distance

Maybe one other example of t-SNE finding correct structures

https://distill.pub/2016/misread-t-SNE/