UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction VS T-SNE

Abdelkarim HAFID
karimhafid97@gmail.com

Abstract

In my project, I will first show how the performance of T-SNE has become unsustainable to efficiently manage the Dimensionality reduction. Then, I will establish a comparison between UMAP and T-SNE in terms of data visualization quality in low-dimension Embedding , to see the advantages of the UMAP algorithm to accelerate the convergence towards an efficient data visualization in low-dimensional Embedding space then I will develop one python code at the base of the UMAP algorithm with some intuitions outsourced from the T-SNE algorithm. I will therefore prove that the visualization of data in low-dimension Embedding can be improved comparing to the original UMAP implementation by merging the UMAP and T-SNE algorithm.

1 Why T-SNE is dead and steal not an efficient tool to do Dimensionnality reduction

- T-SNE does not keep the overall structure of the data: meaning that only within cluster distances are meaningful while between cluster similarities are not guaranteed . it is therefore widely recognized that clustering on t-SNE is not a good idea: I will prove this important point in the third section in this report
- T-SNE cannot work directly with high-dimensional data, We need first an Autoencoder or the PCA method to perform a pre-dimension reduction before using T-SNE
- T-SNE consumes a lot of memory to do its calculations and need a considerable time to converge to the optimal cost
- TSNE can practically only be incorporated in 2 or 3 dimensions, only for visualization purposes, so it is difficult to use T-SNE as a general dimension reduction technique in order to produce 10 or 50 components.

2 The main differences between the UMAP and T-SNE algorithms which affect the speed of convergence towards an efficient visualization of data in low-dimensional Embedding

A) UMAP uses an exponential probability distribution in high dimensions like the first picture below show . When we increase the number of dimensions in the original data set we introduce directly more sparsity on the data. So, we get more and more fragmented manifold. sometimes there are dense regions sometimes there are isolated points.The UMAP algorithm solves this problem by introducing the local connectivity parameter which glues together the sparse regions via introducing the local data connectivity

$$p_{i|j} = e^{-\frac{d(x_i, x_j) - \rho_i}{\sigma_i}}$$

The T-SNE uses Euclidean distances and Normalisation to either high- or low-dimensional probabilities. This small normalisation step that UMAP avoid in his algorithm had actually a dramatic effect on the performance. This is because summation or integration is a computational expensive step.

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

B) UMAP uses the number of nearest neighbors instead of the perplexity used in tSNE like this :

$$\text{Perplexity} = 2^{-\sum_j p_{j|i}\log_2 p_{j|i}}$$

UMAP defines the number of nearest neighbor k without the log2 function , practically putting a log-prefactor in front of the linear term does not add much since log-function is slower than the linear function:

$$k = 2^{\sum_i p_{ij}}$$

C) UMAP uses a different symmetrization of the high-dimensional probability to make the weight similar in the two directions between each two nodes in the graph that represent our data. But I found that this symmetrization is not efficient because it had a minor effect on the final visualation on the low-dimentionnal embedding, so I tried to use my own symmetrization to have better visualisation of the data in low-dimension embedding .as you can remarque in the .ipynb file joined to this report.

$$p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}p_{j|i}$$

D) UMAP uses one family of curves that is similar to Student t-distribution to model distance probabilities in low dimensions without any normalisation like the T-SNE algorithm did :

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

for the original UMAP algorithm they use a1.93 and b0.79 in the picture below but for me I want to merge the idea of the T-SNE algorithm with the UMAP one by choosing a=1 and b=1. You find this update in the python code attached to this report :

$$q_{ij} = \left(1 + a(y_i - y_j)^{2b}\right)^{-1}$$

E) UMAP uses this cross-entropy (CE) as a cost function:

$$CE(X,Y) = \sum_i\sum_j\left[p_{ij}(X)\log\left(\frac{p_{ij}(X)}{q_{ij}(Y)}\right) + (1 - p_{ij}(X))\log\left(\frac{1 - p_{ij}(X)}{1 - q_{ij}(Y)}\right)\right]$$

Instead of the Kullback-Leibler divergence like the tSNE cost function:

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

So the differrence between The UMAP and the T-SNE is that we have one added second term in the CE of UMAP. This second term makes UMAP capable of capturing the global data structure in contrast to T-SNE that can only model the local structure as I will prove in the third section of this Report. F)Since I need to know the gradient of the cross-entropy of UMAP in order to implement it later in the code attached to this report. I will show you the result of the calculation of the Gradient of the UMAP cross-entropy:

$$\frac{\delta CE}{\delta y_i} = \sum_j\left[\frac{2abd_{ij}^{2(b-1)}P(X)}{1 + ad_{ij}^{2b}} - \frac{2b(1 - P(X))}{d_{ij}^2(1 + ad_{ij}^{2b})}\right](y_i - y_j)$$

G)UMAP assigns initial low-dimensional coordinates Y using Graph Laplacian in contrast to random normal initialization used by T-SNE, this initialisation normaly should make minor effect but the Graph Laplacian still better than a random initialization anymore. The Graph Laplacian refer practicaly to the interesting methodology that combines Matrix Factorization and Neighbor Graph approaches to the dimension reduction problem. For me I used the scikit-learn Python library and easily I displayed the initial low-dimensional coordinates using the SpectralEmbedding function as the picture below token from the code zipped with this report show :

```
Entrée [47]: N_LOWER_DIMS = 2
             LEARNING_RATE = 1
             MAX_ITER = 150
             np.random.seed(12345)
             model = SpectralEmbedding(n_components = N_LOWER_DIMS, n_neighbors = 20)
             y = model.fit_transform(np.log(X_train[:,1:5]+1))
```

H) UMAP uses the Stochastic Gradient Descent (SGD) instead of the regular Gradient Descent (GD) like T-SNE . This improves the speed since for SGD we calculate the gradient from a random subset of samples instead of using all of them like for gradient Descent. In addition this also reduces the memory consumption since you we are no longer obliged to keep gradients for all the samples in the memory but for a subset only. For me I worked with a batch stochastic gradient descent and I have gor a better visualisation of data in low dimension Embedding.

3 The reasons that lead UMAP algorithm to preserve the Global structures of data and the T-SNE algorithm to preserve only the local structures.

3.1 The T-SNE algorithm can only preserves the local structures of the data

I will prove that T-SNE can only preserve the local structures by studying the divergence function of Kullback-Leibler. I will suppose that X is a distance between points in a large dimension space and Y is the distance between point in the low dimension space.To simplify the prove we have to do this two transformations from the two first pictures to the third one:

$$p_{j|i} = \frac{\exp\left(-||x_i - x_j||^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-||x_i - x_k||^2/2\sigma_i^2\right)} \qquad q_{ij} = \frac{\left(1+||y_i - y_j||^2\right)^{-1}}{\sum_{k\neq l}\left(1+||y_k - y_l||^2\right)^{-1}} \qquad P(X) \approx e^{-X^2} \qquad Q(Y) \approx \frac{1}{1+Y^2}$$

So from the definition of the Kullback-Leibler cost I will have:

$$KL(X,Y) = P(X)\log\left(\frac{P(X)}{Q(Y)}\right) = P(X)\log P(X) - P(X)\log Q(Y)$$

and then by considering only the variable part that is the second one that contain the Q . I will have:

$$KL(X,Y) \approx -P(X)\log Q(Y) = e^{-X^2}\log\left(1+Y^2\right)$$

Now, let's start the analyses, For large distances X in high dimensions, Y can be basically any value in IR since the exponential term goes to zero . Therefore it can happen that points far apart in high dimensions end up close to each other in low dimensions. As a result, T-SNE does not guarantee that points far apart in high dimensions will be preserved to be far apart in low dimensions. However, it does guarantee that points close to each other in high dimensions will remain close to each other in low dimensions. So T-SNE is not really good at projecting large distances into low dimensions, so it preserves only the local data structure.

3.2 UMAP algorithm can preserve the global structures of the data

From the Two first relations I will have the cross-Entropy of UMAP like the third relation:

$$CE(X,Y) = \sum_i\sum_j\left[p_{ij}(X)\log\left(\frac{p_{ij}(X)}{q_{ij}(Y)}\right) + (1-p_{ij}(X))\log\left(\frac{1-p_{ij}(X)}{1-q_{ij}(Y)}\right)\right]$$

$$P(X) \approx e^{-X^2} \qquad Q(Y) \approx \frac{1}{1+Y^2}$$

$$CE(X,Y) = e^{-X^2}\log\left[e^{-X^2}\left(1+Y^2\right)\right] + \left(1-e^{-X^2}\right)\log\left[\frac{\left(1-e^{-X^2}\right)\left(1+Y^2\right)}{Y^2}\right]$$

$$\approx e^{-X^2}\log\left(1+Y^2\right) + \left(1-e^{-X^2}\right)\log\left(\frac{1+Y^2}{Y^2}\right)$$

For large distances X in high dimensions, the Cross Entropy(X,Y) of UMAP become like this :

$$X \to \infty : CE(X,Y) \approx \log\left(\frac{1+Y^2}{Y^2}\right)$$

So let's start again the analyses, With UMAP if Y is small, we get a high penalty because of the Y in the denominator of the logarithm, So Y is encouraged by the optimization of the cross-Entropy to be large so that the ratio under logarithm becomes 1 and we get zero penalty. As a result with UMAP we get a high value of Y when X is eventually high, so the global distances and structures are preserved when moving from high to low-dimensional space, It is exactly what we want from the UMAP algorithm .

informations about the code carried out in this project: Before each part of the code that I developped you will find an explanation of what I will do in this fragment. I developped the code by using jupyter notebook so to rebuild the experimentations carried out in this project you can use jupyter notebook and run the file .ipynb that you find zipped with this report .