UMAP Tutorial

Xinqian Wang

January 2022

This is a tutorial of UMAP which explains the algorithm's fundamental details. The UMAP paper is mostly related to the work of t-SNE. I would recommend people who is interested to it should firstly understand t-SNE and then read the UMAP paper. This tutorial, is built on my UMAP knowledge together with the source code of UMAP (https://github.com/lmcinnes/umap/tree/master/umap).

1 Neighbour Embeddings

Preserve the high dimensional data onto low dimensional space by using euclidean distance doesn't work because of the curse of dimensionality. The points in high dimensionality tend to be far away from each other but in 2-D space we can see some points can overlap with each other. Many of the dimension reduction algorithms use K Nearest Neighbours to do neighbour embedding. There are many ways for computing the KNN matrix. Some KNN matrix's approximation can also work (and more faster).

2 Fuzzy Simplicial Set Calculation

From the UMAP paper, it constructs the graph by making the simplicial set after the KNN matrix is built. The definition of simplicial set is mathematically defined in the paper. Computationally, calculating the parameters of σ and ρ.

$$\rho_i = \min\left\{d\left(x_i, x_{i_j}\right) \mid 1 \leq j \leq k, d\left(x_i, x_{i_j}\right) > 0\right\}$$

$$\sum_{j=1}^{k} \exp\left(\frac{-\max\left(0, d\left(x_i, x_{i_j}\right) - \rho_i\right)}{\sigma_i}\right) = \log_2(k)$$

Then, calculating the weight of the points in the high dimension space.

$$w\left(\left(x_i, x_{i_j}\right)\right) = \exp\left(\frac{-\max\left(0, d\left(x_i, x_{i_j}\right) - \rho_i\right)}{\sigma_i}\right)$$

Lastly, return the new graph with the weights between node i and node j. The weights can be understood as how much possibility the node i and node j are connected. We set them as $\bar{G} = (V, E, w)$. Set V are nodes from source data, E are KNN edges with weights w we just calculated. During the computation, the instance of the Graph is just a sparse matrix with a shape of (N,N). The sparse matrix has no entries below or equal to zero but only the value of w. This is realized by the python library scipy.sparse.coo-matrix in the source code.
After we got the weight $w((x_i, x_{i_j}))$, $1 \leq j \leq K, 1 \leq i \leq N$. The problem came to be the matrix A is not symmetric (The weights of $w_{ij} and w_{ji}$ are unequivalent). The symmetrization is made by the formula $A^* \leftarrow A + A^\top - A \circ A^\top$.

3 Spectral Embedding

Our goal is to initialize a set of matrix Y that represents the data in the low dimension space.The author of UMAP said the matrix can be initialized randomly. However, considering the symmetric Laplacian of the graph $\bar{G}$ is a discrete approximation of the Laplace Beltrami operator of the manifold, we can use a spectral layout as an initialization.
Suppose the weighted adjacency matrix of $\bar{G}$ is A. Set A's degree matrix as D. $L \leftarrow D^{1/2}(D - A)D^{1/2}$. Then, we take L's d-th first largest eigenvectors to make the Y. d is the dimensionality we want to reduce to.

evec $\leftarrow$ Eigenvectors of L (sorted)

Y $\leftarrow evec[1 \ldots d + 1]$

return Y

4 Loss Function and Embedding Optimization

The loss function we used here is the same as it used in t-SNE, the Cross Entropy Loss. The goal of the dimension reduction is to preserve the structure of source data in the low dimension space. From t-SNE the loss function is:

$$L = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}} = \sum_{i,j} [p_{ij}(\log p_{ij} - \log q_{ij})]$$

p_{ij} represents the possibilities of connectivity in source data and q_{ij} represents the possibilities in low dimension data. The possibility is calculated through a kernel, usually is an Gaussian kernel.
From the formulation above, we can see the value of p_{ij} is fixed since the source data does not change. All we need to care about is the formula of $-\sum_{i,j} p_{ij} \log q_{ij}$. From the t-SNE paper it also further decomposes the rest of

the loss function into two parts:

$$-\sum_{i,j} p_{ij} \log w_{ij} + \log \sum_{i,j} w_{ij}$$

From the formula above, the first part is called the attractive force and the second part is called the repulsive force. The first part is determined by the node i 's neighbour, so to minimize it the weight w_{ij} needs to increase. The second part is the sum of the weight for all the nodes, so it needs to be decrease. From UMAP, the author wrote Cross Entropy as below:

$$C((A,\mu),(A,\nu)) = \sum_{a\in A} \mu(a) \log\left(\frac{\mu(a)}{\nu(a)}\right) + (1-\mu(a)) \log\left(\frac{1-\mu(a)}{1-\nu(a)}\right)$$

$$= \sum_{a\in A} (\mu(a)\log(\mu(a)) + (1-\mu(a))\log(1-\mu(a))) - \sum_{a\in A} (\mu(a)\log(\nu(a)) + (1-\mu(a))\log(1-\nu(a)))$$

The function $\mu(\cdot)$ can be seen as a kernel which used to embed points in the high dimension space. Similarly, $\nu(\cdot)$ is the kernel which used to embed points in the low dimension space. We only care about the second sum. As the result, we want to minimize:

$$-\sum_{a\in A} (\mu(a)\log(\nu(a)) + (1-\mu(a))\log(1-\nu(a)))$$

In the UMAP paper, $\nu(\cdot)$ is represented as $\Phi(\mathbf{x},\mathbf{y}) = \left(1 + a\left(\|\mathbf{x}-\mathbf{y}\|_2^2\right)^b\right)^{-1}$.

To accelerate the computation, the author used negative sampling (invented in word2vec model) to calculate the second part of the loss function above. Sample some unreal nodes from Y and make the negative edge samples with the target node.

$$a\, is\, the\, target\, node\, from\, \mathrm{Y}$$

$$c \leftarrow random\, sample\, from\, \mathrm{Y}$$

$$y_a \leftarrow y_a + \alpha \cdot \nabla(\log(1-\Phi))\,(y_a, y_c)$$

As for the first part, the algorithm samples some positive edges according to w and do gradient descent.

$$y_a \leftarrow y_a + \alpha \cdot \nabla(\log(\Phi))\,(y_a, y_b)$$

The α is just the learning rate. The two gradient descent formulas above are circulated multiple times until it get the max number of epoch.

This complete the UMAP algorithm.