A quick overview of Graph Embedding Methods

Poulain Timothée

14 December 2020

Summary

Introduction

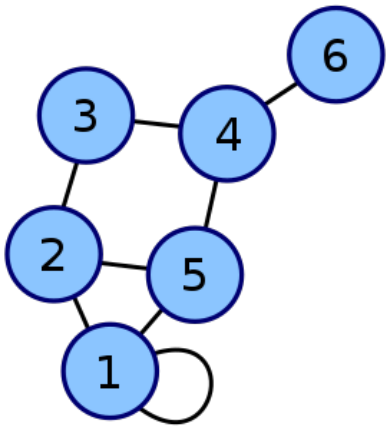
- Graph are naturally occurring in the variety of situations such as social media network, knowledge graph, users graph,...
- Analysing these graphs provides insights into how to make good use of the information hidden in graphs, and thus has received significant attention in the last few decades.
- Graph analytic tasks can be divided into four categories : Nodes clustering, link prediction, node retrieval/recommendation, visualisation
- **Major problem :** Most existing graph analytics methods suffer the high computation and space cost.
- How to fix this ?

Summary

Problem Formalization

Definition : Graph

A graph $G = (V, E)$, where $v \in V$ is a node and $e \in E$ is an edge. The adjacency matrix A of graph G contains non-negative weights associated with each edge : $A_{i,j} = s_{ij} \geq 0$. If v_i and v_j are not connected to each other, then $s_{ij} = 0$.

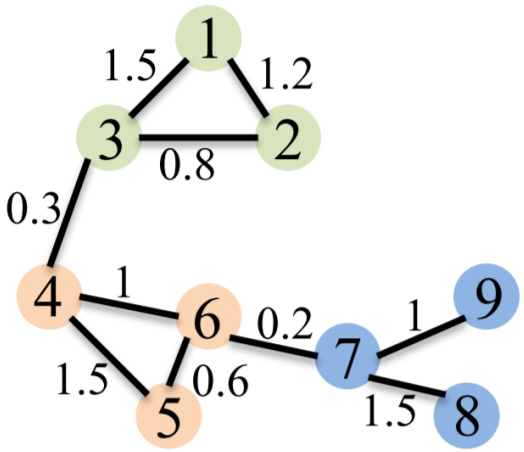


$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Problem Formalization : First-order proximity

Definition : First-order proximity

The **first-order proximity** between node v_i and node v_j is the weight of the edge e_{ij}, ie, $s_{ij} = A_{i,j}$

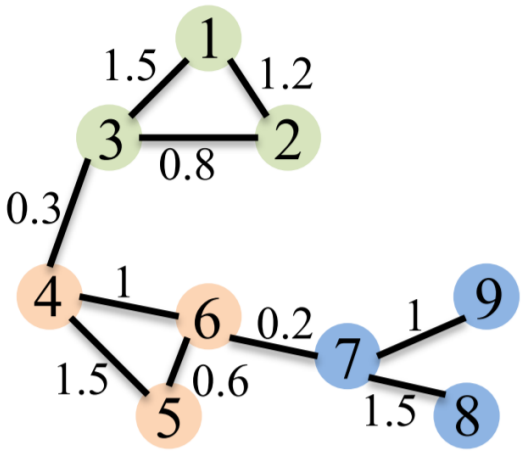


- Let $s_1 = [s_{11}^1, s_{12}^1, ..s_{1|V|}^1]$ denote the first-order proximity between v_1 and other nodes.
- In this case, $s_1^{(1)} = [0, 1.2, 1.5, 0, 0, 0, 0, 0, 0]$

Problem Formalization : Second-order proximity

Definition : Second-order proximity

The **second-order proximity** $s_{ij}^{(2)}$ between node v_i and v_j is a similarity between v_i's neighbourhood $s_i^{(1)}$ and v_j's neighborhood $s_j^{(1)}$

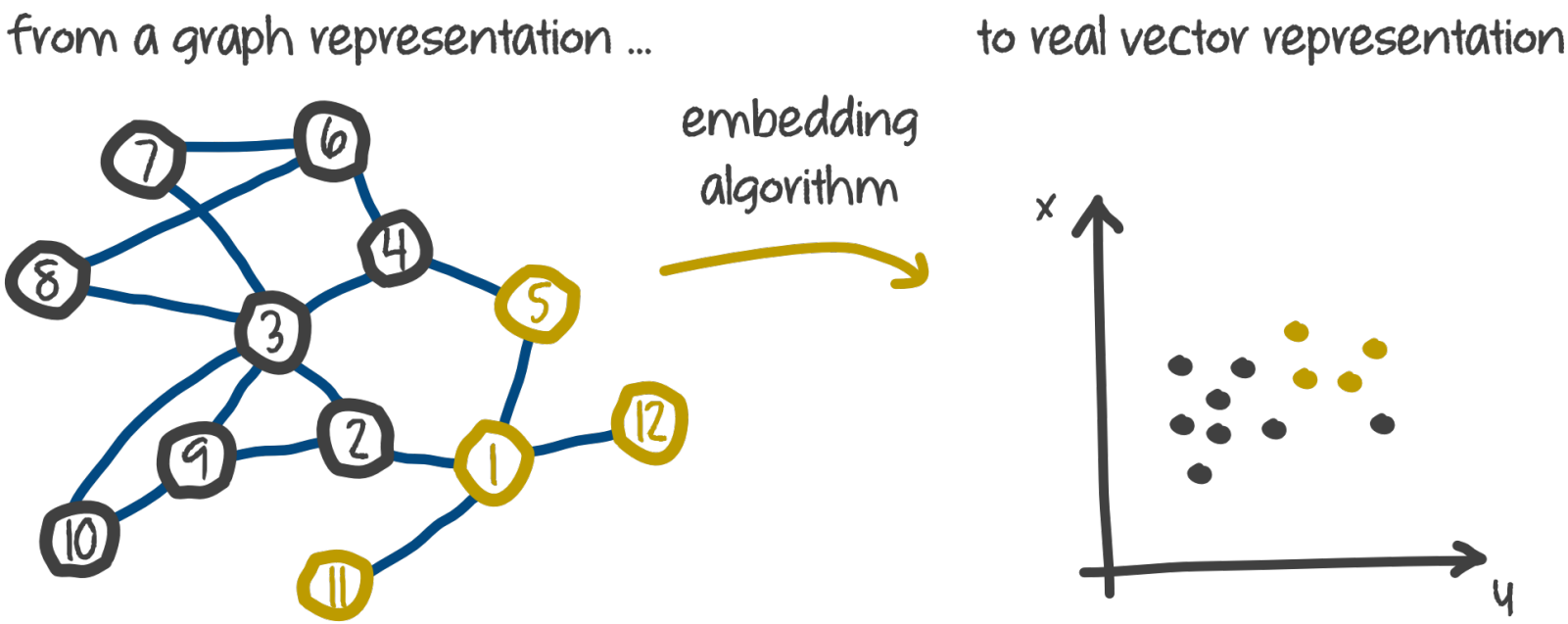


- Let's take a look at $s_{12}^{(2)}$ which is the similarity between $s_1^{(1)}$ and $s_2^{(1)}$.
- We can choose the cosine similarity measure and another one.
- In this case, $s_1^{(1)} = [0, 1.2, 1.5, 0, 0, 0, 0, 0, 0]$, $s_2^{(1)} = [1.2, 0, 0.8, 0, 0, 0, 0, 0, 0]$
- So, $s_{12}^{(2)} = cosinus(s_1^{(1)}, s_2^{(1)}) = 0.43$

Problem Formalization

Definition : Graph Embedding

Given a graph $G = (V, E)$, a graph embedding is a mapping $f : v_i \longrightarrow y_i \in \mathbb{R}^d$ such that $d \ll |V|$ and the function f preserves some proximity measure defined on graph G.



Summary

Factorization-based methods

- Matrix factorization based methods, which are directly inspired by classic techniques for dimensional reduction, represent the connections between nodes in the form of a matrix and factorize this matrix to obtain the embeddings.
- A number of algorithms have emerged, from the simplest MDS (directly adopted the Euclidean distance between two features), passing through the Locally Linear Embedding algorithm up to Laplacian Eigenmaps (spectral techniques)...

DL-based methods

- We can separate Deep learning based graph embedding into two types :
  - Using a random walk : The graph is represented as a set of random walk paths and we can apply deep learning methods to the sampled paths which preserve the properties of the graph.
    Some references : DeepWalk [Perozzi, Al-Rfou et Skiena 2014], node2vec [Grover et Leskovec 2016], ...
  - **Not using a random walk :** The multi-layered learning architecture is a robust and effective solution to encode the graph into low dimensional space.
  - Focusing on one method in particular : **Graph Convolution Network**

Summary

What is a Graph Convolutional Network ?

- A Graph Convolutional Network is a multilayer neural network that operates directly on a graph and induces embedding vectors of nodes based on properties of their neighborhoods.
- Graph convolution network is equal to a CNN based on graphs.

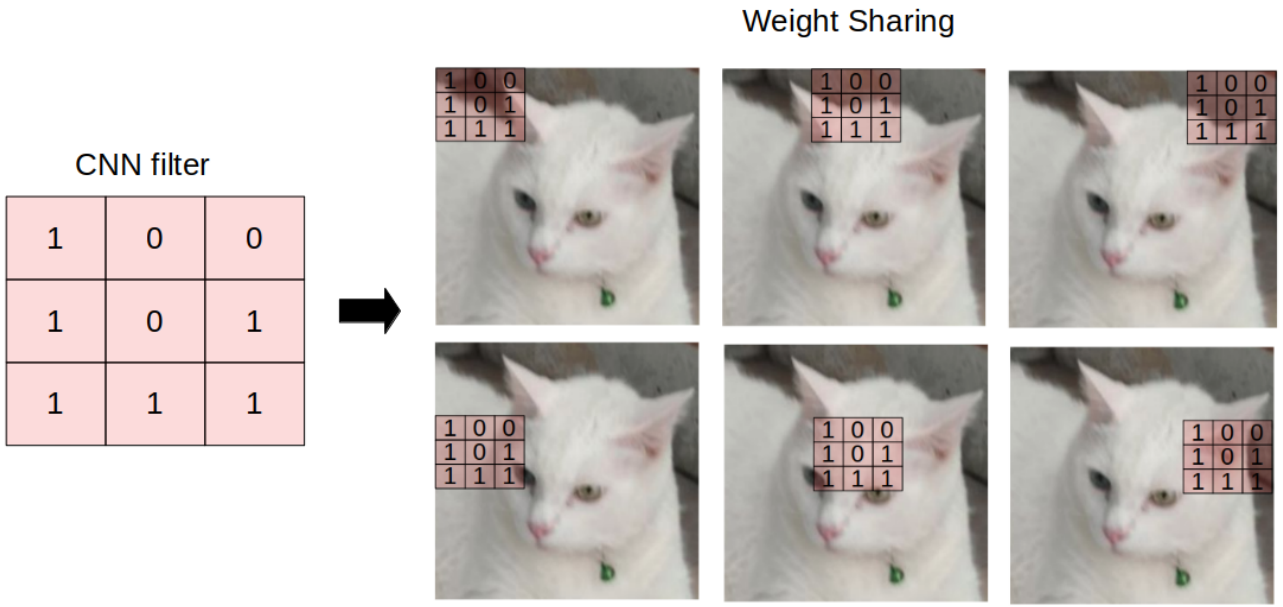


Figure – The same filter in CNNs is applied throughout the image

GCN Formalization

- Goal : Learn a **function of features f** on a graph $\mathcal{G} = (V, E)$ which takes as input :
  - A **feature matrix** $X \in \mathbb{R}^{NxD}$ (N :number of nodes, D :dimension of features)
  - A graph structure representative matrix, **adjacency matrix A in** $\mathbb{R}^{NxN}$
  - and produces a **output feature matrix** $Z \in \mathbb{R}^{NxF}$ (F : number of output features per node)
- Each neural network layer can be written as a non-linear function

$$H^{(l+1)} = f(H^{(l)}, A)$$

with $H^{(0)} = X$ and $H^{(l)} = Z$, L is the number of layers.

GCN

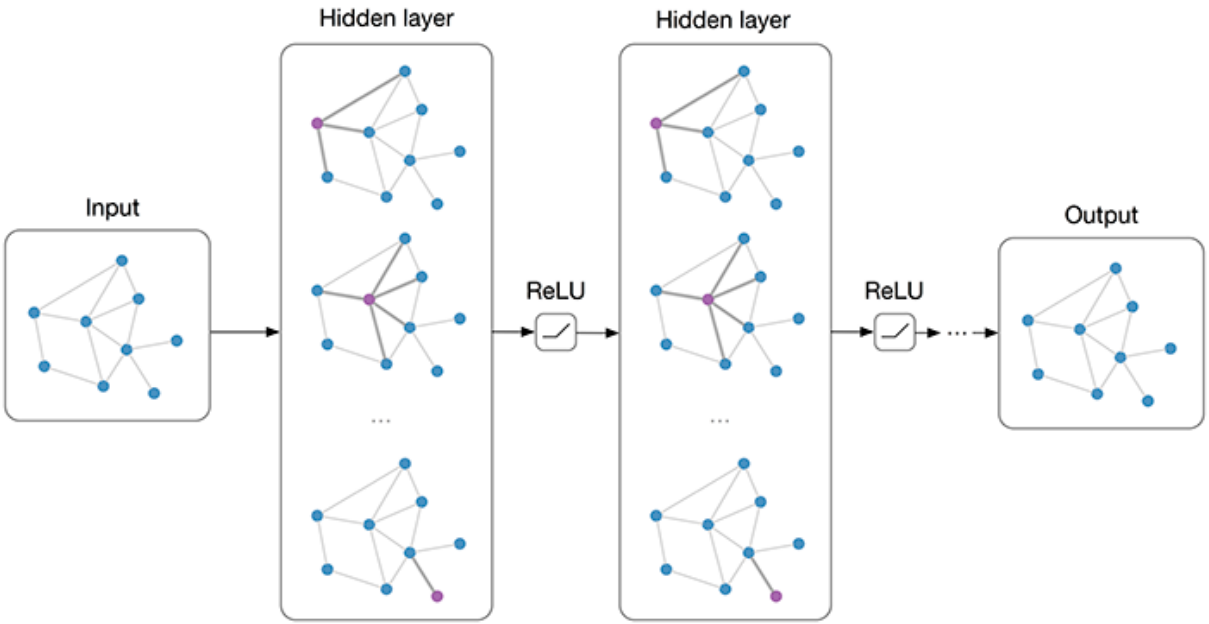


Figure – GCN with two layers

GCN : Propagation rule

- Example of very simple form of layer-wise propagation rule :

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

where

  - $W^{(l)}$ is a weight matrix for the layer l
  - σ is a non-linear activation function such as the ReLU

- Layer-wise propagation rule introduced by Kipf et Welling 2016 :

$$f(H^{(l)}, A) = \sigma(\underbrace{\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}}_{NSAM\ [1]} H^{(l)}W^{(l)})$$

with $\hat{A} = A + I$ (I : Identity matrix) and $\hat{D}$ is the diagonal node degree matrix of A.

1. Normalized Symmetric Adjacency Matrix

Summary

GCN for Text Classification

- **Graph Convolutional Networks for Text Classification**
  - Yao, Mao et Luo 2019
- **Goal :** Transform text classification problem into a node classification problem
- The major difference is that these methods (Doc2vec Le et Mikolov 2014, ...) build text representation after learning words embeddings while we learn word and document embeddings simultaneously.

GCN for Text Classification

- Building a wide and heterogeneous [2] text graph which contains word nodes and document nodes so that global word co-occurrence can be explicitly modeled and graph convolution can be easily adapt.

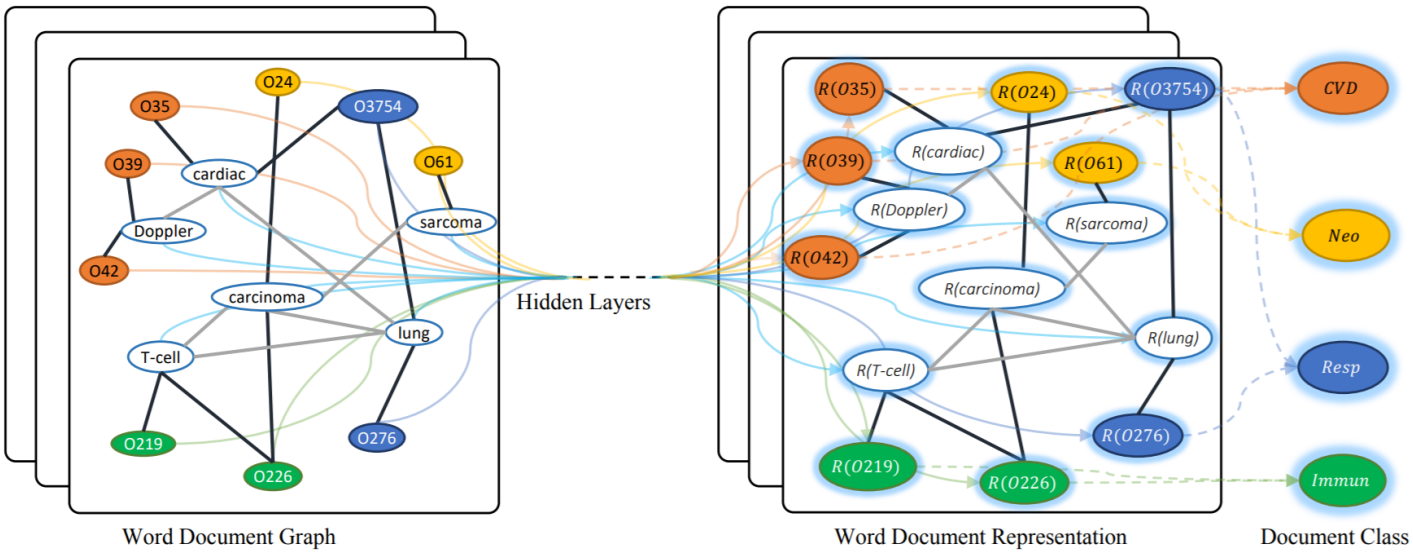


Figure – Text GCN with "O" nodes represent documents, other represent word. Black bold lines are doc-word edges and regular lines are words-words edges

2. All nodes in G don't belong to a single type and all edges don't belong to one single type

GCN for Text Classification

- Set feature matrix $X = I$ as identity matrix (each word/doc represent as a one-hot vector)
- The weight of the edge between a **document node** and a **word node** is the TFIDF of the word in the document.
- The weight of the edge between a word node and a **word node** is the point-wise mutual information (PMI).

GCN for Text Classification

- So, formally, our adjacency matrix is constructed as follows :

$$A_{ij} = \begin{cases} PMI(i,j) & i,j \text{ are words, } PMI(i,j) > 0 \\ \text{TF-IDF}_{ij} & i \text{ is document, } j \text{ is a word} \\ 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

GCN for Text Classification

- After building the adjacency matrix A, we'll use the multi-layer GCN previously explained and and only replaced the second layer of RELU in a Softmax because our goal is to classify the documents.

$$Z = softmax(\overbrace{\hat{A}\ ReLU(\underbrace{\hat{A}XW_0}_{\text{first layer}})W_1}^{\text{second layer}}) \quad (1)$$

with W_0 and W_1 are the weight matrix for layer 0 et 1 and $\hat{A} = \hat{D}^{-\frac{1}{2}}(A + I)\hat{D}^{-\frac{1}{2}}$

- minimize the cross-entropy error over all labeled documents :

$$\mathcal{L} = -\sum_{d \in \mathcal{Y}_{\mathcal{D}}} \sum_{f=1}^{F} Y_{df} \ln Z_{df}$$

where Y_D is the set of document indices that have labels and F is equals to the number of classes.

GCN for Text Classification : Results

Model	**20NG** [3]	**R8** [4]
TF-IDF+LR	0.8319 ± 0.0000	0.9374 ± 0.0000
Bi-LSTM	0.7318 ± 0.0185	0.9631 ± 0.0033
fastText	0.7938 ± 0.0030	0.9613 ± 0.0021
Text GCN	0.8634 ± 0.0009	0.9707 ± 0.0010

Table – Accuracy measure on document classification task. They run all models 10 times and report mean ± standard deviation

3. The 20 newsgroups dataset comprises around 18000 (11k train, 7.5k test) newsgroups posts on 20 topics.

4. A dataset subset of Reuters 21578 comprises around 8k documents (5.5k train, 2.1k test) into 8 categories.

Summary

Discussion and conclusion

- Only a very small part of all Graph Embeddding Techniques
- A comprehensive survey Graph Embedding : Problems, Techniques and Applications [Cai, Zheng et Chang 2018], the authors propose to categorize graph embedding work based on problem setting.
-

Summary

References

Cai, Hongyun, Vincent W Zheng et Kevin Chen-Chuan Chang (2018). "A comprehensive survey of graph embedding : Problems, techniques, and applications". In : IEEE Transactions on Knowledge and Data Engineering 30.9, p. 1616-1637.

Grover, Aditya et Jure Leskovec (2016). "node2vec : Scalable feature learning for networks". In : Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery p. 855-864.

Kipf, Thomas N et Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In : arXiv preprint arXiv :1609.02907.

Le, Quoc et Tomas Mikolov (2014). "Distributed representations of sentences and documents". In : International conference on machine learning, p. 1188-1196.

Perozzi, Bryan, Rami Al-Rfou et Steven Skiena (2014). "Deepwalk : Online learning of social representations". In : Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery p. 701-710.

Yao, Liang, Chengsheng Mao et Yuan Luo (2019). "Graph convolutional networks for text classification". In : Proceedings of the AAAI Conference on Artificial Intelligence. T. 33, p. 7370-7377.

References

Thank you for your attention !

For further..

- Point-wise mutual information (PMI),

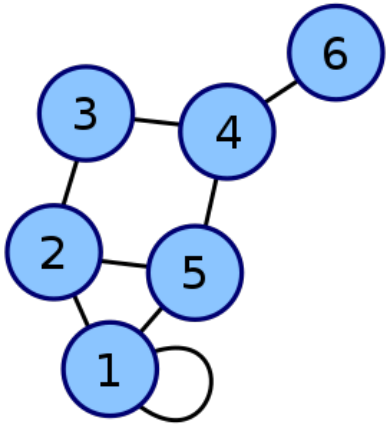
$$PMI(i,j) = \log \frac{p(i,j)}{p(i)p(j)}$$

$$p(i,j) = \frac{\sharp W(i,j)}{\sharp W}$$

$$p(i) = \frac{\sharp W(i)}{\sharp W}$$

where $\sharp W(i)$ is the number of sliding windows in a copus that contain word i, $\sharp W(i,j)$ is the number of sliding windows that contain both word i and j, and $\sharp W$ is the total number of sliding windows in the corpus.

Adjacency matrix

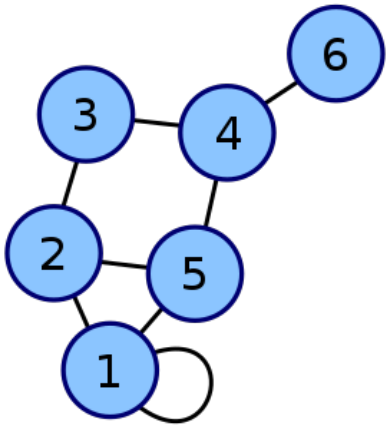


$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

- Given G $= (V, E)$, the adjacency matrix A of G relating to this set of vertices is the n $\times$ n Boolean matrix A with :

$$a_i j = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{Otherwise} \end{cases}$$

Degree matrix



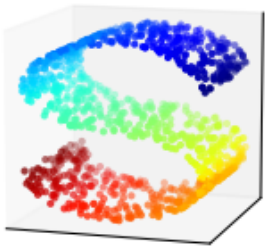
$$\begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

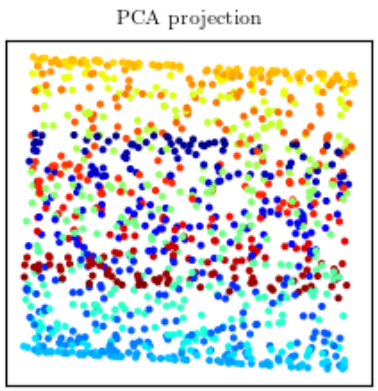
- Given G $= (V, E)$, the degree matrix D for G is a $n \times n$ diagonal matrix defined as :

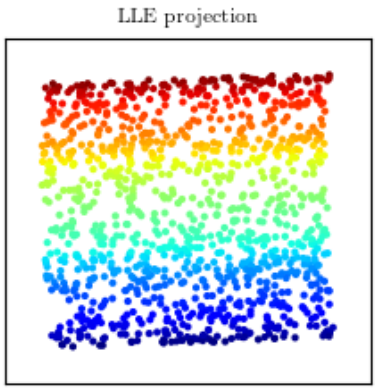
$$d_ij = \begin{cases} deg(v_i) & \text{if } i = j \\ 0 & \text{Otherwise} \end{cases}$$

LLE : Main idea

- Other dimensionality reduction methods fail to be successful on nonlinear space. (PCA, . . .)
- LLE takes advantage of the local geometry and pieces it together to preserve the global geometry on a lower dimensional space.







Locally Linear Embedding

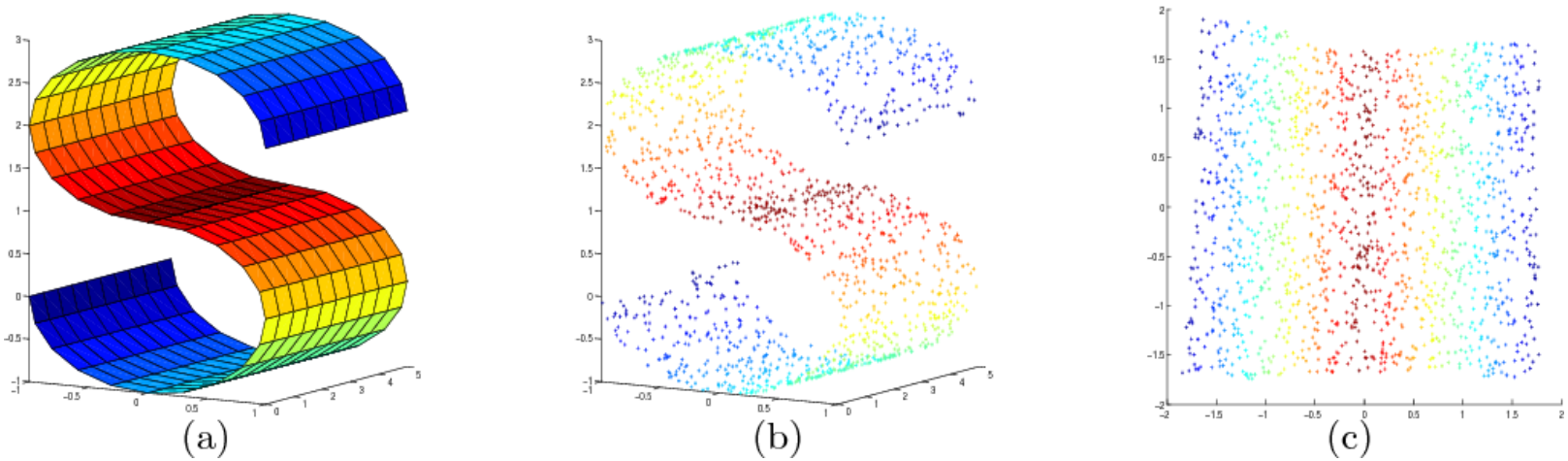


Figure – N-dimensional manifold to 2-dimensional space

- LLE basic idea is to take n-dimensional manifold and cast it into a lower dimensional space while preserving the manifold geometric features.

Locally Linear Embedding

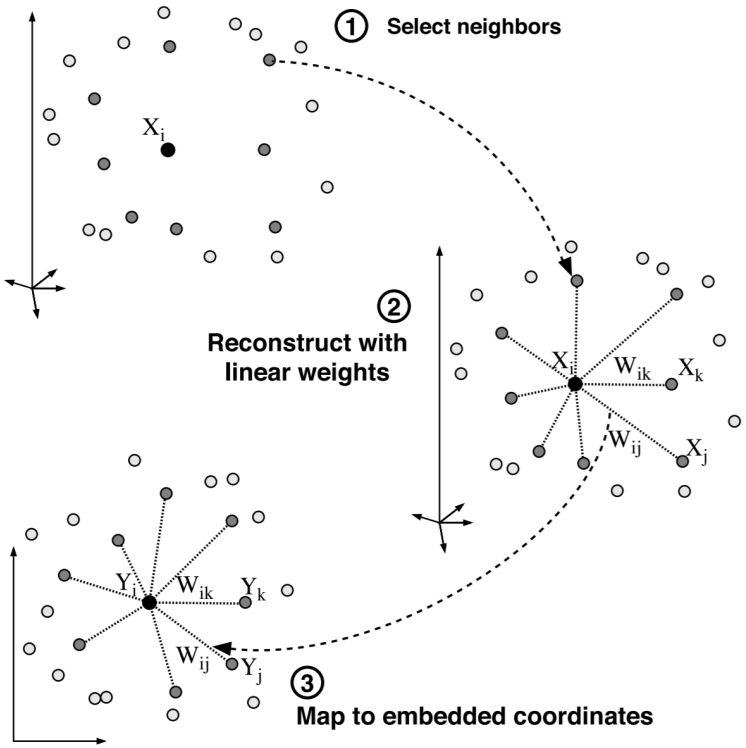


Figure – Caption

- **Step 1 :** Assign neighbors to each data point X_i using the K-nearest neighbors (K is the only parameters)
- **Step 2 :**
  - Rows of the weight matrix must equal to 1 : $\sum_j A_{ij} = 1$
  - Minimize the cost function below, ie, find the best linear combination of A_{ij} which reconstructs X_i

$$\epsilon(W) = \sum_i |\hat{X}_i - \sum_j A_{ij}\hat{X}_j|^2$$

Factorization-based methods

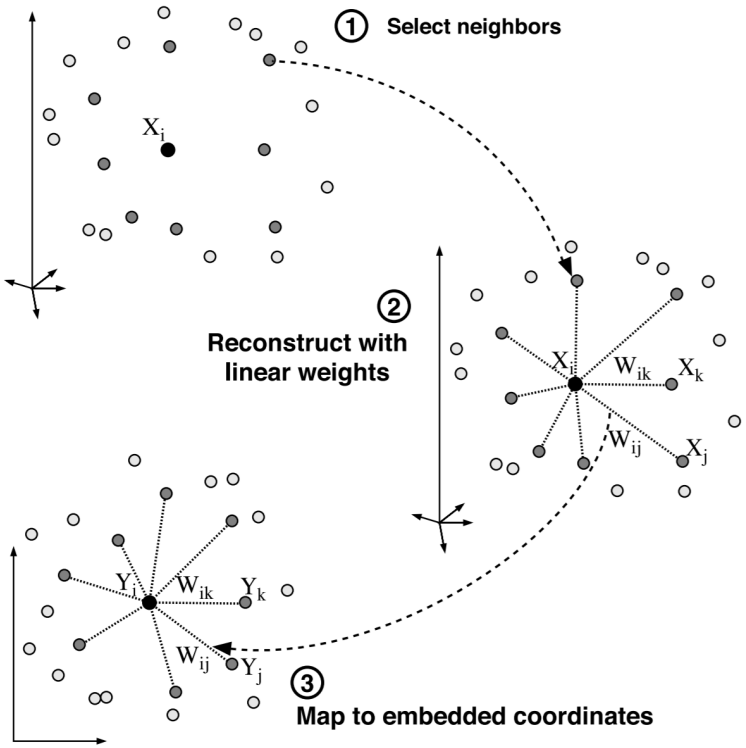


Figure – Caption

- **Step 3 :** Thanks to weights W_{ij} that we were previously defined in **step 2**, we obtain the embedding $Y^{N\times d}$ by minimizing the cost function below :

$$\psi(Y) = \sum_i |\hat{Y}_i - \sum_j W_{ij}\hat{Y}_j|^2$$

Factorization-based methods

- The LLE method is simply a tiny part of all Factorization-based methods that they exist. Indeed, there are no less than 30 in the surveys (Laplacian Eigenmaps, GraRep, Hope, Rescal, ...)
- For the LLE method, several limations appear. Besides the temporal complexity which is of $\mathcal{O}(|E|d^2)$, it's a shallow model and preserves only the structure of the first-order proximity.
- Indeed, Due to the limited representation ability of shallow models, it's difficult for them to capture the highly non linear network structure.