Deep Learning and Data Science (CSE5851) **Assignment4**

Department of Statistics and Data Science **2020321803 Song minsoo**

March 25, 2021

1. Matrix Factorization for Network Embedding Suppose that given a network, $\boldsymbol{A}$ is the $V \times V$ adjacency matrix and $\boldsymbol{z}_u \in R^d$ is the embedding vector of vertex $u \in V$, where V denotes the set of vertices. Then, we aim at finding the embedding vectors such that the following loss function is minimized:

$$L = \sum_{(u,v)\in V\times V} (\boldsymbol{z}_u^T \boldsymbol{z}_v - \boldsymbol{A}_{u,v})^2$$

where $\boldsymbol{A}_{u,v}$ is the (u, v)-th element of $\boldsymbol{A}$. That is, we are interested in finding the embedding vectors based on matrix factorization. Use the file " karate_club.adjlist ", which corresponds to the adjacency list, so that you work on the dataset for the Zachary' s karate club network.

- code

```python
import pandas as pd
import numpy as np

# make adjacency matrix
adj = pd.read_csv('adj.txt', header=None)
adj_matrix = np.zeros((len(adj),len(adj)))
for i in range(len(adj)):
    for j in range(len(adj)):
        if i == j:
            continue
        if str(j) in adj.iloc[i,0].split(' '):
            adj_matrix[i,j] = 1
            adj_matrix[j,i] = 1
# label
label = np.loadtxt('karate_label.txt')
```

Figure 1: set up code

(a) (85 points) Perform matrix factorization via stochastic gradient descent (SGD) to find the embedding matrix whose size is $V \times d$. To this end, update the embedding vectors according to the following rules:

$$\boldsymbol{z}_u \leftarrow \boldsymbol{z}_u - \eta e_{u,v} \boldsymbol{z}_v$$

$$\boldsymbol{z}_v \leftarrow \boldsymbol{z}_v - \eta e_{u,v} \boldsymbol{z}_u$$

where $e_{u,v} = \boldsymbol{z}_u^T \boldsymbol{z}_v - A_{u,v}$ and η is the learning rate. Set the dimension of each embedding vector to $d = 4$. You may set other hyperparameters including η arbitrarily. For more details of updates via SGD, refer to Section 2 in the article below: http://dm.postech.ac.kr/MLGF MF/fp352.pdf . (Do not consider any regularization)

- code
  - randomly initialize the embedding vectors
  - update embedding vectors with SGD method

```python
embedding_dim = 4
learning_rate = 0.005
n_iter = 200

np.random.seed(seed=0)
embedding_vec = np.random.rand(len(adj),embedding_dim)

for _ in range(n_iter):
    # e_vu  = z_v^T z_u - A_vu
    # update after each 1 iteration
    e_matrix = np.matmul(embedding_vec, embedding_vec.T) - adj_matrix
    for i in range(0, 34):
        for j in range(0, 34):
            # no consider node itself
            if i == j:
                continue
            # SGD
            embedding_vec[i] -= learning_rate * e_matrix[i][j] * embedding_vec[j]
```

Figure 2: code making embedding vector

(b) (15 points) Plot V points on the two dimensional space through the t-SNE visualization tool . Make discussions on how the vertices are embedded in comparison with the figure below. Use the Scikit Learn package on Python by changing the default setting on 'perplexity' to a small value.

- code

```
from sklearn.manifold import TSNE

tsne_result = TSNE(n_components=2, perplexity=4).fit_transform(embedding_vec)

import matplotlib.pyplot as plt

plt.title(f'n_iter : {n_iter}, learning_rate : {learning_rate}')
plt.scatter(tsne_result[:,0],tsne_result[:,1], c=list(map(int, label[:,1])))
for i in range(0, 34):
        plt.text(float(tsne_result[i,0]), float(tsne_result[i,1]), i+1 , fontsize=10)
plt.show()
```

Figure 3: t-sne code

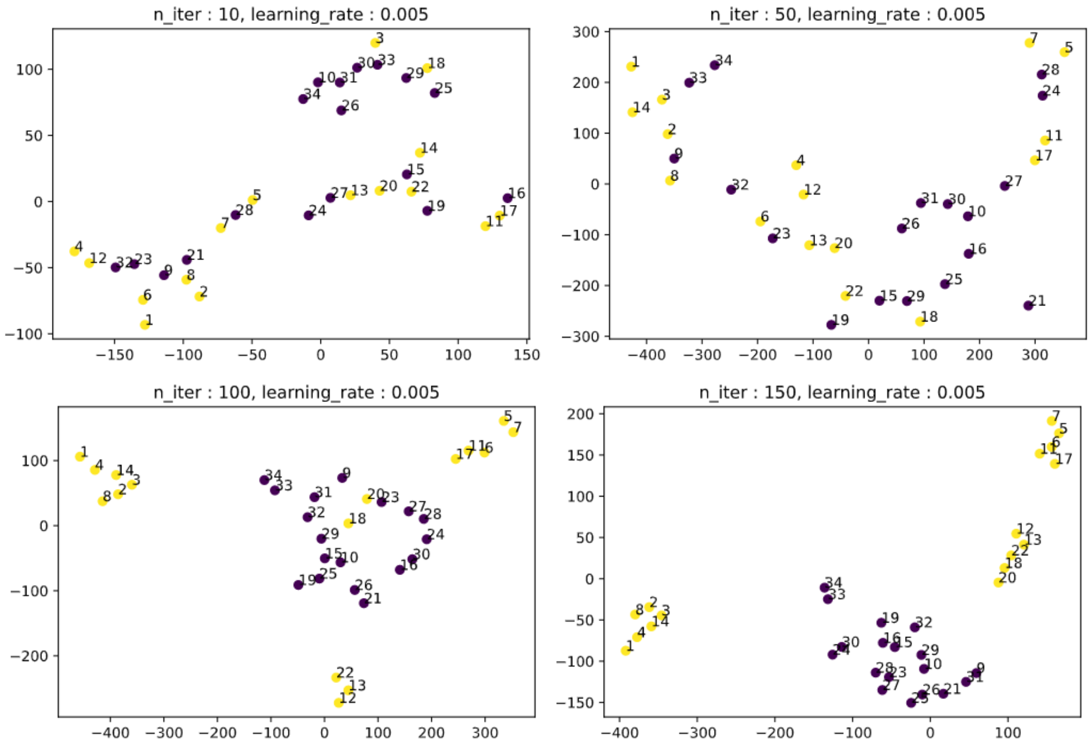

Figure 4: t-sne (with perplexity=4) result

- final result
  - as we can see, the result is quite satisfied.
  - two groups are divided well.
  - it means that our embedding vectors are well defined as intended.

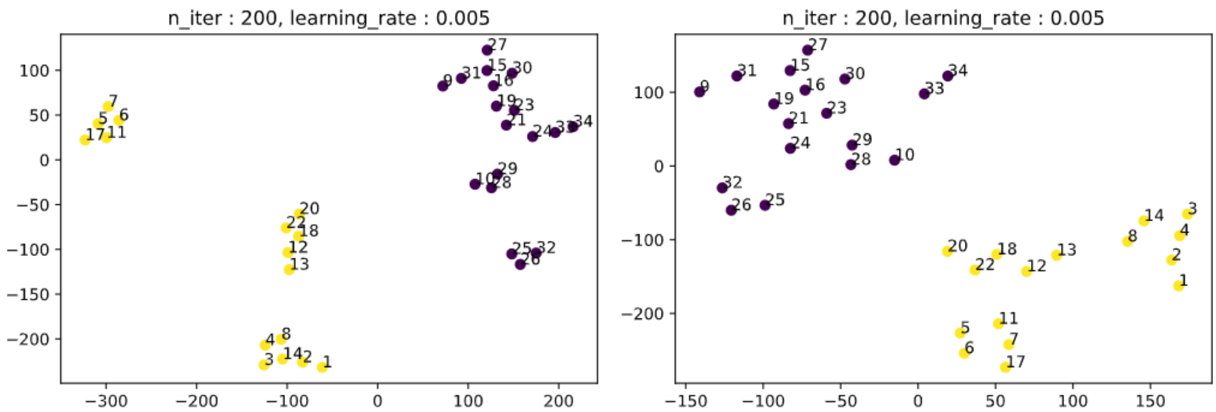

Figure 5: t-sne (left : perplexity=4, right : perplexity=8) ; # of iter=200, learning rate=0.005