Deep Learning and Data Science (CSE5851) **Assignment10**

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

May 14, 2021

1. (100 points) [node2vec with Negative Sampling] Find embedding vectors of vertices in a given network based on a biased random walk-aided approach, the so-called node2vec. Use one of the files "karate_club.adjlist" or "karate_club.edgelist", which correspond to the adjacency list and edge list, respectively, so that you work on the dataset for the Zachary's karate club network. Adopt the stochastic gradient descent (SGD) optimizer and hyperparameters set to the following values:

- dimension of each embedding vector: 2
- learning rate: 0.02 -walk length: 10
- window size: 3
- walks per vertex: 5
- the number of negative samples: 5,

which can however be replaced by other ones if another setting leads to a better result. To see how embeddings behave according to two key parameters including return parameter p and in-out parameter q, consider the following two cases:

- $q < 1 < p$ (Case 1)
- $p < 1 < q$ (Case 2),

where p and q can be arbitrarily chosen as long as the above conditions are fulfilled. You may set other hyperparameters arbitrarily. Use Negative Sampling to approximate the probability distribution. For each case, to show the convergence, plot the loss versus the number of epochs using the above dataset. Additionally, plot all resulting embedding vectors on the two-dimensional space. Make discussions on how Case 1 differs from Case 2 on the embedding space.

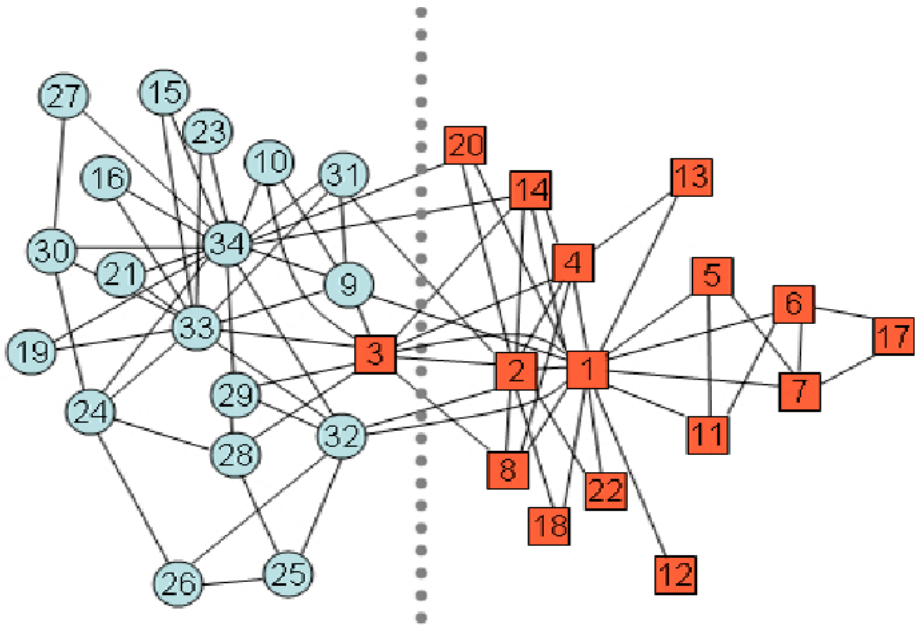

Figure 1: karate network

- q < 1 < p (Case 1)

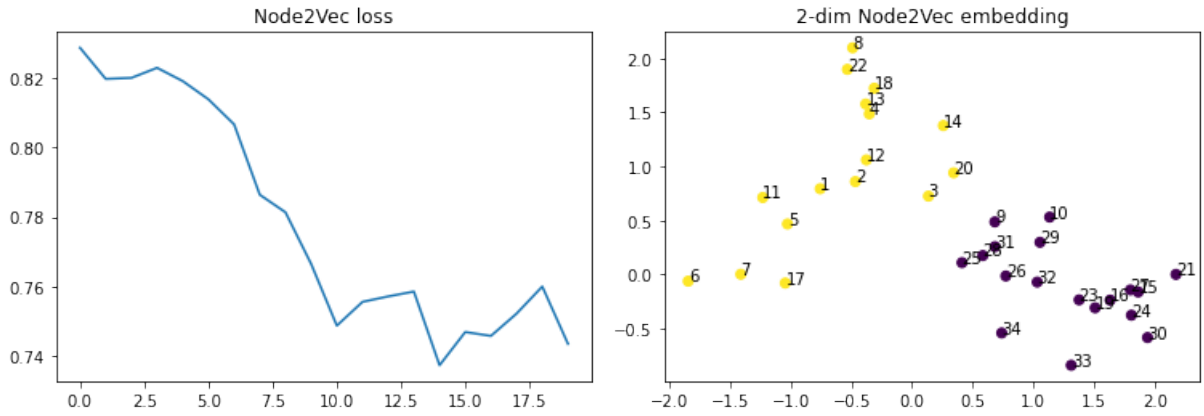

Figure 2: loss and embedding

- p < 1 < q (Case 2),

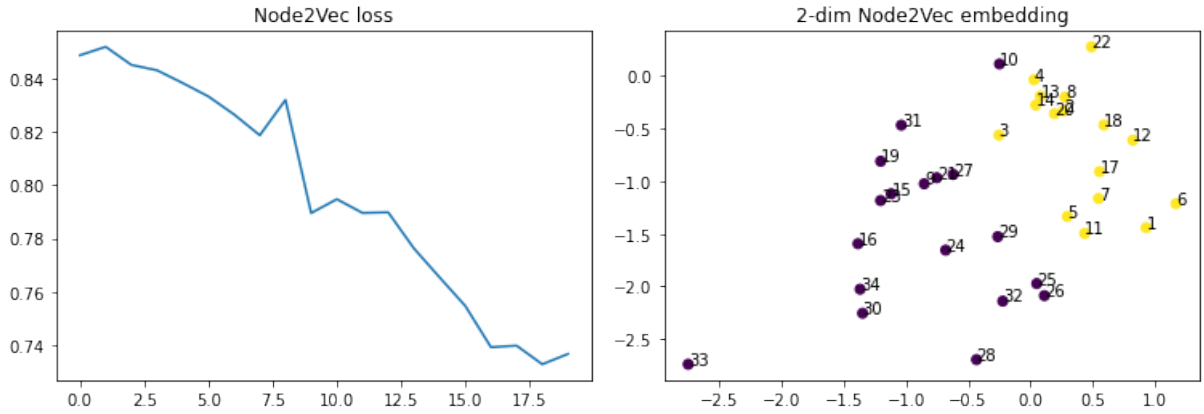

Figure 3: loss and embedding

- Case1의 경우 node 1,2와 node 33,34가 상대적으로 가까이 있는 것을 알 수 있다. 이는 각자의 군집에서 비슷한 위치(역할)을 하기 때문이라고 예상할 수 있다.
- Case2의 경우 BFS이기 때문에 좀 더 끼리끼리 뭉쳐있는 모습
  - 예를 들어, 노란색 node 1,6,7,17이 모여있는데 실제로 Figure1의 모습을 보면 서로 가깝게 연결되어 있다는 점을 알 수 있다. 이는 Case1과 다른 모습을 보여준다.

```python
class Node2Vec:
    def __init__(self,
                 adj_matrix,
                 embedding_dim=2,
                 walks_per_vertex=5,
                 walk_len=10,
                 window_size=3,
                 num_negetive_sample=5,
                 learning_rate=0.02,
                 p=1, q=1):

        self.adj_matrix = adj_matrix
        self.embedding_dim = embedding_dim
        self.walks_per_vertex = walks_per_vertex
        self.walk_len = walk_len
        self.window_size = window_size
        self.learning_rate = learning_rate
        self.k = num_negetive_sample
        self.whole_sampling_prob = np.power(np.sum(adj_matrix, axis=1), 3/4)
        self.p = p
        self.q = q

        self.w1 = np.random.rand(len(adj_matrix),embedding_dim)
        self.w2 = np.random.rand(embedding_dim,len(adj_matrix))

        self.loss = 0.0
        self.epoch_loss = []

    def _biased_walk(self, start_node: int)-> List:
        walk = [0] * self.walk_len
        walk[0] = start_node
        next_node = np.random.choice(np.where(self.adj_matrix[start_node]==1)[0])
        walk[1] = next_node
        node = next_node

        for i in range(2, self.walk_len):
            prev_node = walk[i-1]
            node_1 = np.where((self.adj_matrix[node]==1)&(self.adj_matrix[prev_node]==1))[0].tolist()
            node_q = np.where((self.adj_matrix[node]==1)&(self.adj_matrix[prev_node]==0))[0].tolist()
            transition_prob = [1/self.p] + [1] * len(node_1) + [1/self.q] * len(node_q)
            next_node = np.random.choice(a=[prev_node]+node_1+node_q, size=1,
                                         p=np.array(transition_prob)/sum(transition_prob),
                                         replace=False)
            walk[i] = next_node[0]
            node = next_node[0]
        return walk

    def _softmax(self, a: np.array)-> np.array :
        c = np.max(a)
        exp_a = np.exp(a-c)
        sum_exp_a = np.sum(exp_a)
        y = exp_a / sum_exp_a
        return y

    def _skip_gram_train(self, walk: List)-> None:
        for idx, input_node in enumerate(walk):
            # make dataset
            left_idx = idx - self.window_size
            right_idx = idx + self.window_size
            if left_idx < 0:
                left_idx = 0
            if right_idx > self.walk_len-1:
                right_idx = self.walk_len
            left_node = walk[left_idx:idx]
            right_node = walk[idx+1:right_idx+1]
            output_node = left_node + right_node

            # forward
            hidden = self.w1[input_node]
            ## |hidden| = (2,)
            out = np.matmul(hidden, self.w2)
            ## |out| = (34,)

            # loss calculate
            self.loss += (-np.sum(out[output_node]) \
                                + len(output_node)*np.log(np.sum(np.exp(out))))\
                               / (self.walk_len*self.walks_per_vertex*len(self.adj_matrix))

            # Negative sampling
            # walk에 없는 node중에서 뽑기
            neg_sample_idx = np.delete(np.arange(len(self.adj_matrix)), output_node)
            neg_sampling_prob = self.whole_sampling_prob[neg_sample_idx]
            neg_samples = np.random.choice(a=neg_sample_idx, size=self.k,
                                           p=neg_sampling_prob/np.sum(neg_sampling_prob),
                                           replace=False)

            # calcaulate gradient
            dEdo = np.zeros(len(out))
            dEdo[neg_samples] = self._softmax(out[neg_samples]) * len(output_node)
            dEdo[output_node] = dEdo[output_node] - 1.0
            dEdw2 = hidden.reshape(self.embedding_dim,1) @ dEdo.reshape(1,len(self.adj_matrix))

            # update
            self.w2 = self.w2 - self.learning_rate * dEdw2
            self.w1[input_node] = self.w1[input_node] - \
                self.learning_rate * np.matmul(self.w2, dEdo)

    def train(self)-> float:
        V = np.arange(0, len(self.adj_matrix))
        for _ in range(self.walks_per_vertex):
            # shuffle vertex
            np.random.shuffle(V)
            for start_node in V:
                # random walk
                W = self._biased_walk(start_node)
                # skip-gram
                self._skip_gram_train(W)
             # consider epoch as if all node be start_node
             # = consider epoch as walks_per_vertex
            self.epoch_loss.append(self.loss)
            self.loss = 0.0
        return self.epoch_loss

    def show_embedding(self):
        return self.w1, self.w2
```

Figure 4: node2vec code