Deep Learning and Data Science (CSE5851) **Assignment9**

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

May 6, 2021

1. (100 points) [LINE with Negative Sampling] Find embedding vectors of vertices in a given network by using optimization based on the 1st-order and 2nd-order proximities, the so-called LINE. 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,
- the number of negative samples : 5

which can however be replaced by other ones if another setting leads to a better result. You may set other hyperparameters arbitrarily. Use Negative Sampling to approximate the probability distribution. To show the convergence, plot the loss versus the number of epochs using the above dataset. Additionally, plot all resulting vectors on the two-dimensional space.

(a) (30 points) Find embedding vectors via the 1st-order proximity.

(b) (70 points) Find embedding vectors via the 2nd-order proximity. Make discussions on how the vertices are embedded in comparison with the case of the 1st-order proximity.

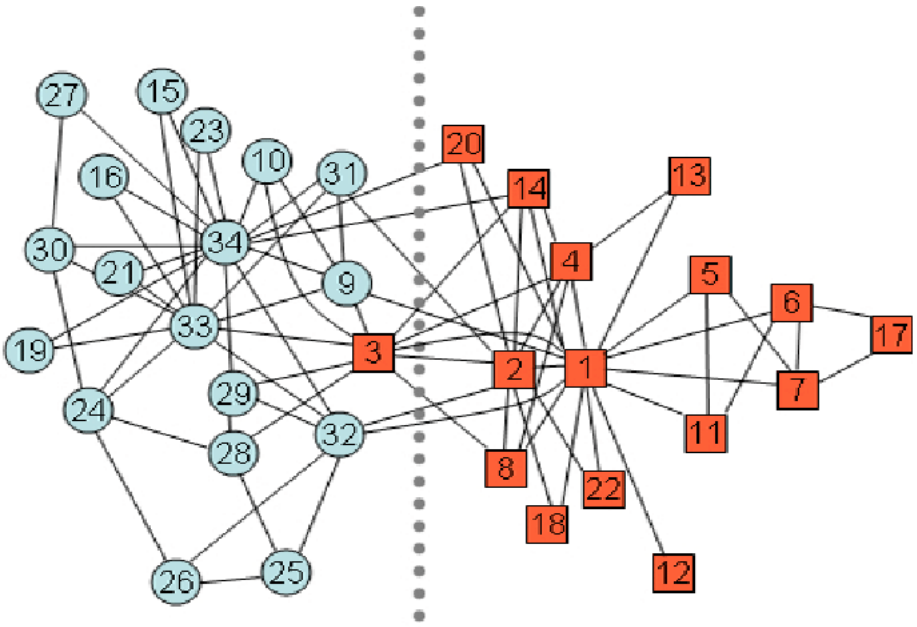

Figure 1: karate network

- First order proximity loss plot

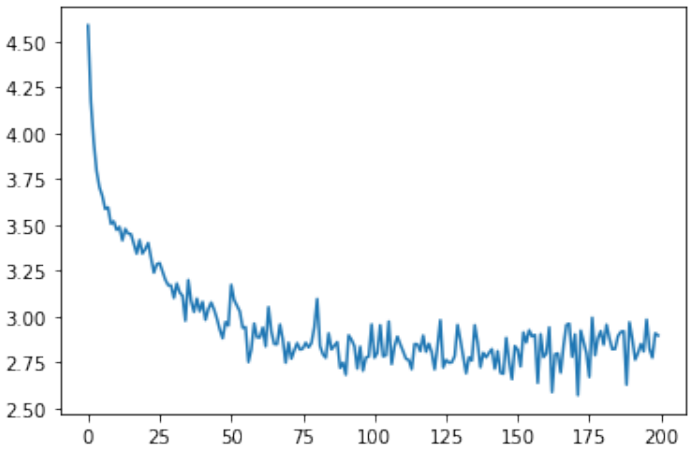

Figure 2: First order proximity loss plot

- First order proximity embedding
  - negative sampling을 사용할 때, 서로 연결되어 있지 않은 노드와는 멀어지게 업데이트가 되기 때문에 이전에 negative sampling을 사용하지 않는 first order proximity경우보다 classifiaction측면에서는 더 임베딩이 잘 되는 것 같다.

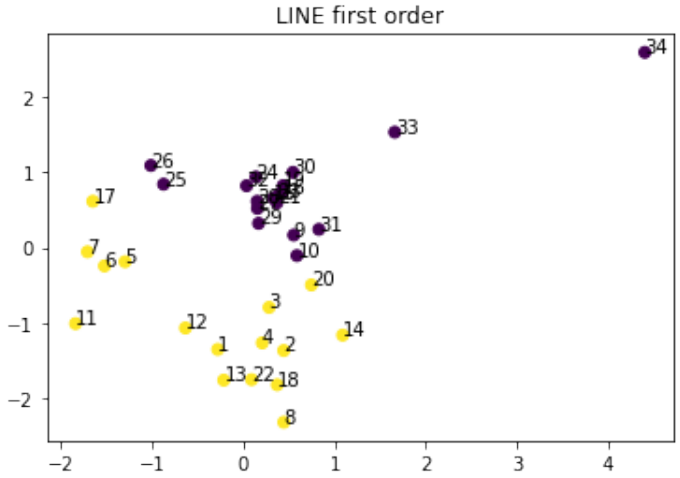

Figure 3: First order proximity embedding

- Second order proximity loss plot

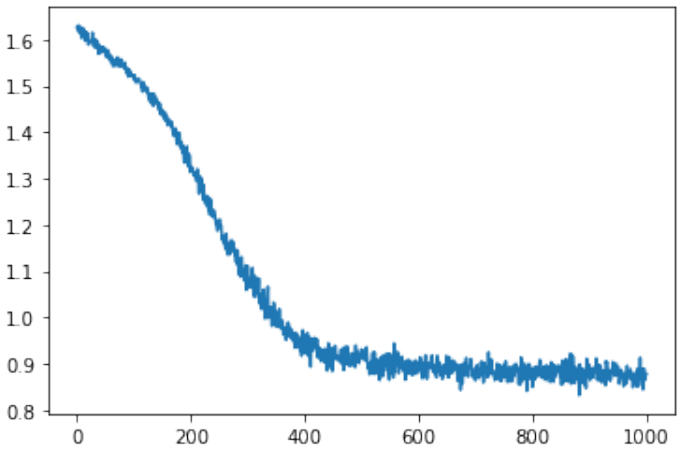

Figure 4: Second order proximity loss plot

- Second order proximity embedding
  - 1번 node는 노란색 집단의 가장 중심에 있다. 노란색 집단과 가장 많이 연결이 되어있어서 청색집단과 멀게 임베딩된 것이다.
  - 반대로 34번, 33번 node는 청색집단의 중심에 있어서 노란색 집단과 멀리 임베딩이 된 것으로 보인다.
  - 3번 node의 경우 실제 Figure1을 보면 노란색과 청색 집단 사이에 위치한 모습을 볼 수 있는데 이 또한 임베딩벡터에서 나름 잘 표현된 것으로 보인다.
  - first order proximity보다 조금 더 섬세한 관계까지 고려할 수 있지 않았나 싶다. 모든 network data에서는 아니지만 karate network의 경우 neighbor의 관계가 중요하기 때문에 second order proximity가 더 적절하다고 생각한다.
  - 모든 context를 고려하지 않았음에도 좋은 성능을 낼 수 있다는 것을 확인했다.

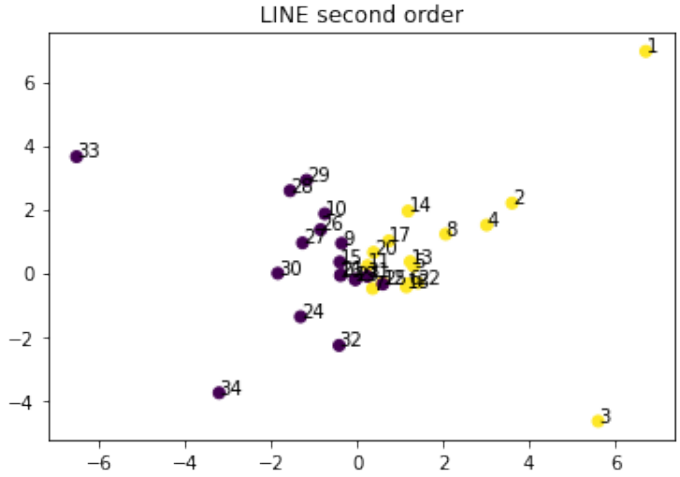

Figure 5: Second order proximity embedding

```python
class LINE_first_negS:
    def __init__(self,
                 adj_matrix,
                 embedding_dim=2,
                 learning_rate=0.02,
                 num_negetive_sample=5):

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

        self.epoch_loss = 0.0

    def _sigmoid(self, a: float)-> float :
        p = 1 / (1+np.exp(-a))
        return p

    def SGD_optim(self):
        self.epoch_loss = 0.0
        n = len(self.adj_matrix)
        total_iter = np.sum(adj_matrix) / 2
        for i in range(0,n):
            for j in range(i+1, n):
                if self.adj_matrix[i,j] == 1:
                    p1_pos = self._sigmoid(np.sum(self.w[i]*self.w[j]))
                    self.epoch_loss += - np.log(p1_pos) / total_iter
                    #여기서 i과 연결안된 node들 self.k개 sampling
                    neg_sample_idx = np.where(adj_matrix[i]==0)[0] # negative sample node's index
                    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)
                    p1_negS_grad = np.zeros(self.embedding_dim)
                    for neg_idx in neg_samples:
                        self.epoch_loss += -np.log(self._sigmoid(-
                                                   np.sum(self.w[i]*self.w[neg_idx]))+0.05) / total_iter
                        p1_negS_grad += self._sigmoid(np.sum(self.w[i]*self.w[neg_idx]))\
                                        * self.w[neg_idx]
                    # update
                    self.w[i] -= self.learning_rate * (p1_pos-1)* self.w[j]
                    self.w[i] -= self.learning_rate * p1_negS_grad
                    self.w[j] -= self.learning_rate * (p1_pos-1) * self.w[i]
        return self.epoch_loss

    def show_embedding(self):
        return self.w
```

Figure 6: first order code

```python
class LINE_second_negS:
    def __init__(self,
                 adj_matrix,
                 embedding_dim=2,
                 learning_rate=0.02,
                 num_negetive_sample=5):

        self.adj_matrix = adj_matrix
        self.embedding_dim = embedding_dim
        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.w = np.random.rand(len(adj_matrix),embedding_dim)
        self.w_context = np.random.rand(len(adj_matrix),embedding_dim)

        self.epoch_loss = 0.0

    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 SGD_optim(self):
        self.epoch_loss = 0.0
        n = len(self.adj_matrix)
        total_iter = np.sum(adj_matrix)
        for i in range(0,n):
            for j in range(0, n):
                if self.adj_matrix[i,j] == 1:
                    # 여기서 i과 연결안된 node를 self.k개 sampling
                    neg_sample_idx = np.where(adj_matrix[i]==0)[0] # negative sample node's index
                    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)
                    # 모든 context vector를 사용하지않고 negative samples만 사용
                    # update
                    p2_j_given_i = np.exp(self.w[i] @ self.w_context[j]) \
                    / np.sum(np.exp(self.w_context[neg_samples] @ self.w[i]))
                    ## w gradient calculation
                    w_i_grad = (-self.w_context[j] +\
                                np.exp(self.w_context[neg_samples] @ self.w[i]) @\
                                self.w_context[neg_samples]/\
                                np.sum(np.exp(self.w_context[neg_samples] @ self.w[i])))
                    ## context vector update
                    p2_j_given_i_vec = np.exp(self.w_context[neg_samples] @ self.w[i]) /\
                        np.sum(np.exp(self.w_context[neg_samples] @ self.w[i]))
                    self.w_context[neg_samples] -= self.learning_rate * np.diag(p2_j_given_i_vec) @\
                    np.tile(self.w[i], self.k).reshape(self.k,-1)
                    self.w_context[j] -= self.learning_rate * (-(1-p2_j_given_i)*self.w[i])
                    ## w update
                    self.w[i] -= self.learning_rate * w_i_grad
                    ## loss
                    self.epoch_loss += - np.log(p2_j_given_i) / total_iter
        return self.epoch_loss

    def show_embedding(self):
        return self.w
```

Figure 7: second order code

```python
line_first_order = LINE_first_negS(adj_matrix, embedding_dim=2, learning_rate=0.02, num_negetive_sample=5)
epoch_losses = []

n_epochs = 200
for epoch in range(1, n_epochs+1):
    epoch_loss = line_first_order.SGD_optim()
    epoch_losses.append(epoch_loss)
    if epoch % 10 == 0:
        print(f'Epoch = {epoch} : loss = {epoch_loss:.5f}')
plt.plot(epoch_losses)
plt.show()

line_first_emb = line_first_order.show_embedding()

plt.title('LINE first order')
plt.scatter(line_first_emb[:,0],line_first_emb[:,1], c=list(map(int, label[:,1])))
for i in range(0, 34):
        plt.text(float(line_first_emb[i,0]), float(line_first_emb[i,1]), i+1 , fontsize=10)
plt.show()

line_second_order = LINE_second_negS(adj_matrix, embedding_dim=2, learning_rate=0.002, num_negetive_sample=5)
epoch_losses = []

n_epochs = 1000
for epoch in range(1, n_epochs+1):
    epoch_loss = line_second_order.SGD_optim()
    epoch_losses.append(epoch_loss)
    if epoch % 100 == 0:
        print(f'Epoch = {epoch} : loss = {epoch_loss:.5f}')
plt.plot(epoch_losses)
plt.show()

line_second_emb = line_second_order.show_embedding()

plt.title('LINE second order')
plt.scatter(line_second_emb[:,0],line_second_emb[:,1], c=list(map(int, label[:,1])))
for i in range(0, 34):
        plt.text(float(line_second_emb[i,0]), float(line_second_emb[i,1]), i+1 , fontsize=10)
plt.show()
```

Figure 8: loss, plot implementation code