# Deep Learning and Data Science (CSE5851) **Assignment8**

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

April 29, 2021

1. (100 points) [LINE with No Approximation] 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,

which can however be replaced by other ones if another setting leads to a better result. You may set other hyperparameters arbitrarily. Do NOT use any approximation techniques such as negative sampling to compute 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 twodimensional 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.

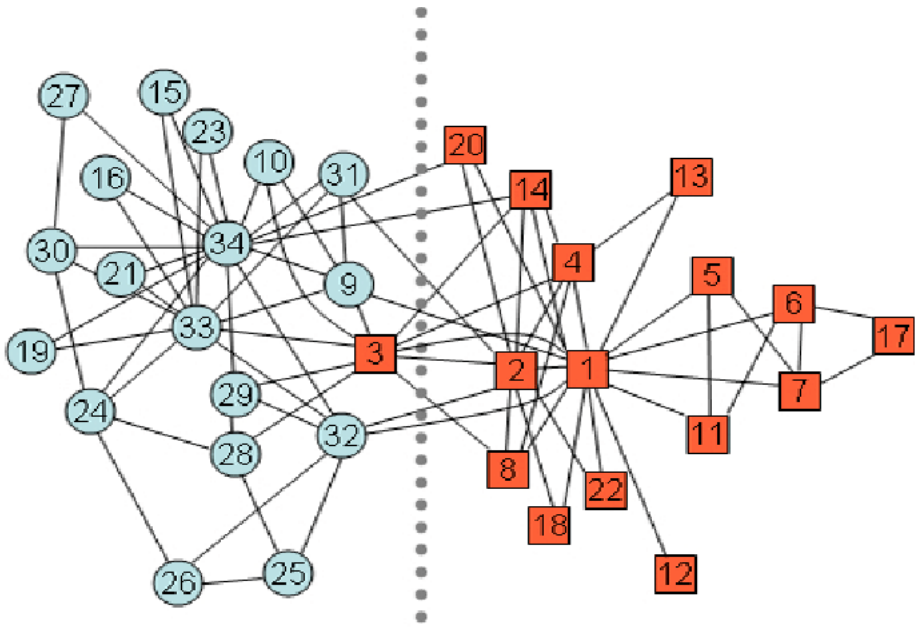


Figure 1: karate network

- First order proximity loss plot

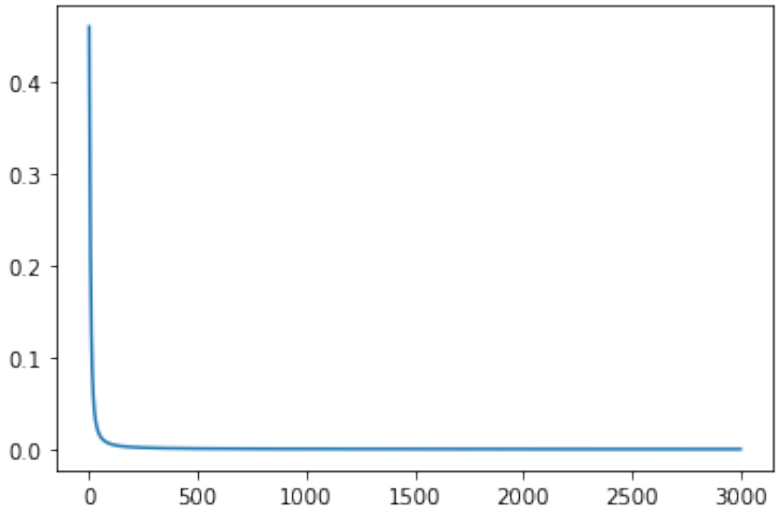


Figure 2: First order proximity loss plot

- First order proximity embedding
  - Figure1을 보면 대표적으로 node 6,7,17가 바로 이어져있는 집단인데 아래 embedding vector를 통해서도 이를 확인할 수 있다.
  - 하지만 classification에 이용한다고 생각했을 때, 성능이 그렇게 좋은지는 모르겠다.
  - Figure1에서 알 수 있듯이 karate network는 서로 다른 class끼리도 directly 연결된 edge가 꽤 있기 때문에 first embedding 결과를 통해서 classification을 하기에는 무리가 있어 보인다.

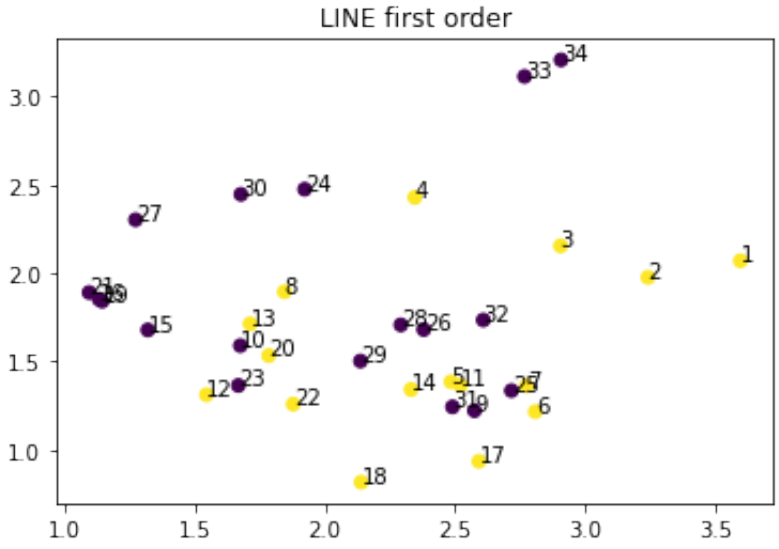


Figure 3: First order proximity embedding

- Second order proximity loss plot

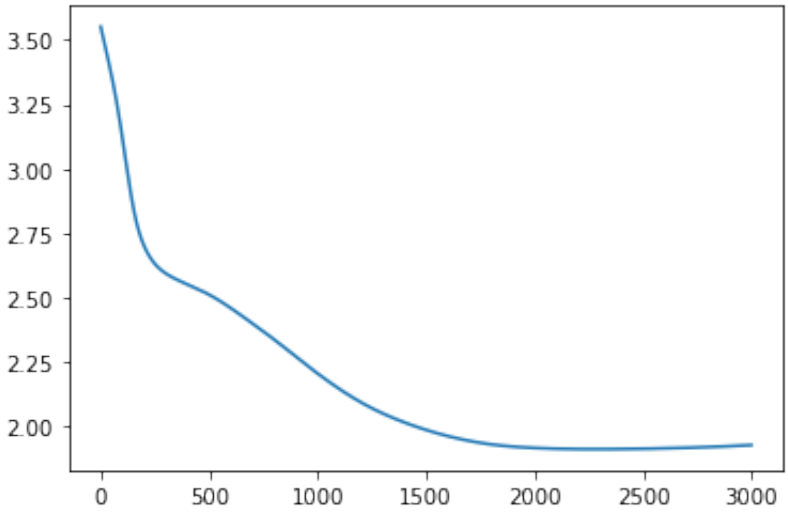


Figure 4: Second order proximity loss plot

- Second order proximity embedding
  - Figure1과 비교했을 때, 바로 매치되지는 않는다. 바로 이어지는 edge를 고려하지 않기 때문인 것 같다.
  - first order의 경우보다 class에 따른 집단형성이 더 잘 된 것 같다.
  - first와 second order를 동시에 이용하면 더 좋은 결과를 얻을 수 있지 않을까 생각이 든다.

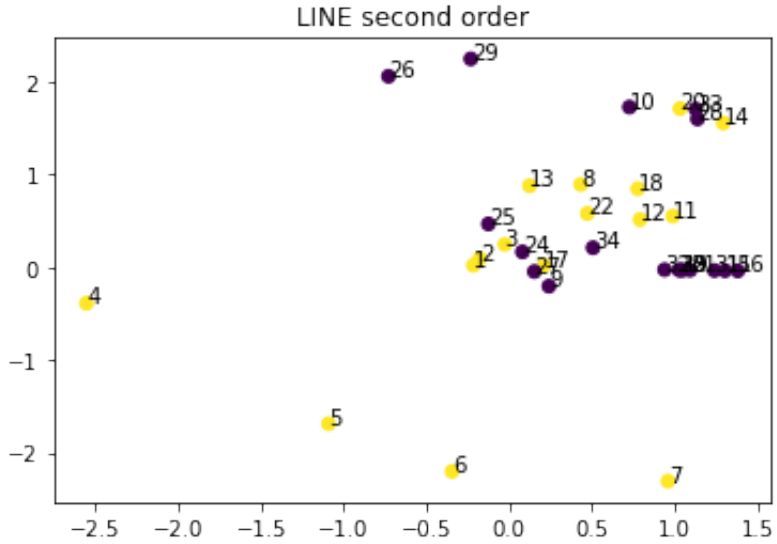


Figure 5: Second order proximity embedding

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

        self.adj_matrix = adj_matrix
        self.embedding_dim = embedding_dim
        self.learning_rate = learning_rate

        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)-> float:
        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 = self._sigmoid(np.sum(self.w[i]*self.w[j]))
                    self.epoch_loss += - np.log(p1) / total_iter
                    self.w[i] += self.learning_rate * (1-p1) * self.w[j]
                    self.w[j] += self.learning_rate * (1-p1) * self.w[i]
        return self.epoch_loss

    def show_embedding(self)-> np.array:
        return self.w

# define model
line_first_order = LINE_first(adj_matrix, embedding_dim=2, learning_rate=0.05)
epoch_losses = []
# train
n_epochs = 3000
for epoch in range(1, n_epochs+1):
    epoch_loss = line_first_order.SGD_optim()
    epoch_losses.append(epoch_loss)
    if epoch % 100 == 0:
        print(f'Epoch = {epoch} : loss = {epoch_loss:.5f}')
# plot loss
plt.plot(epoch_losses)
# get embedding
line_first_emb = line_first_order.show_embedding()
# plot 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()
```

Figure 6: first order code

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

        self.adj_matrix = adj_matrix
        self.embedding_dim = embedding_dim
        self.learning_rate = learning_rate

        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) / 2
        for i in range(0,n):
            for j in range(i+1, n):
                if self.adj_matrix[i,j] == 1:
                    p2_j_given_i = np.exp(self.w[i] @ self.w_context[j]) \
                    / np.sum(np.exp(self.w_context @ self.w[i]))
                    self.epoch_loss += - np.log(p2_j_given_i) / total_iter
                    w_i_grad = (-self.w_context[j] +\
                                np.exp(self.w_context @ self.w[i]) @ self.w_context/\
                                np.sum(np.exp(self.w_context @ self.w[i])))
                    p2_j_given_i_vec = self.w_context @ self.w[i] /\
                                       np.sum(np.exp(self.w_context @ self.w[i]))
                    self.w_context -= self.learning_rate * np.diag(p2_j_given_i_vec) @\
                                      np.tile(self.w[i], n).reshape(n,-1)
                    self.w_context[j] += self.learning_rate * p2_j_given_i_vec[j] * self.w[i]
                    self.w_context[j] -= self.learning_rate * (-(1-p2_j_given_i)*self.w[i])
                    self.w[i] -= self.learning_rate * w_i_grad
        return self.epoch_loss

    def show_embedding(self):
        return self.w

# define model
line_second_order = LINE_second(adj_matrix, embedding_dim=2, learning_rate=0.002)
epoch_losses = []
# train
n_epochs = 3000
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}')
# plot loss
plt.plot(epoch_losses)
# get embedding
line_second_emb = line_second_order.show_embedding()
# plot 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 7: second order code