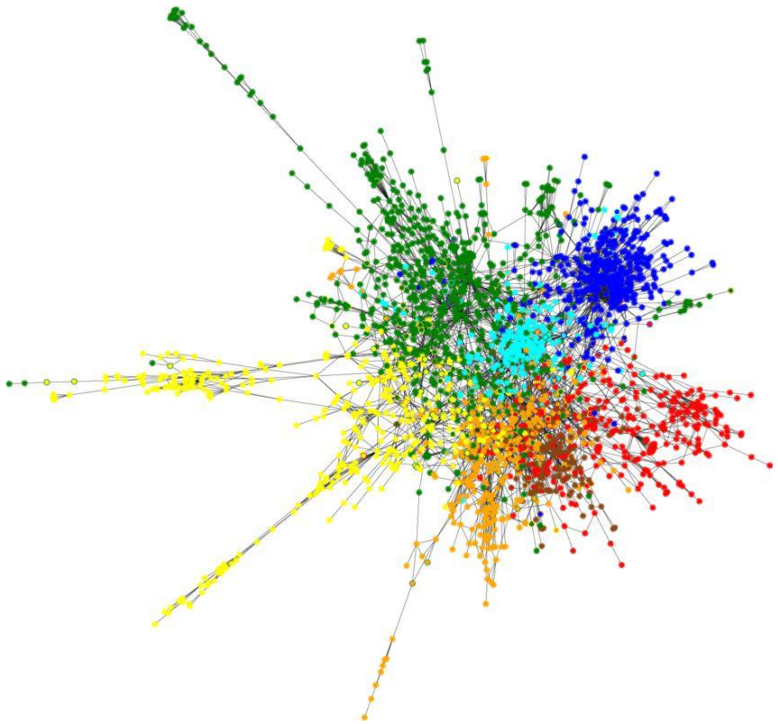
Node Classification in Citation Graph Dataset



Applied Machine Learning Project
20201181 Jihwan Oh
2023.06.08

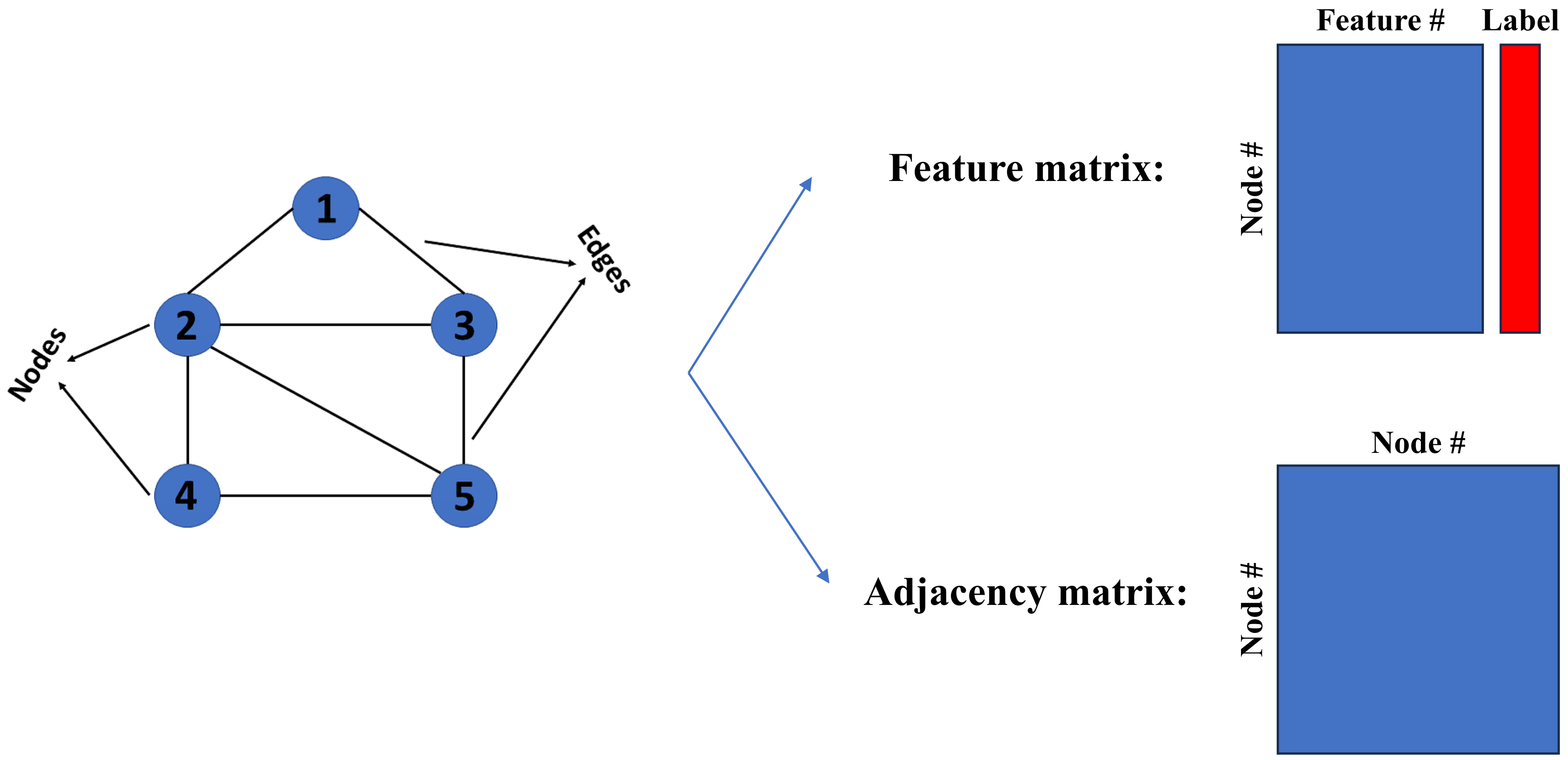
Contents

1. Introduction

Introduction

- **Our objective**: Node classification in citation graph dataset.

What is **graph data**?



Introduction

- **Our objective**: Node classification in citation graph dataset.

What is **graph data**? + What is **citation graph data?**

- We will use two citation graph data: **Citeseer dataset** and **Cora dataset.**
- **Citeseer/Cora dataset** : Nodes mean scientific publications and edges mean citation relationships.

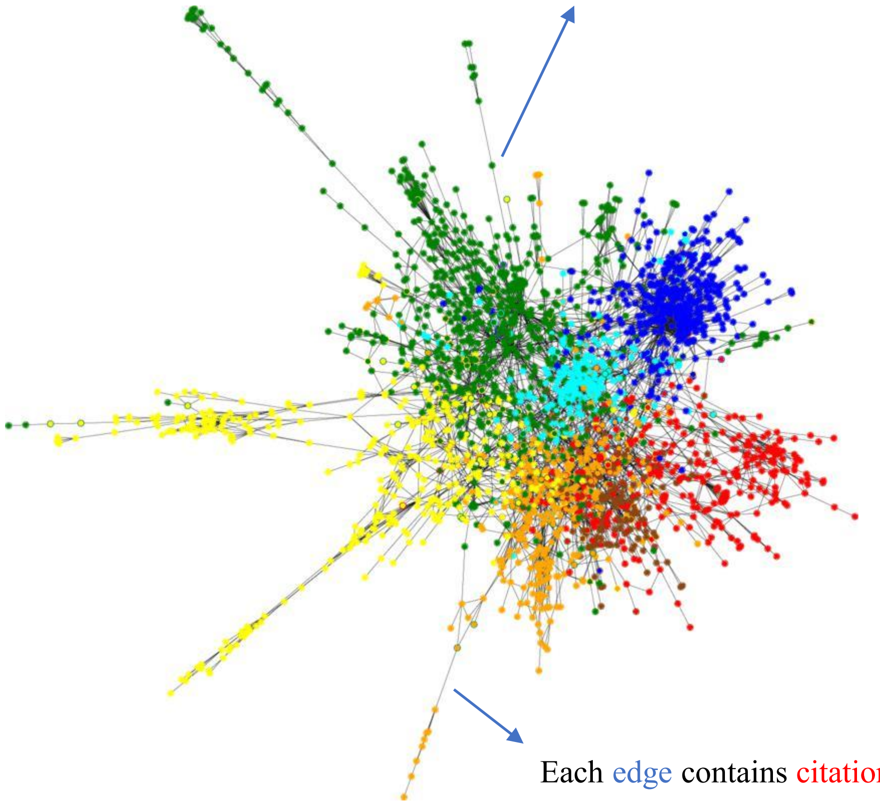
Citeseer Dataset:

```
citeseer = CitationGraphDataset('citeseer')

NumNodes: 3327
NumEdges: 9228
NumFeats: 3703
NumClasses: 6
NumTrainingSamples: 120
NumValidationSamples: 500
NumTestSamples: 1000
```

Cora Dataset:

```
cora = CoraGraphDataset('cora')

NumNodes: 2708
NumEdges: 10556
NumFeats: 1433
NumClasses: 7
NumTrainingSamples: 140
NumValidationSamples: 500
NumTestSamples: 1000
```

Each node contains paper feature and label



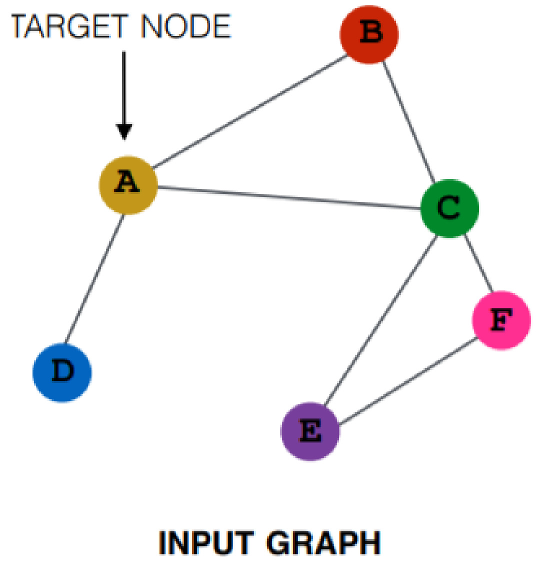
Each edge contains citation relationships

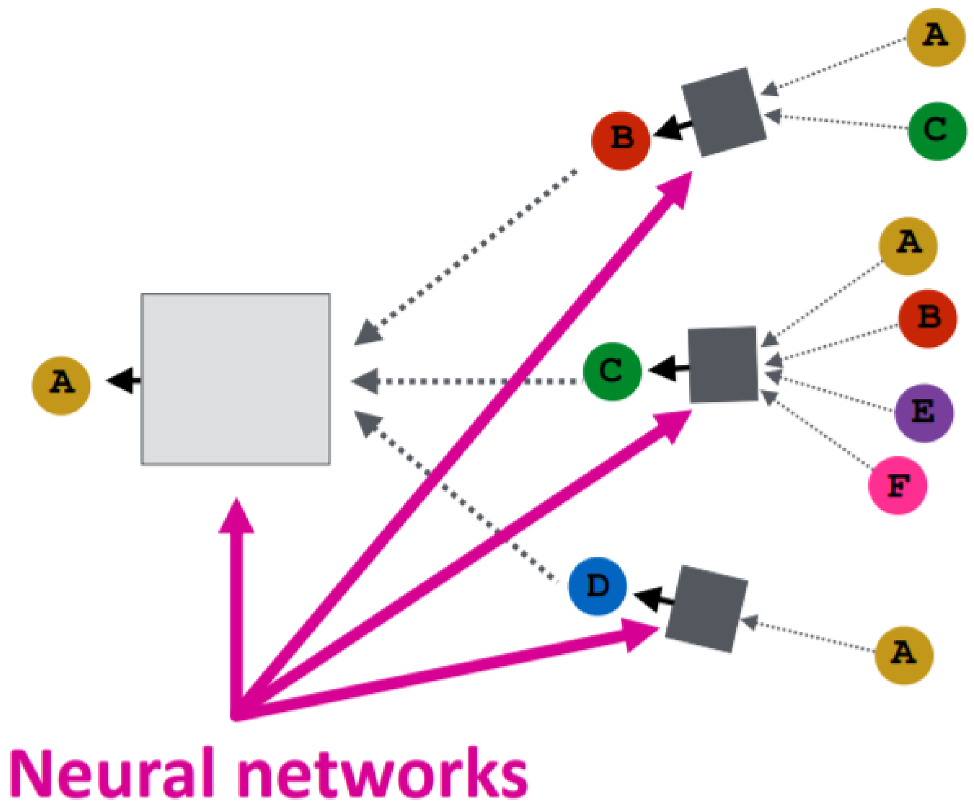
Introduction

- **Our objective**: Node classification in citation graph dataset.

What is **graph data**?

Why **GNN (Graph Neural Network)?**





Introduction

- **Our objective**: Node classification in citation graph dataset.

What is **graph data**?

Why **GNN (Graph Neural Network)?**

We will use state-of-art GNN : **GrpahCL**
It is introduced by 'Graph Contrastive Learning with Augmentations' (2020 NIPS)

Graph Contrastive Learning with Augmentations

Yuning You[1*], Tianlong Chen[2*], Yongduo Sui[3], Ting Chen[4], Zhangyang Wang[2], Yang Shen[1]
[1]Texas A&M University, [2]University of Texas at Austin,
[3]University of Science and Technology of China, [4]Google Research, Brain Team
{yuning.you,yshen}@tamu.edu, {tianlong.chen,atlaswang}@utexas.edu
syd2019@mail.ustc.edu.cn, iamtingchen@google.com

1. We will study the model **GraphCL** in paper briefly.
2. Then, implement the model in real world **citation graph data**.
3. Lastly, we check the **classification accuracy** of our model.

2. Model : GraphCL

Method : Data Augmentation for Graph & Graph Contrastive Learning

Table 1: Overview of data augmentations for graphs.

Data augmentation	**Type**	**Underlying Prior**
Node dropping	Nodes, edges	Vertex missing does not alter semantics.
Edge perturbation	Edges	Semantic robustness against connectivity variations.
Attribute masking	Nodes	Semantic robustness against losing partial attributes.
Subgraph	Nodes, edges	Local structure can hint the full semantics.

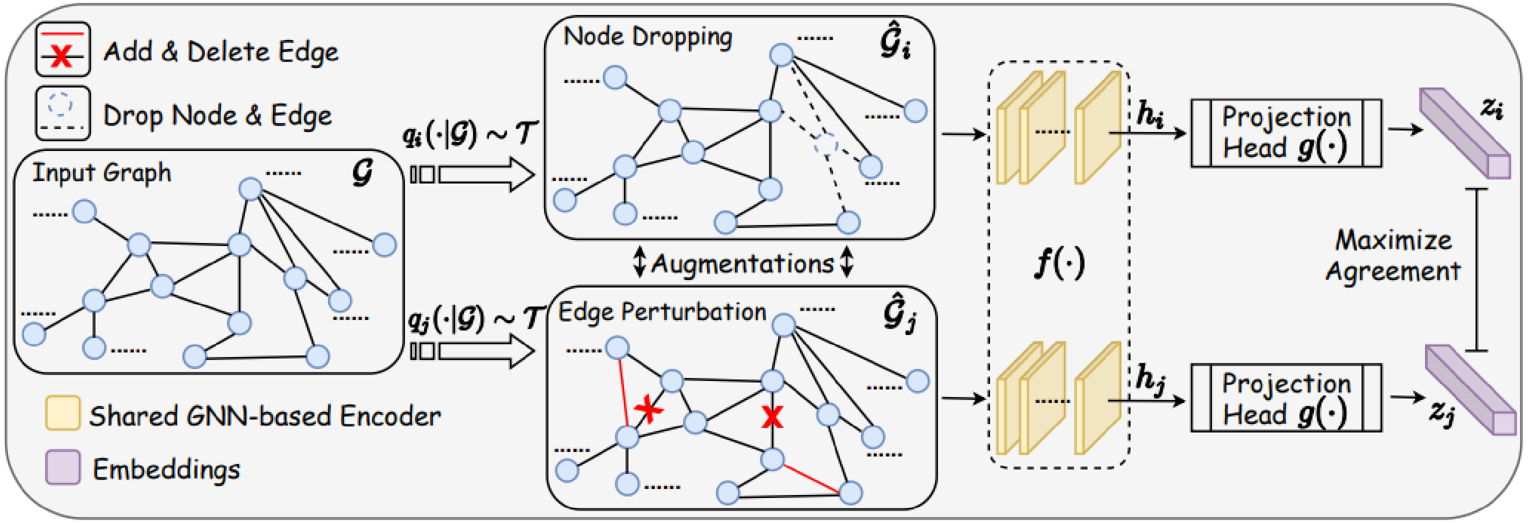
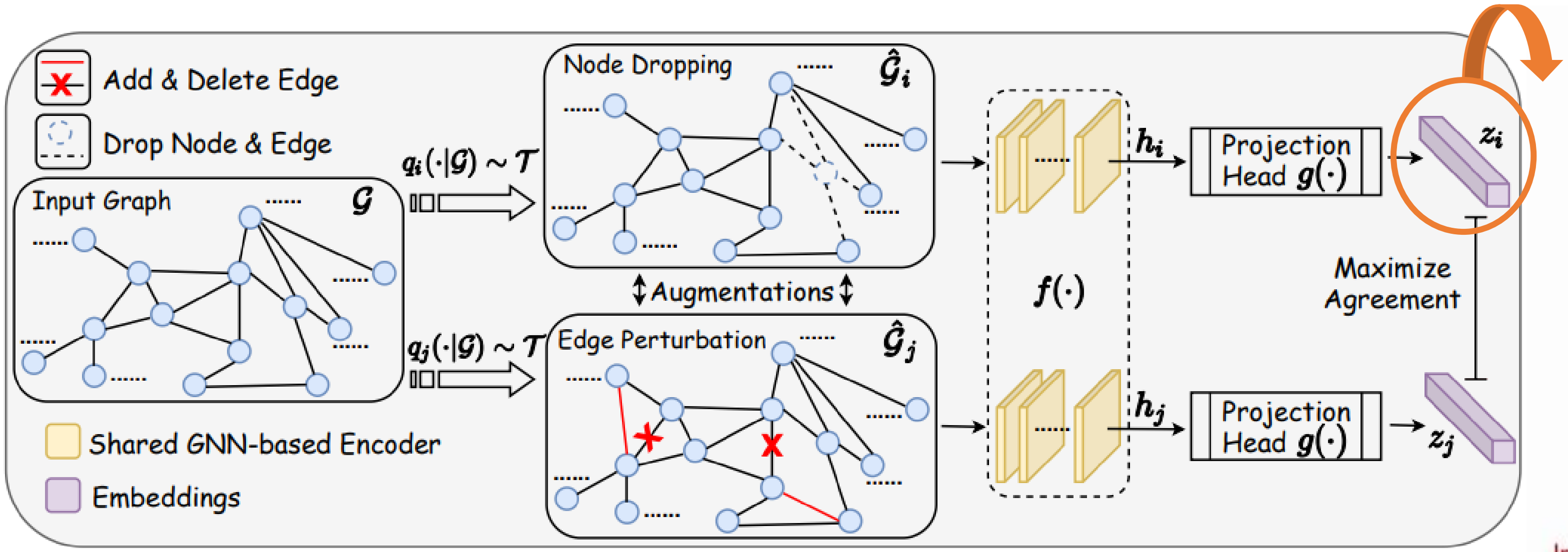


Figure 1: A framework of graph contrastive learning. Two graph augmentations $q_i(\cdot|\mathcal{G})$ and $q_j(\cdot|\mathcal{G})$ are sampled from an augmentation pool $\mathcal{T}$ and applied to input graph $\mathcal{G}$. A shared GNN-based encoder $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize the agreement between representations z_i and z_j via a contrastive loss.

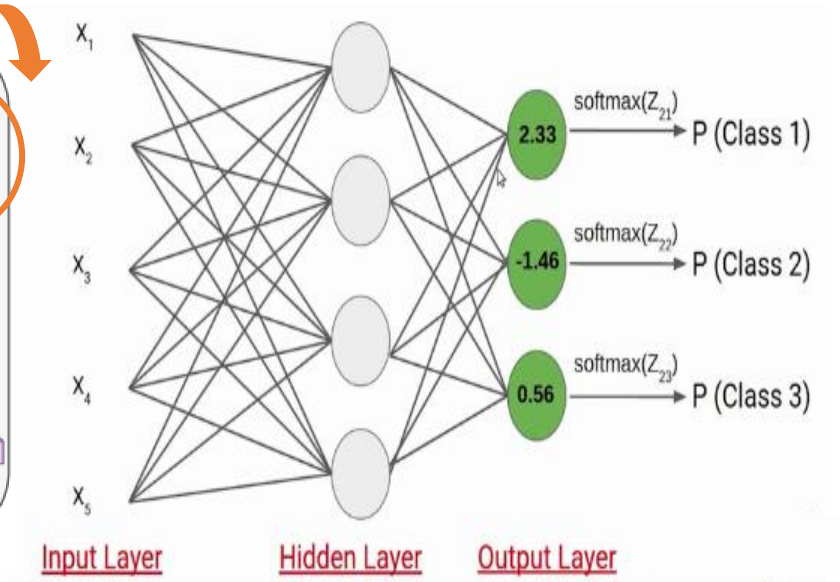
$$\ell_n = -\log \frac{\exp(\mathrm{sim}(\boldsymbol{z}_{n,i}, \boldsymbol{z}_{n,j})/\tau)}{\sum_{n'=1, n' \neq n}^{N} \exp(\mathrm{sim}(\boldsymbol{z}_{n,i}, \boldsymbol{z}_{n',j})/\tau)}, \tag{3}$$

Adopt Softmax regression for evaluation

- We can’t check the performance of our unsupervised model in real labeled data.
- We should adopt simple **softmax regression**.
    - Inputs are final representation vectors (z_i) of our unsupervised model.
    - They contain the information of each node.
    - We can check our unsupervised model performance through simple supervised learning



GraphCL
(Unsupervised)



Softmax regression
(Supervised)

3. Implementation Process

Table 1: Overview of data augmentations for graphs.

Data augmentation	Type	Underlying Prior
Node dropping	Nodes, edges	Vertex missing does not alter semantics.
Edge perturbation	Edges	Semantic robustness against connectivity variations.
Attribute masking	Nodes	Semantic robustness against losing partial attributes.
Subgraph	Nodes, edges	Local structure can hint the full semantics.

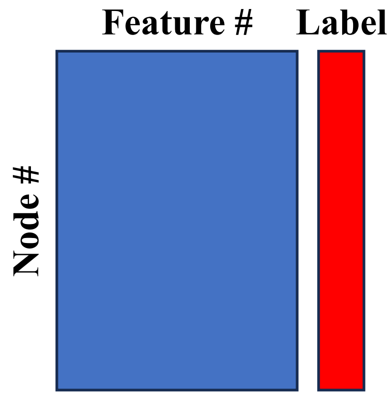
1. **Node drop**: Dropping node randomly.
   → Node # changed randomly
   → **Feature matrix** and **Adjacency matrix** are changed both

2. **Edge perturbation**: Adding or Dropping edge randomly.
   → **Adjacency matrix** is changed

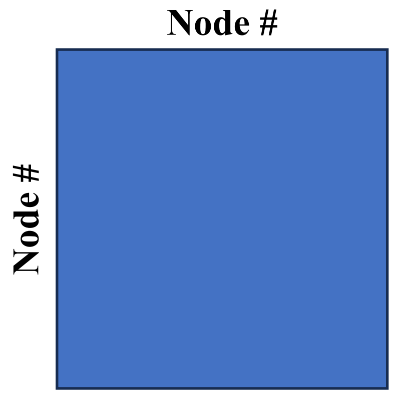
3. **Attribute masking**: Dropping feature randomly.
   → **Feature matrix** is changed

4. **Subgraph**: Sampling the subgraph from original data.
   → Node # changed strategically.
   → **Feature matrix and Adjacency matrix** are changed both

Feature matrix:



Adjacency matrix:



Edge perturbation

```python
def aug_random_edge(input_adj, drop_percent=0.2):

    percent = drop_percent / 2
    row_idx, col_idx = input_adj.nonzero()

    index_list = []
    for i in range(len(row_idx)):
        index_list.append((row_idx[i], col_idx[i]))

    single_index_list = []
    for i in list(index_list):
        single_index_list.append(i)
        index_list.remove((i[1], i[0]))


    edge_num = int(len(row_idx) / 2)      # 9228 / 2
    add_drop_num = int(edge_num * percent / 2)
    aug_adj = copy.deepcopy(input_adj.todense().tolist())

    edge_idx = [i for i in range(edge_num)]
    drop_idx = random.sample(edge_idx, add_drop_num)


    for i in drop_idx:
        aug_adj[single_index_list[i][0]][single_index_list[i][1]] = 0
        aug_adj[single_index_list[i][1]][single_index_list[i][0]] = 0

    '''
    above finish drop edges
    '''
    node_num = input_adj.shape[0]
    l = [(i, j) for i in range(node_num) for j in range(i)]
    add_list = random.sample(l, add_drop_num)

    for i in add_list:

        aug_adj[i[0]][i[1]] = 1
        aug_adj[i[1]][i[0]] = 1

    aug_adj = np.matrix(aug_adj)
    aug_adj = sp.csr_matrix(aug_adj)
    return aug_adj
```

Drop Node

```python
def aug_drop_node(input_fea, input_adj, drop_percent=0.2):

    input_adj = torch.tensor(input_adj.todense().tolist())
    input_fea = input_fea.squeeze(0)

    node_num = input_fea.shape[0]
    drop_num = int(node_num * drop_percent)    # number of drop nodes
    all_node_list = [i for i in range(node_num)]

    drop_node_list = sorted(random.sample(all_node_list, drop_num))

    aug_input_fea = delete_row_col(input_fea, drop_node_list, only_row=True)
    aug_input_adj = delete_row_col(input_adj, drop_node_list)

    aug_input_fea = aug_input_fea.unsqueeze(0)
    aug_input_adj = sp.csr_matrix(np.matrix(aug_input_adj))

    return aug_input_fea, aug_input_adj
```

Attribute masking

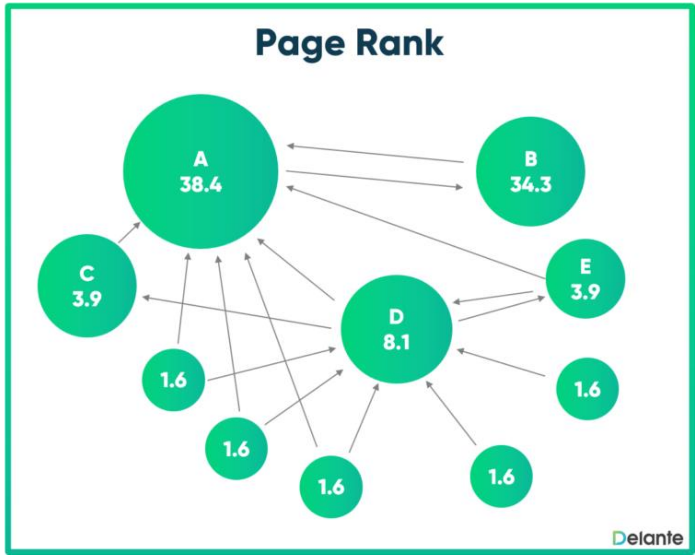
```
def aug_random_mask(input_feature, drop_percent=0.2):

    node_num = input_feature.shape[1]
    mask_num = int(node_num * drop_percent)
    node_idx = [i for i in range(node_num)]
    mask_idx = random.sample(node_idx, mask_num)
    aug_feature = copy.deepcopy(input_feature)
    zeros = torch.zeros_like(aug_feature[0][0])
    for j in mask_idx:
        aug_feature[0][j] = zeros
    return aug_feature
```

```
def delete_row_col(input_matrix, drop_list, only_row=False):

    remain_list = [i for i in range(input_matrix.shape[0]) if i not in drop_list]
    out = input_matrix[remain_list, :]
    if only_row:
        return out
    out = out[:, remain_list]

    return out
```

Subgraph

```
def aug_subgraph(input_fea, input_adj, drop_percent=0.2):

    input_adj = torch.tensor(input_adj.todense().tolist())
    input_fea = input_fea.squeeze(0)
    node_num = input_fea.shape[0]

    all_node_list = [i for i in range(node_num)]
    s_node_num = int(node_num * (1 - drop_percent))
    center_node_id = random.randint(0, node_num - 1)
    sub_node_id_list = [center_node_id]
    all_neighbor_list = []

    for i in range(s_node_num - 1):

        all_neighbor_list += torch.nonzero(input_adj[sub_node_id_list[i]], as_tuple=False).squeeze(1).tolist()

        all_neighbor_list = list(set(all_neighbor_list))
        new_neighbor_list = [n for n in all_neighbor_list if not n in sub_node_id_list]
        if len(new_neighbor_list) != 0:
            new_node = random.sample(new_neighbor_list, 1)[0]
            sub_node_id_list.append(new_node)
        else:
            break


    drop_node_list = sorted([i for i in all_node_list if not i in sub_node_id_list])

    aug_input_fea = delete_row_col(input_fea, drop_node_list, only_row=True)
    aug_input_adj = delete_row_col(input_adj, drop_node_list)

    aug_input_fea = aug_input_fea.unsqueeze(0)
    aug_input_adj = sp.csr_matrix(np.matrix(aug_input_adj))

    return aug_input_fea, aug_input_adj
```

Modified Subgraph :
(Adopt PageRank)




```python
#subgraph 수정
def aug_subgraph(input_fea, input_adj, drop_percent=0.2):

    matrix_D_array = input_adj.todense()
    matrix_D_array = np.nan_to_num(matrix_D_array)

    # L 에 대한 eigenvalue, eigenvector 계산
    eigen_value , eigen_vector = np.linalg.eig( matrix_D_array )

    # eigen value 내림차순 정렬
    order = np.absolute(eigen_value).argsort()[::-1]

    # 정렬 순서에 따라 재정렬
    eigen_value = eigen_value[order]
    eigen_vector = eigen_vector[:,order]

    # 첫번째 eigen value 에 대한 eigenvector 추출 및 비중확인
    r = eigen_vector[:,0]  # 0번째 열
    value = 100*np.real(r/np.sum(r)) ## np.real : 복소수 인수의 실수부를 반환
    print(value) #pagerank

    #------------------------------------------------------------------------------------------
    input_adj = torch.tensor(input_adj.todense().tolist())
    input_fea = input_fea.squeeze(0)
    node_num = input_fea.shape[0]

    all_node_list = [i for i in range(node_num)]
    s_node_num = int(node_num * (1 - drop_percent))
    center_node_id = value.tolist().index(max(value)) ### 수정됨
    #center_node_id = random.randint(0, node_num - 1) ### 오리지널
    sub_node_id_list = [center_node_id]
    all_neighbor_list = []

    for i in range(s_node_num - 1):

        all_neighbor_list += torch.nonzero(input_adj[sub_node_id_list[i]], as_tuple=False).squeeze(1).tolist()

        all_neighbor_list = list(set(all_neighbor_list))
        new_neighbor_list = [n for n in all_neighbor_list if not n in sub_node_id_list]
        if len(new_neighbor_list) != 0:
            new_node = random.sample(new_neighbor_list, 1)[0]
            sub_node_id_list.append(new_node)
        else:
            break
```

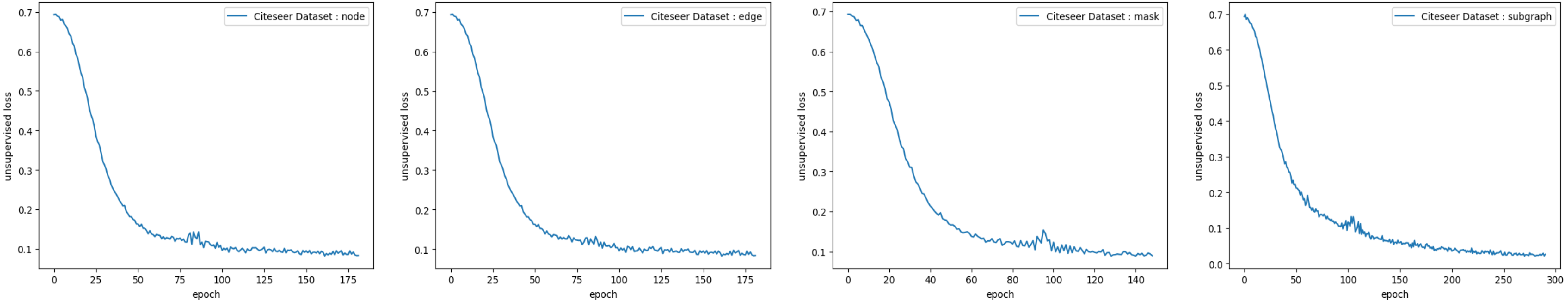
Derive PageRank
(Using eigenvector of
adjacency matrix)

Adopt maximum
PageRank to center node

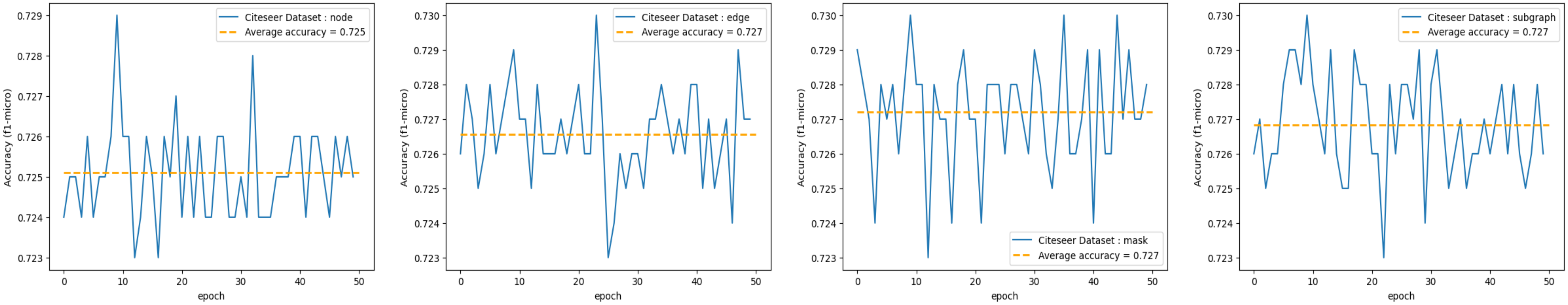
4. Implementation Result

1. Citeseer Dataset

GraphCL (Unsupervised Loss – based on DGI)

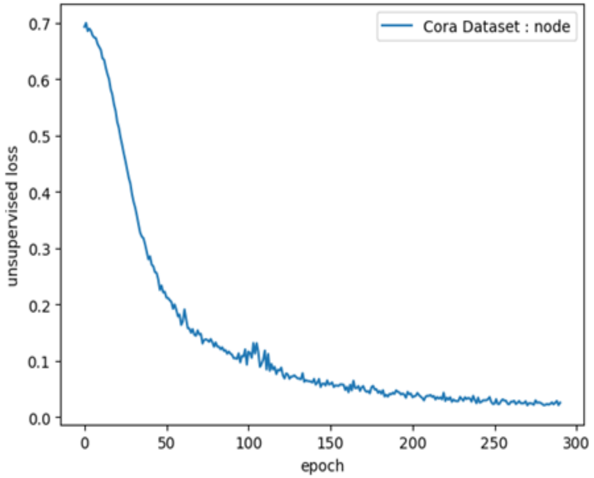


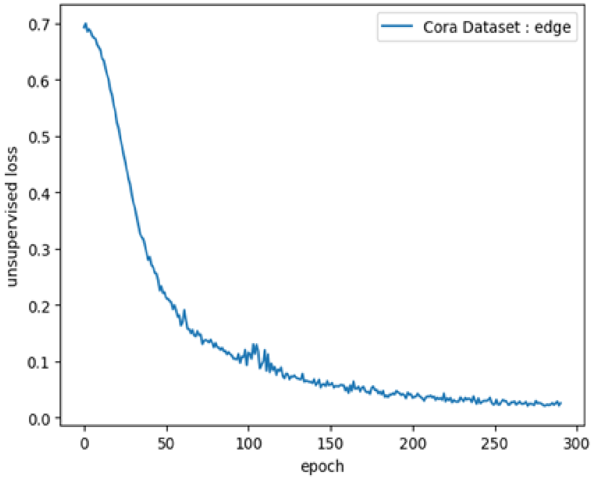
Softmax Regression (Supervised Loss)

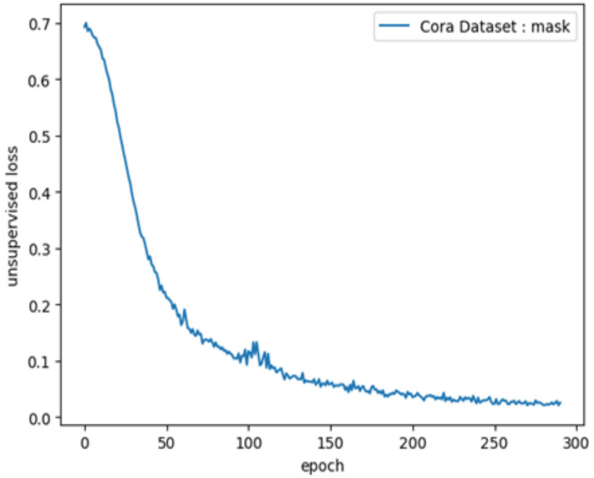


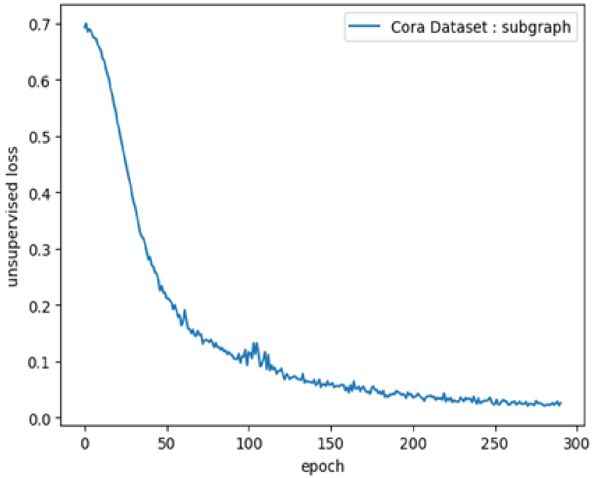
2. Cora Dataset

GraphCL (Unsupervised Loss – based on DGI)

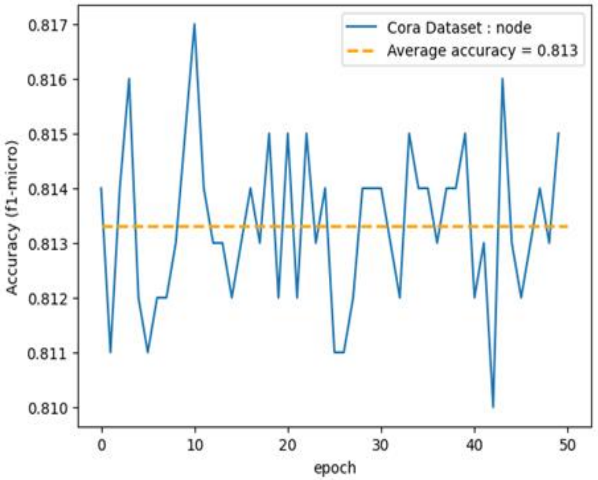


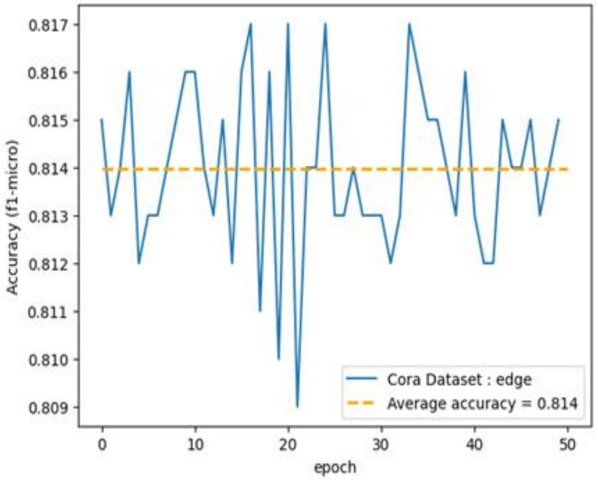


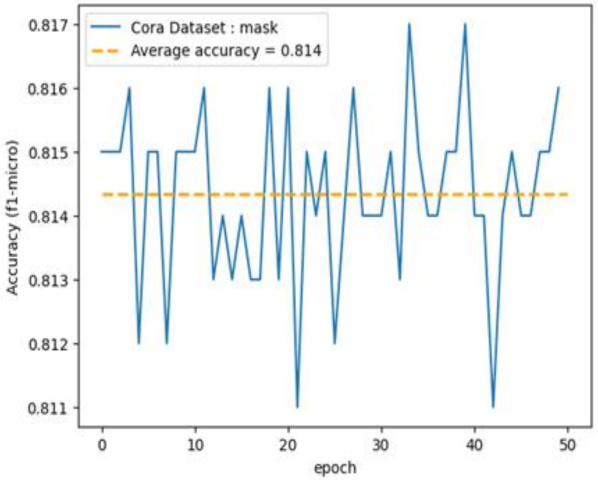


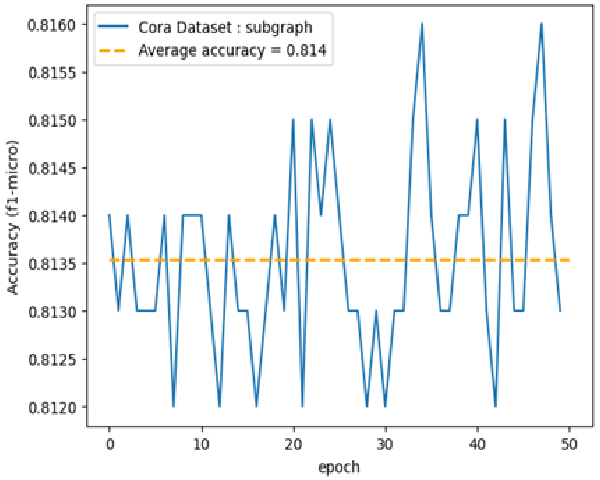


Softmax Regression (Supervised Loss)



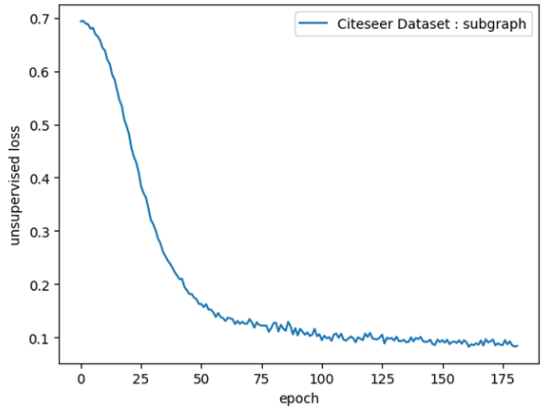


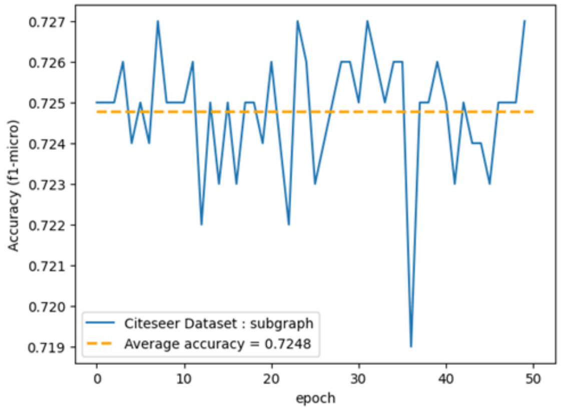




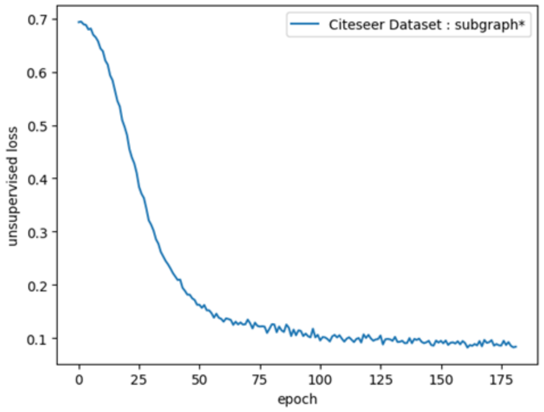
Citeseer dataset: Modified Subgraph

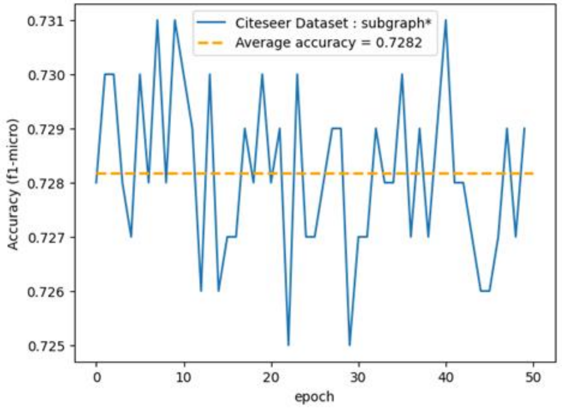
Random center node: (Original Model)



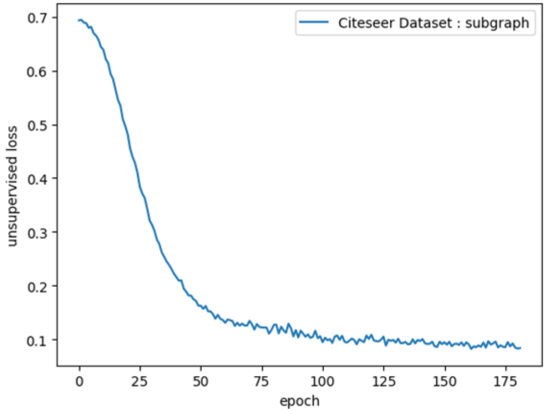


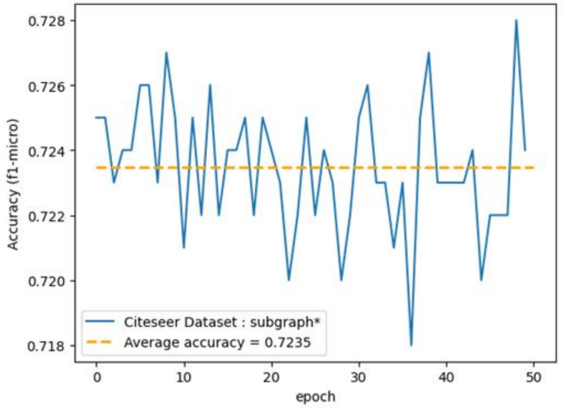
PageRank max center node:





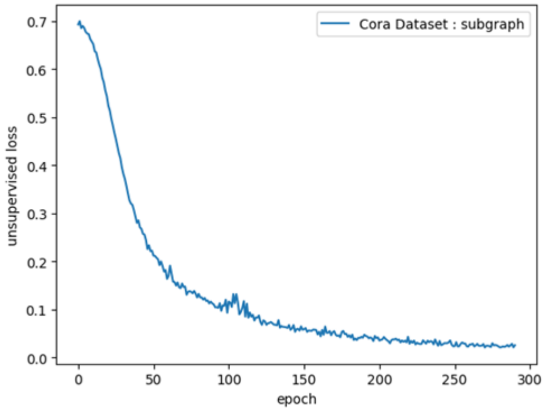
PageRank min center node:

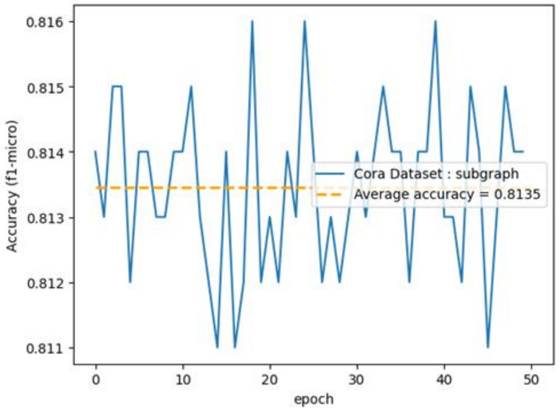




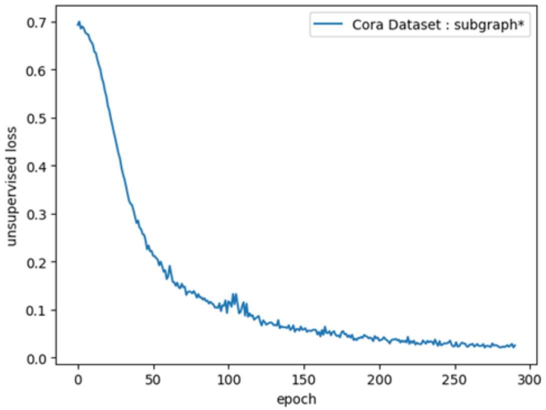
Cora dataset: Modified Subgraph

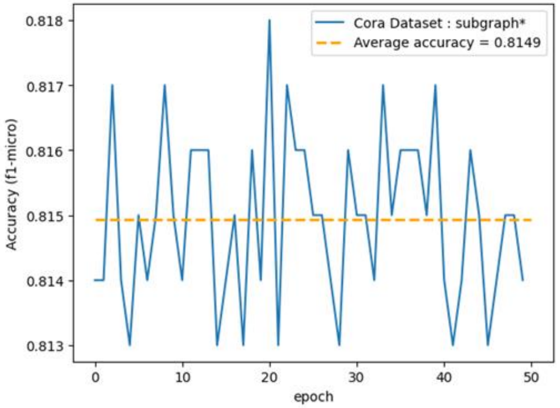
Random center node:
(Original Model)



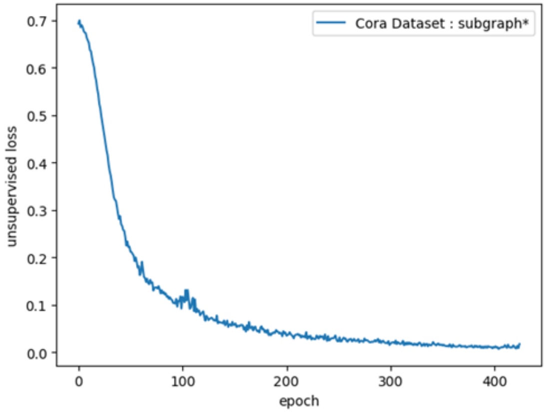


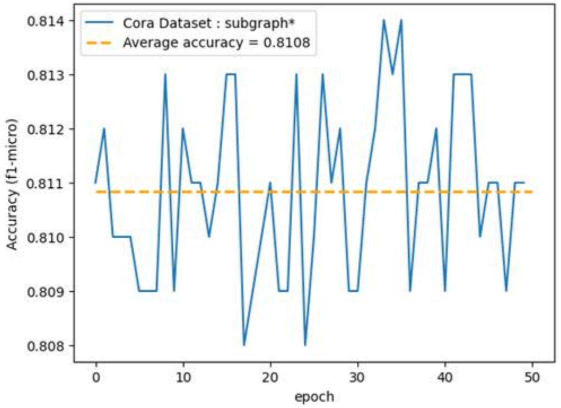
PageRank max center node:





PageRank min center node:





5. Conclusion

Conclusion

1. We simply identified the GraphCL unsupervised version.
2. We implementation the GraphCL model with 4 augment functions.
3. We classified node feature in citation dataset using GraphCL successfully.
4. We modify the subgraph function with PageRank algorithm.
5. PageRank algorithm seems to work quite well but needs experimentation with a larger dataset.

Thank you