Augmentation-Free Self-Supervised Learning on Graphs (AFGRL)

Namkyeong Lee, Junseok Lee, Chanyoung Park

Dept. of Industrial and Systems Engineering, KAIST, Daejeon, Republic of Korea

Graduate School of Artificial Intelligence, KAIST, Daejeon, Republic of Korea

Contents

1. Introduction – Self supervised learning

직접적인 supervision이 없는 데이터셋에서 스스로 supervision을 만들어 학습한다.

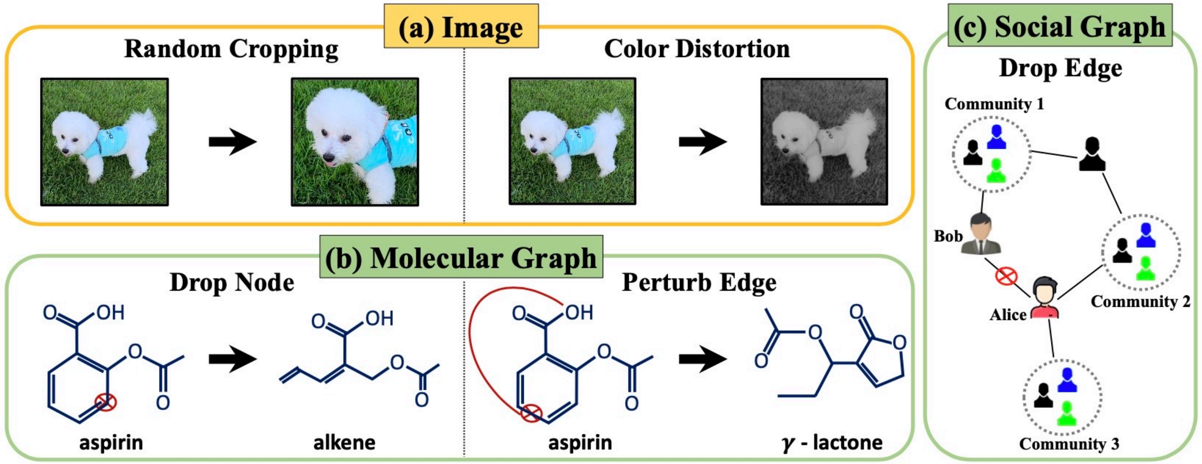
Pretext task + Contrastive Method

의미적으로 유사한 쌍(positive pair)과 유사하지 않은 쌍(Negative pair)을 분리하여 task를 해결하는방법

따라서 같은 이미지로부터 파생된 augmentation은 positive pair로 인식함.

→ Recently adopted to graphs !

1. Introduction - Limitation



Augmentation is well defined on images, it may behave arbitrarily on graphs.

Graphs contain not only the semantic but also the structural information

Treating all other nodes apart from the node itself as negatives → overlooks the structural information of graphs

large amount of negative samples is required → high computational and memory costs

1. Introduction - AFGRL

Self-supervised learning framework for graphs

Augmentation-Free Graph Representation Learning (AFGRL)

Requires neither augmentation techniques nor negative samples for learning representations of graphs.

Use

original graph +

discovering k-NN search positive samples in the representation space

Filter out naively selected false positive samples.

2. Related Work – DGI

DGI : aims to learn node representations by maximizing the mutual information between the local patch of a graph.

Capturing the global information of the graph that is overlooked by graph convolutional networks (GCNs)

DGI is further improved by taking into account the mutual information regarding the edges and node attribute.

2. Related Work - GRACE

Inspired by SimCLR

Creates two augmented views of a graph

Contrastive method

However, sampling bias occurs.

Requires a large amount of negative samples for the model training, which incurs high computational and memory costs.

2. Related Work - GCA

GCA = GRACE + advanced adaptive augmentation techniques

However, the performance on downstream tasks is highly dependent on the selection of the augmentation scheme,

		Comp.	Photo	CS	Physics
Node Classi.	BGRL	-4.00%	-1.06%	-0.20%	-0.69%
	GCA	-19.18%	-5.48%	-0.27%	OOM
Node Clust.	BGRL	-11.57%	-13.30%	-0.78%	-6.46%
	GCA	-26.28%	-23.27%	-1.64%	OOM

3. Problem Statement

G = {V, E} # 그래프

V : 노드들의 집합, v1, v2, …. vN

E : 엣지들의 집합, V × V

X : 그래프의 Feature matrix, N × F

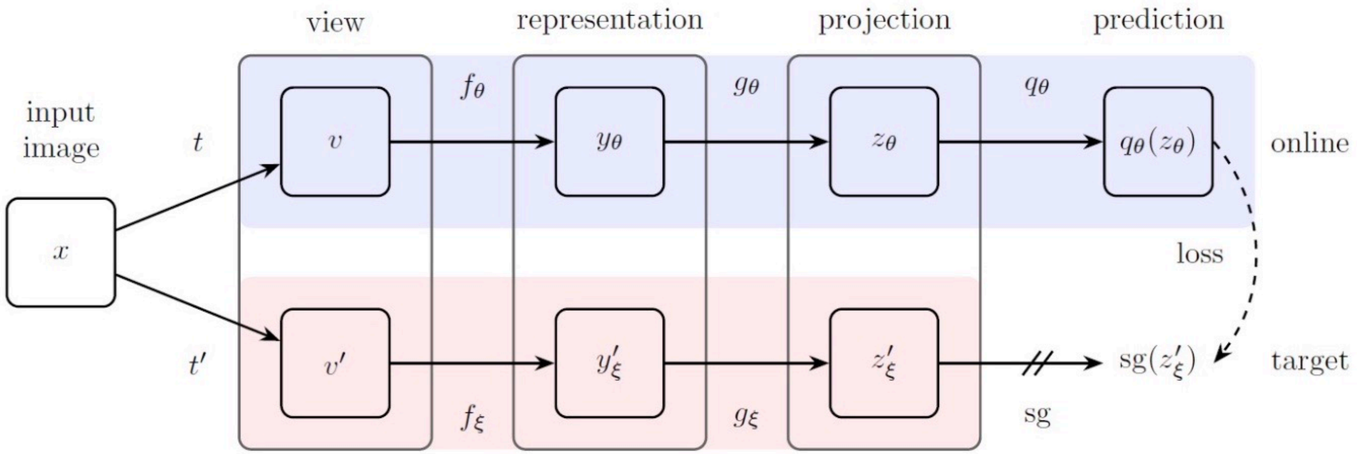
A : Adjacency matrix, N × N, (vi, vj)가 E에 속하면 1, 아니면 0

목표 : X와 A가 함께 G가 주어지면 노드 임베딩(H)을 수행하는 f(X, A), 인코더 f를 학습

클래스 정보를 사용하지 않고도 다양한 downstream으로 잘 일반화하는 노드 임베딩을 학습

4. Preliminary : Bootstrap Your Own Latent(BYOL)

Negative samples를 사용하지 않고 두 개의 네트워크의 출력을 반복적으로 bootstrap하는 방법



$$\mathcal{L}_{\theta,\xi} \triangleq \left\| \overline{q_\theta}(z_\theta) - \overline{z}'_\xi \right\|_2^2$$

$$\mathcal{L}^{\mathrm{BYOL}}_{\theta,\xi} = \mathcal{L}_{\theta,\xi} + \widetilde{\mathcal{L}}_{\theta,\xi}$$

$$\theta \leftarrow \mathrm{optimizer}\left(\theta, \nabla_\theta \mathcal{L}^{\mathtt{BYOL}}_{\theta,\xi}, \eta\right)$$

Online network

$$\xi \leftarrow \tau\xi + (1-\tau)\theta$$

Target network Online network

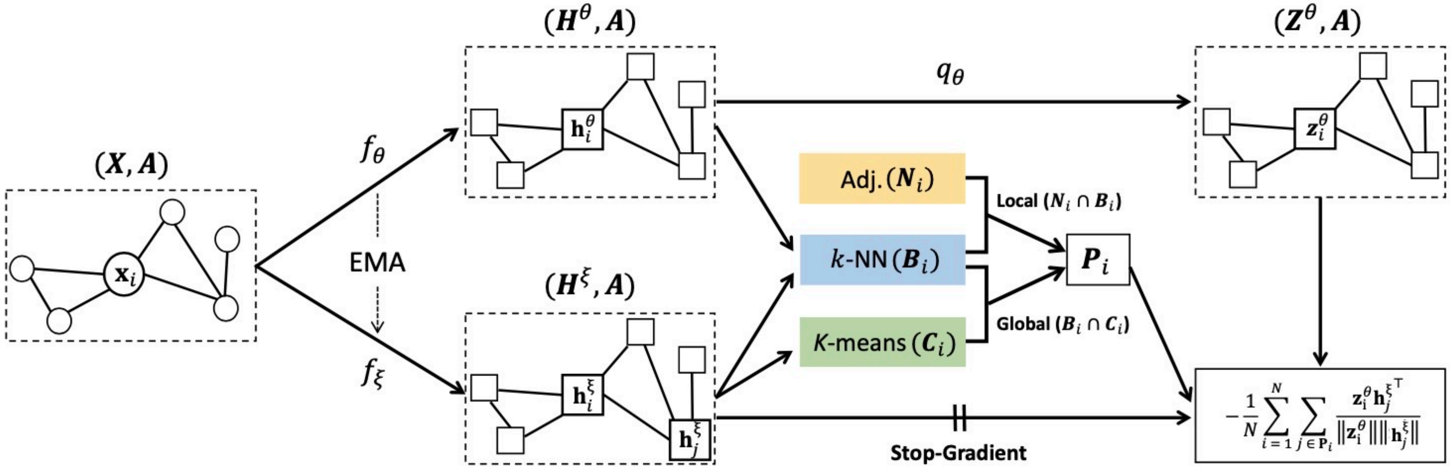
5. Proposed Method - BGRL

BGRL → BYOL을 그래프에 적용한 것, negative sample 불필요

		Comp.	Photo	CS	Physics
Node Classi.	BGRL	-4.00%	-1.06%	-0.20%	-0.69%
	GCA	-19.18%	-5.48%	-0.27%	OOM
Node Clust.	BGRL	-11.57%	-13.30%	-0.78%	-6.46%
	GCA	-26.28%	-23.27%	-1.64%	OOM

→ 증강 기법에 의존하지 않고도 원래 그래프의 representation을 형성하는 안정적이고 general한 framework가 필요하다 → AFGRL

5. Proposed Method - AFGRL



$$sim(v_i, v_j) = \frac{\mathbf{h}_i^{\theta} \cdot \mathbf{h}_j^{\xi}}{\|\mathbf{h}_i^{\theta}\| \|\mathbf{h}_j^{\xi}\|}, \forall v_j \in \mathcal{V}$$

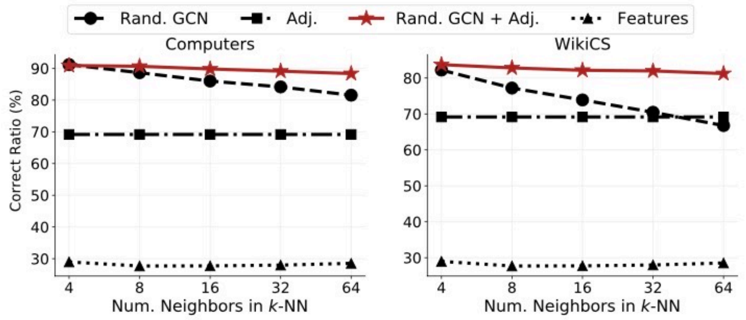
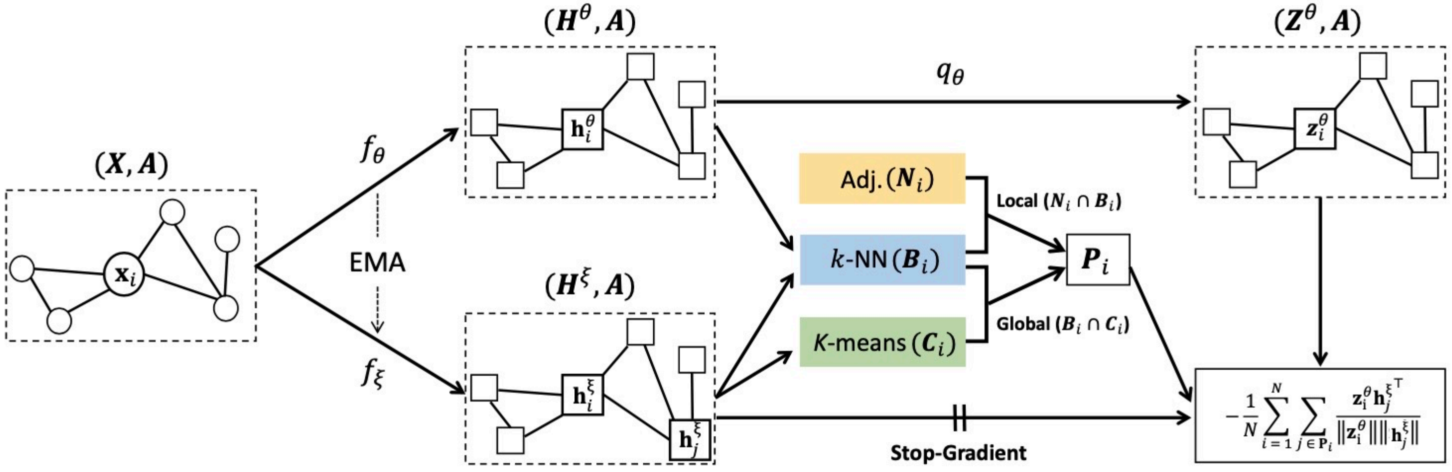


Figure 3: Analysis on the ratio of its neighboring nodes being the same label as the query node across different ks.

local한 관점과 global한 관점에서 동시에 filtering

5. Proposed Method - AFGRL



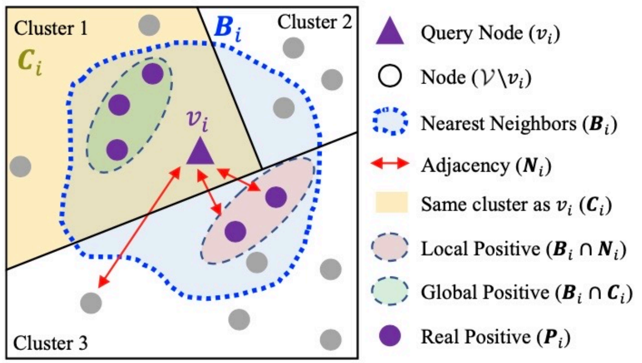


Figure 4: An overview of obtaining real positives of node v_i.

즉, real positive for query node(P)는
B : Query node의 k-NN 노드들의 집합
N : Query node의 adjacent 노드들의 집합
C : Query node와 같은 cluster에 포함된 노드들의 집합
이라고 할 때, P = (B∩N) ∪ (B∩C)로 나타내어진다.

이때 Query node과 P 사이의 cosine distance를 최소화하는 방향으로 학습된다.

6. Experiment - methods

WikiCS, Amazon computer, Amazon Photo, Coauthor-Cs, Coauthor-Physics

Node classification, Node clustering, Node similarity search

Node classification : 학습된 임베딩을 이용해 로지스틱 regression classification을 training하여 테스트. (best)

Node clustering, Node similarity search : 매 epoch마다 학습된 임베딩에 대해 평가함(best)

6. Experiment - methods

Encoder : GCN

W_l : l layer의 weight matrix

A-hat : including self-loop

$$\mathbf{H}^{(l)} = \mathrm{GCN}^{(l)}(\mathbf{X}, \mathbf{A}) = \sigma(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{W}^{(l)})$$

6. Experiment - results

	WikiCS	Computers	Photo	Co.CS	Co.Physics
Sup. GCN	77.19 ± 0.12	86.51 ± 0.54	92.42 ± 0.22	93.03 ± 0.31	95.65 ± 0.16
Raw feats.	71.98 ± 0.00	73.81 ± 0.00	78.53 ± 0.00	90.37 ± 0.00	93.58 ± 0.00
node2vec	71.79 ± 0.05	84.39 ± 0.08	89.67 ± 0.12	85.08 ± 0.03	91.19 ± 0.04
DeepWalk	74.35 ± 0.06	85.68 ± 0.06	89.44 ± 0.11	84.61 ± 0.22	91.77 ± 0.15
DW + feats.	77.21 ± 0.03	86.28 ± 0.07	90.05 ± 0.08	87.70 ± 0.04	94.90 ± 0.09
DGI	75.35 ± 0.14	83.95 ± 0.47	91.61 ± 0.22	92.15 ± 0.63	94.51 ± 0.52
GMI	74.85 ± 0.08	82.21 ± 0.31	90.68 ± 0.17	OOM	OOM
MVGRL	77.52 ± 0.08	87.52 ± 0.11	91.74 ± 0.07	92.11 ± 0.12	95.33 ± 0.03
GRACE	**77.97 ± 0.63**	86.50 ± 0.33	92.46 ± 0.18	92.17 ± 0.04	OOM
GCA	77.94 ± 0.67	87.32 ± 0.50	92.39 ± 0.33	92.84 ± 0.15	OOM
BGRL	76.86 ± 0.74	89.69 ± 0.37	93.07 ± 0.38	92.59 ± 0.14	95.48 ± 0.08
AFGRL	77.62 ± 0.49	**89.88 ± 0.33**	**93.22 ± 0.28**	**93.27 ± 0.17**	**95.69 ± 0.10**

Table 2: Performance on node classification (OOM: Out of memory on 24GB RTX3090).

섬세한 augmentation parameter 조절이 필요한 모델 < AFGRL

6. Experiment -results

		GRACE	GCA	BGRL	AFGRL
WikiCS	NMI	**0.4282**	0.3373	0.3969	0.4132
	Hom.	**0.4423**	0.3525	0.4156	0.4307
Computers	NMI	0.4793	0.5278	0.5364	**0.5520**
	Hom.	0.5222	0.5816	0.5869	**0.6040**
Photo	NMI	0.6513	0.6443	**0.6841**	0.6563
	Hom.	0.6657	0.6575	**0.7004**	0.6743
Co.CS	NMI	0.7562	0.7620	0.7732	**0.7859**
	Hom.	0.7909	0.7965	0.8041	**0.8161**
Co.Physics	NMI	OOM	OOM	0.5568	**0.7289**
	Hom.	OOM	OOM	0.6018	**0.7354**

Table 3: Performance on node clustering in terms of NMI and homogeneity.

		GRACE	GCA	BGRL	AFGRL
WikiCS	Sim@5	0.7754	0.7786	0.7739	**0.7811**
	Sim@10	0.7645	**0.7673**	0.7617	0.7660
Computers	Sim@5	0.8738	0.8826	0.8947	**0.8966**
	Sim@10	0.8643	0.8742	0.8855	**0.8890**
Photo	Sim@5	0.9155	0.9112	**0.9245**	0.9236
	Sim@10	0.9106	0.9052	**0.9195**	0.9173
Co.CS	Sim@5	0.9104	0.9126	0.9112	**0.9180**
	Sim@10	0.9059	0.9100	0.9086	**0.9142**
Co.Physics	Sim@5	OOM	OOM	0.9504	**0.9525**
	Sim@10	OOM	OOM	0.9464	**0.9486**

Table 4: Performance on similarity search. (Sim@n: Average ratio among n nearest neighbors sharing the same label as the query node.)

Capturing global semantics !

6. Experiment – Ablation Study

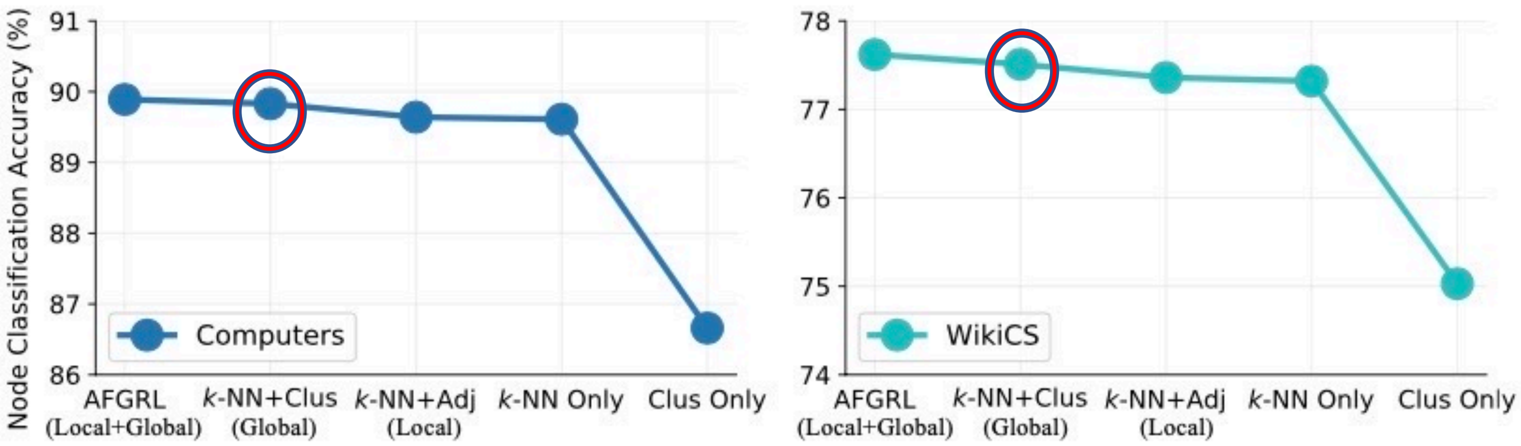


Figure 6: Ablation study on AFGRL.

Global semantics > local structure

When performing k-NN, we can obtain enough local information contained in the adjacency matrix.

→ Practical

6. Experiment – Visualization of embeddings

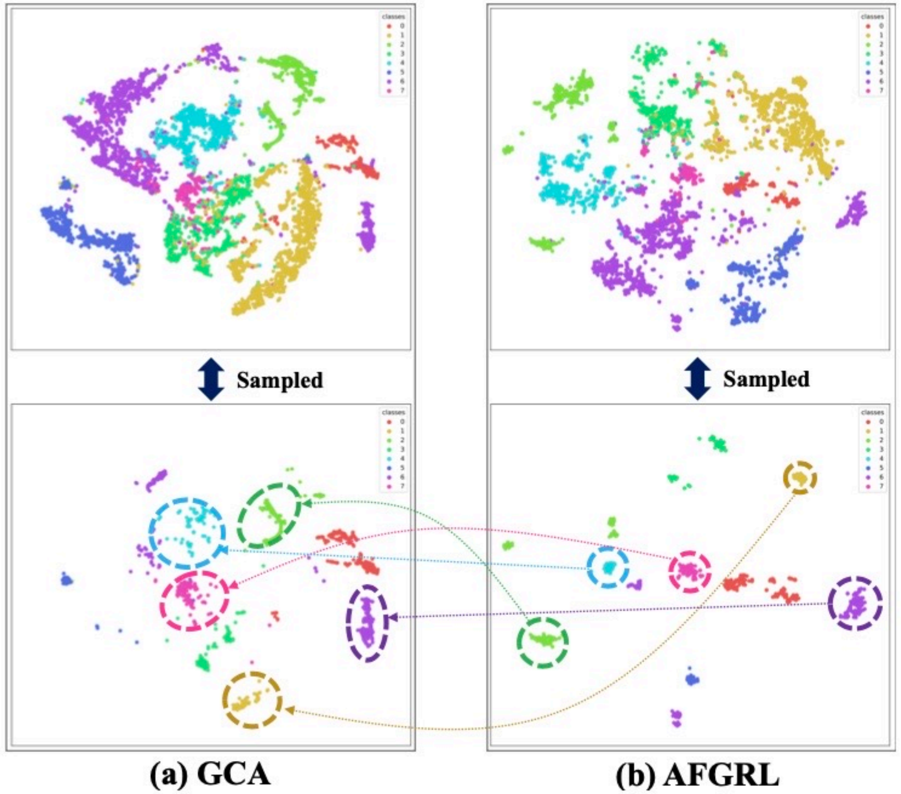


Figure 8: t-SNE embeddings of nodes in *Photo* dataset.

7. Conclusion

Graph representation learning

Self-supervised learning framework for graphs that do not require Augmentation and negative samples

Consider Local structure, global semantic

→Discovering positive sample

Low dependency on the hyperparameter

Generally available → can solve various downstream problems.

8. Codes

```python
class embedder:
    def __init__(self, args):
        self.args = args
        self.hidden_layers = eval(args.layers)
        printConfig(args)

    def infer_embeddings(self, epoch):
        self._model.train(False)
        self._embeddings = self._labels = None
        self._train_mask = self._dev_mask = self._test_mask = None
        for bc, batch_data in enumerate(self._loader):
            batch_data.to(self._device)
            emb, _, _, _ = self._model(x=batch_data.x, y=batch_data.y,
                                       edge_index=batch_data.edge_index, edge_weight=batch_data.edge_attr, epoch=epoch,
                                        neighbor=[batch_data.neighbor_index, batch_data.neighbor_attr])
            emb = emb.detach()
            y = batch_data.y.detach()
            if self._embeddings == None:
                self._embeddings, self._labels = emb, y
            else:
                self._embeddings = torch.cat([self._embeddings, emb])
                self._labels = torch.cat([self._labels, y])

    def evaluate(self, task, epoch):
        if task == "node":
            self.evaluate_node(epoch)
        elif task == "clustering":
            self.evaluate_clustering(epoch)
        elif task == "similarity":
            self.run_similarity_search(epoch)


    def evaluate_node(self, epoch):

        # print()
        # print("Evaluating ...")
        emb_dim, num_class = self._embeddings.shape[1], self._labels.unique().shape[0]

        dev_accs, test_accs = [], []

        for i in range(20):

            self._train_mask = self._dataset[0].train_mask[i]
            self._dev_mask = self._dataset[0].val_mask[i]
```

```python
class Encoder(nn.Module):

    def __init__(self, layer_config, dropout=None, project=False, **kwargs):
        super().__init__()
        self.stacked_gnn = nn.ModuleList([GCNConv(layer_config[i - 1], layer_config[i]) for i in range(1, len(layer_config))])
        self.stacked_bns = nn.ModuleList([nn.BatchNorm1d(layer_config[i], momentum=0.01) for i in range(1, len(layer_config))])
        self.stacked_prelus = nn.ModuleList([nn.PReLU() for _ in range(1, len(layer_config))])

    def forward(self, x, edge_index, edge_weight=None):
        for i, gnn in enumerate(self.stacked_gnn):
            x = gnn(x, edge_index, edge_weight=edge_weight)
            x = self.stacked_bns[i](x)
            x = self.stacked_prelus[i](x)

        return x
```

8. Codes

```python
class Dataset(InMemoryDataset):

    """
    A PyTorch InMemoryDataset to build multi-view dataset through graph data augmentation
    """

    def __init__(self, root="data", dataset='cora', transform=None, pre_transform=None):
        self.root, self.dataset, self.data_dir = download_data(root=root, dataset=dataset)
        create_dirs(self.dirs)
        super().__init__(root=self.data_dir, transform=transform, pre_transform=pre_transform)
        path = osp.join(self.data_dir, "processed", self.processed_file_names[0])
        self.data, self.slices = torch.load(path)


    def process_full_batch_data(self, data):

        print("Processing full batch data")
        nodes = torch.tensor(np.arange(data.num_nodes), dtype=torch.long)

        edge_index, edge_attr = add_self_loops(data.edge_index, data.edge_attr)

        data = Data(nodes=nodes, edge_index=data.edge_index, edge_attr=data.edge_attr, x=data.x, y=data.y,
                    train_mask=data.train_mask, val_mask=data.val_mask, test_mask=data.test_mask,
                    num_nodes=data.num_nodes, neighbor_index=edge_index, neighbor_attr=edge_attr)

        return [data]
```

8. Codes

```python
class AFGRL_ModelTrainer(embedder):

    def __init__(self, args):
        embedder.__init__(self, args)
        self._args = args
        self._init()
        self.config_str = config2string(args)
        print("\n[Config] {}\n".format(self.config_str))
        self.writer = SummaryWriter(log_dir="runs/{}".format(self.config_str))

    def _init(self):
        args = self._args
        self._task = args.task
        print("Downstream Task : {}".format(self._task))
        os.environ["CUDA_VISIBLE_DEVICES"] = str(args.device)
        self._device = f'cuda:{args.device}' if torch.cuda.is_available() else "cpu"
        torch.cuda.set_device(self._device)
        self._dataset = Dataset(root=args.root, dataset=args.dataset)
        self._loader = DataLoader(dataset=self._dataset)
        layers = [self._dataset.data.x.shape[1]] + self.hidden_layers
        self._model = AFGRL(layers, args).to(self._device)
        self._optimizer = optim.AdamW(params=self._model.parameters(), lr=args.lr, weight_decay= 1e-5)

    def train(self):

        self.best_test_acc, self.best_dev_acc, self.best_test_std, self.best_dev_std, self.best_epoch = 0, 0, 0, 0, 0
        self.best_dev_accs = []

        # get Random Initial accuracy
        self.infer_embeddings(0)
        print("initial accuracy ")
        self.evaluate(self._task, 0)

        f_final = open("results/{}.txt".format(self._args.embedder), "a")

        # Start Model Training
        print("Training Start!")
        self._model.train()
        for epoch in range(self._args.epochs):
            for bc, batch_data in enumerate(self._loader):
                batch_data.to(self._device)
                _, loss, ind, k = self._model(x=batch_data.x, y=batch_data.y, edge_index=batch_data.edge_index,
```

```python
class AFGRL(nn.Module):
    def __init__(self, layer_config, args, **kwargs):
        super().__init__()
        self.student_encoder = Encoder(layer_config=layer_config, dropout=args.dropout, **kwargs)
        self.teacher_encoder = copy.deepcopy(self.student_encoder)
        set_requires_grad(self.teacher_encoder, False)
        self.teacher_ema_updater = EMA(args.mad, args.epochs)
        self.neighbor = Neighbor(args)

        rep_dim = layer_config[-1]

        self.student_predictor = nn.Sequential(nn.Linear(rep_dim, args.pred_hid), nn.BatchNorm1d(args.pred_hid), nn.PReLU(), nn.Linear(args.pred_hid, rep_dim))
        self.student_predictor.apply(init_weights)

        self.topk = args.topk

    def reset_moving_average(self):
        del self.teacher_encoder
        self.teacher_encoder = None

    def update_moving_average(self):
        assert self.teacher_encoder != None, 'teacher encoder has not been created yet'
        update_moving_average(self.teacher_ema_updater, self.teacher_encoder, self.student_encoder)

    def forward(self, x, y, edge_index, neighbor, edge_weight=None, epoch=None):
        student = self.student_encoder(x=x, edge_index=edge_index, edge_weight=edge_weight)
        pred = self.student_predictor(student)

        with torch.no_grad():
            teacher = self.teacher_encoder(x=x, edge_index=edge_index, edge_weight=edge_weight)

        if edge_weight == None:
            adj = torch.sparse.FloatTensor(neighbor[0], torch.ones_like(neighbor[0][0]), [x.shape[0], x.shape[0]])
        else:
            adj = torch.sparse.FloatTensor(neighbor[0], neighbor[1], [x.shape[0], x.shape[0]])

        ind, k = self.neighbor(adj, F.normalize(student, dim=-1, p=2), F.normalize(teacher, dim=-1, p=2), self.topk, epoch)

        loss1 = loss_fn(pred[ind[0]], teacher[ind[1]].detach())
        loss2 = loss_fn(pred[ind[1]], teacher[ind[0]].detach())
        loss = loss1 + loss2

        return student, loss.mean(), ind, k
```

AFGRL 모델의 학습에 실패함.

Thank you for listening my presentation