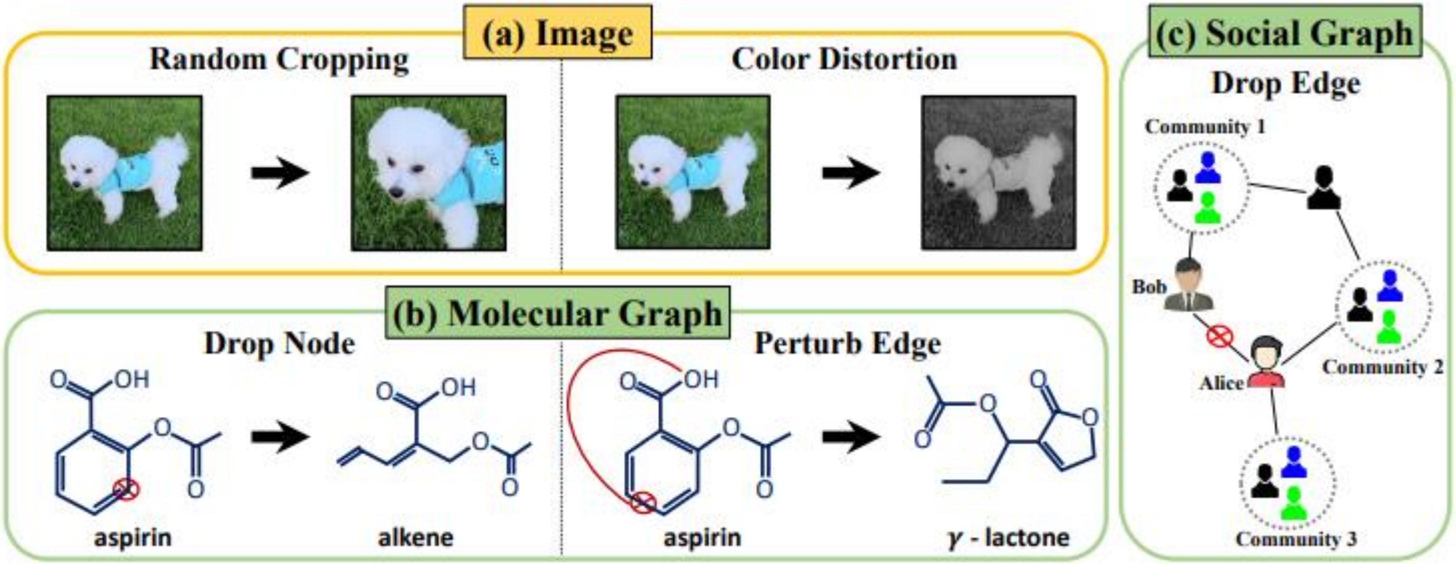
AFGRL

20200045 김건우

Contents

Introduction

- 최근 self-supervised learning분야에서 큰 성공 -> Graph에 적용?
- 그러나 Graph의 경우 semantic info(의미적 정보), structural info(구조적 정보)를 모두 포함



Introduction

- 따라서 graph에 대한 augmentation은 info를 해칠 수 있다.
- 더 나아가 augmentation-based contrastive methods는 hyperparameter 에 의존적이며
- 많은 negative samples이 필요함

- Augmentation-free 하면서도 negative sample이 필요하지 않는 새로운 self-supervised learning framework
- AFGRL(Augmentation-Free Graph Representation Learning)

Related Work

Contrastive Methods on Graphs

문제점 : sampling bias, large amount of negative samples, high computational and memory costs

BGRL은 sampling bias issue는 해결하였으나 여전히 augmentation에 관한 문제가 존재

Augmentations on Graphs

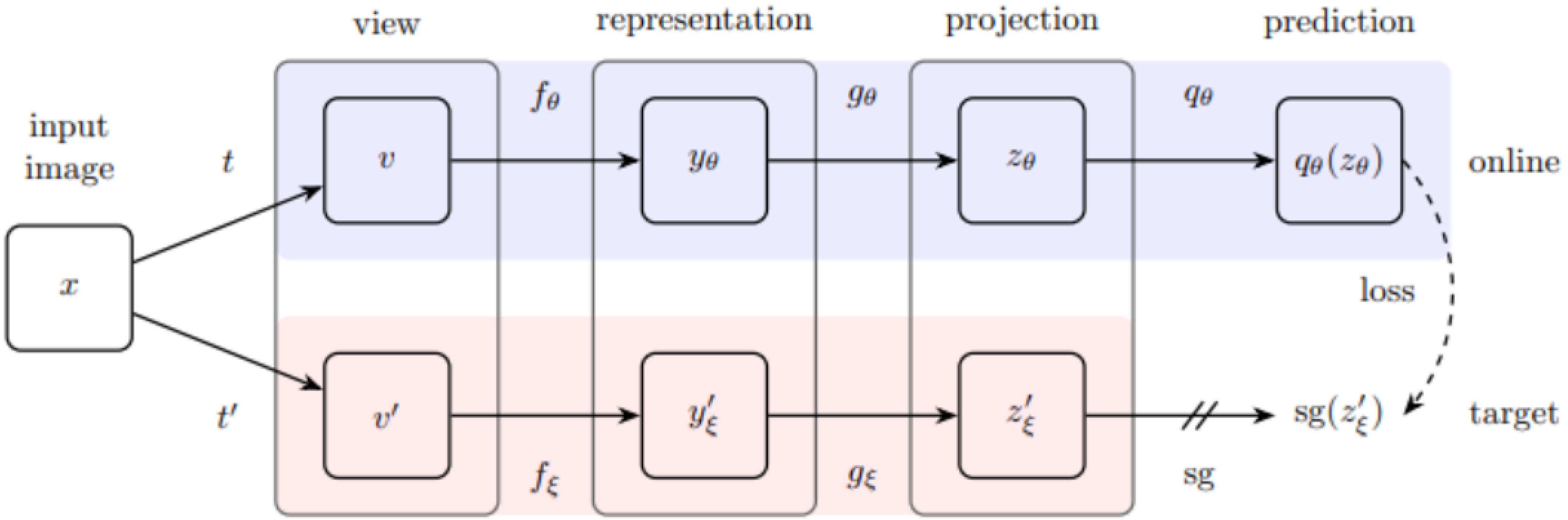
문제점 : 보편적인 augmentations는 없음(parameter에 의존적),

domain information이 효과적인 augmentation에 도움되나 항상 적용 가능하지는 않음

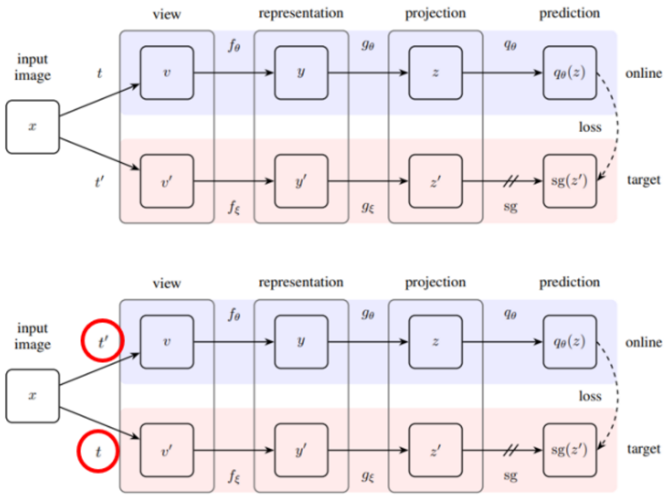
Problem Statement

- Graph G에 대해 X(feature matrix), A(adjacency matrix)가 주어졌을 때, node embedding H = f(X, A) 를 생성하는 **인코더 f(·) 를 학습하는 것**

Preliminary(BYOL)



Preliminary(BYOL)



$\mathcal{L}_\theta^{\text{BYOL}}$

$\widetilde{\mathcal{L}}_\theta^{\text{BYOL}}$

[BYOL의 loss function symmetrization]

Total Loss

$$\mathcal{L}_\theta^{\text{BYOL}} + \widetilde{\mathcal{L}}_\theta^{\text{BYOL}}$$

$$\theta \leftarrow \text{optimizer}\left(\theta, \nabla_\theta \mathcal{L}_{\theta,\xi}^{\text{BYOL}}, \eta\right),$$
$$\xi \leftarrow \tau\xi + (1-\tau)\theta,$$

(exponential moving average)

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

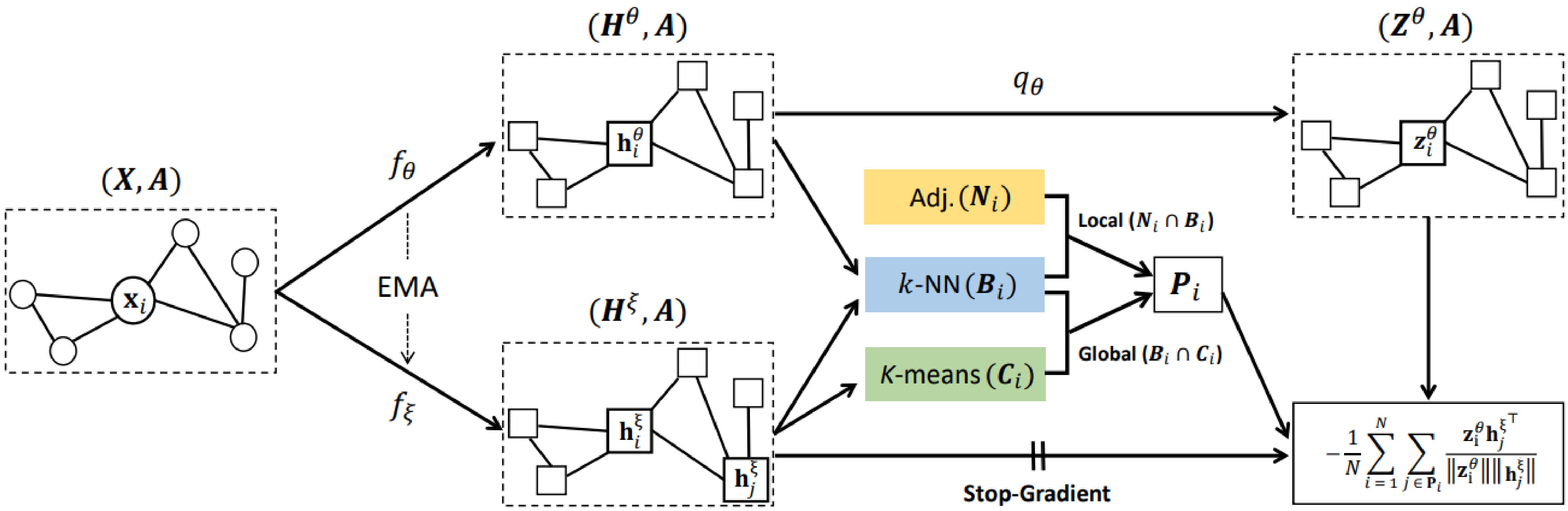
(MSE with l2 norm)

Proposed Method(AFGRL)

- BYOL을 그래프에 적용: BGRL
- 장점: Negative sample 불필요, 큰 graph에 적용 가능
- 한계점: augmentation scheme에 의존, 다양한 downstream task에 대해 각각의 hyperparameters를 선택해야 함

		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

Proposed Method(AFGRL)



f_θ : 노드 임베딩 H^θ 생성, update by gradient descent
f_ξ : 노드 임베딩 H^ξ 생성, update by EMA of f_θ
P_i: real positive
q_θ : predictor(H^θ to Z^θ)

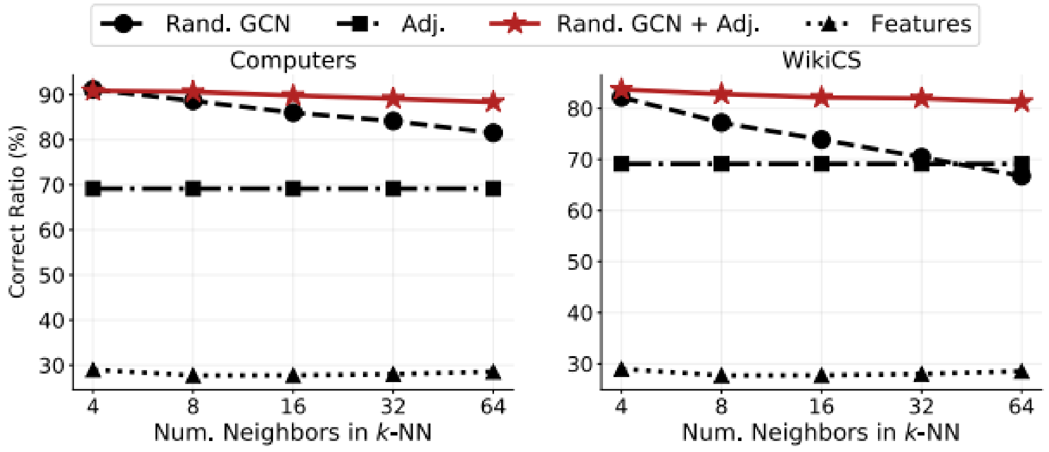
Proposed Method(AFGRL)

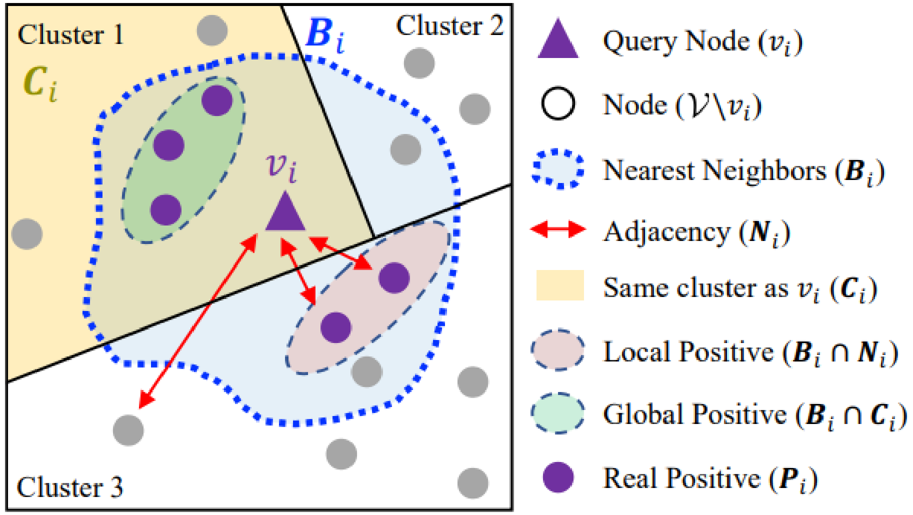
$$sim(v_i, v_j) = \frac{\mathbf{h}_i^{\theta} \cdot \mathbf{h}_j^{\xi}}{\|\mathbf{h}_i^{\theta}\| \|\mathbf{h}_j^{\xi}\|}, \forall v_j \in \mathcal{V}$$

for B_i(k-nn)

Capturing Local Structural Information + Capturing Global Semantics

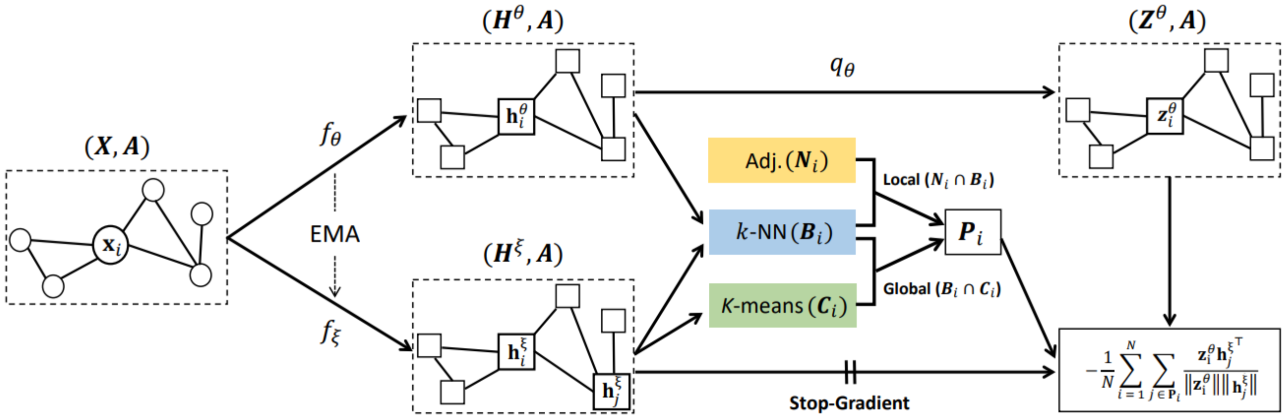
(B ∩ N) (B ∩ C)





P = (B ∩ N) ∪ (B ∩ C)

Proposed Method(AFGRL)



$$\mathbf{P}_i = (\mathbf{B}_i \cap \mathbf{N}_i) \cup (\mathbf{B}_i \cap \mathbf{C}_i)$$

$$\mathcal{L}_{\theta,\xi} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{v_j \in \mathbf{P}_i} \frac{\mathbf{z}_i^\theta \mathbf{h}_j^{\xi\top}}{\left\|\mathbf{z}_i^\theta\right\| \left\|\mathbf{h}_j^\xi\right\|},$$

Query node과 P 사이의
cosine distance를 최소화하며 학습

Experiments

Performance on node classification

	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**

Augmentation parameters 조정이 필요한 모델보다 AFGRL이 전반적으로 뛰어난 성능을 보임

Experiments

Performance on node clustering

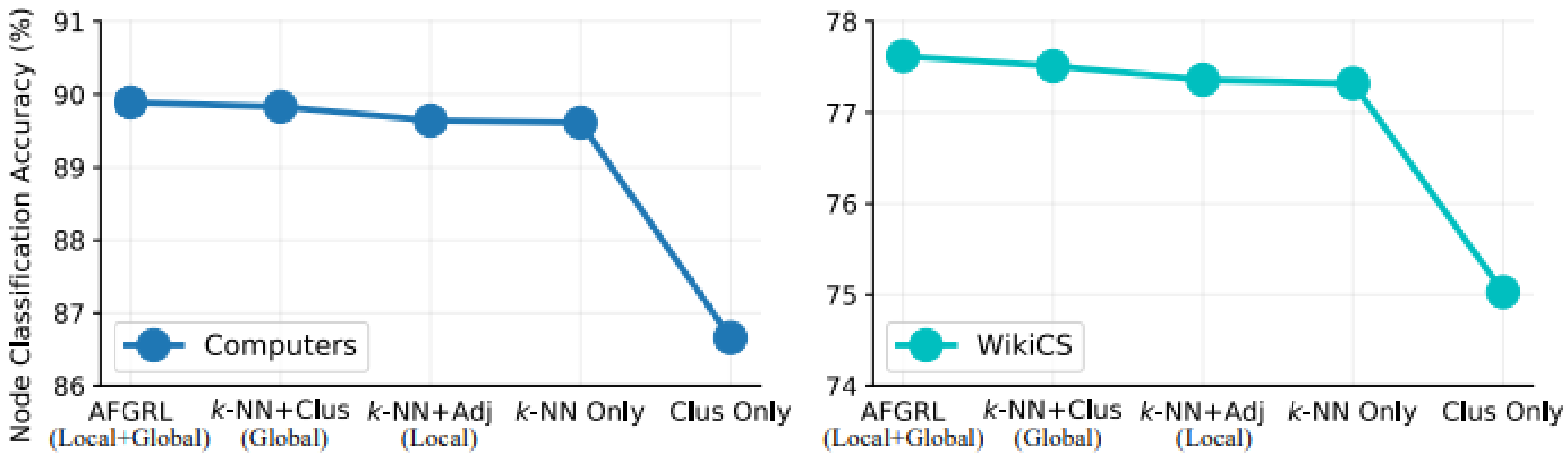
		GRACE	GCA	BGRL	AFGRL
WikiCS	NMI	**0.4282**	0.3373	0.3969	0.4132
	Hom.	**0.4423**	0.3525	0.4156	0.4307
Computers	NMI	0.4793	0.5278	0.5364	**0.5520**
	Hom.	0.5222	0.5816	0.5869	**0.6040**
Photo	NMI	0.6513	0.6443	**0.6841**	0.6563
	Hom.	0.6657	0.6575	**0.7004**	0.6743
Co.CS	NMI	0.7562	0.7620	0.7732	**0.7859**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

Performance on similarity search

		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**

AFGRL은 global information도 포함하기
때문에 node clustering/ similarity search 에서
좋은 성능을 보임

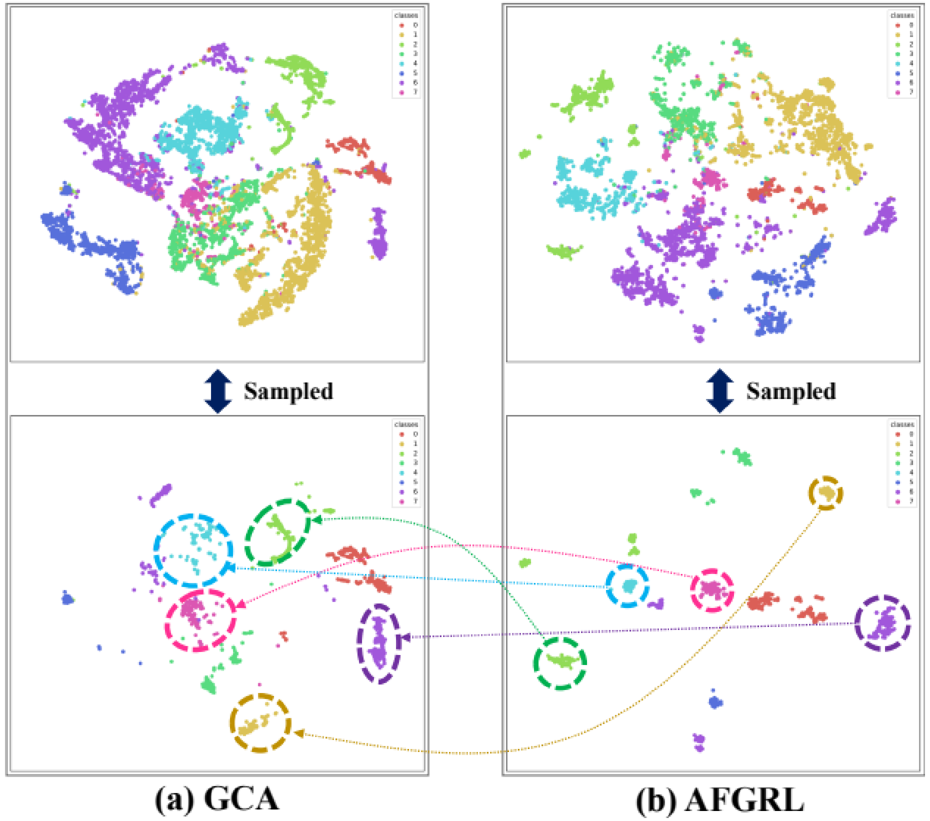
Experiments-Ablation Studies



Real Positive를 선택하는 방법에 따른 성능의 차이를 볼 수 있다.

- Local+Global인 경우가 가장 효과적
- Local보다는 Global을 고려하는 것이 더 효과적
- Adj matrix가 sparse한 경우에도 AFGRL이 효과적

Experiments-Visualization



GCA에 비해 AFGRL이 더 tightly(단단히) 하게 그룹화가 되어있음

Conclusion

- AFGRL은 그래프에 대한 self-supervised learning framework
- Augmentation, negative sample이 필요하지 않음
- local structural information + global semantics information을 통해 real positive를 결정
- Hyper parameter에 민감하지 않으며 다양한 downstream work에 적용가능(성능 역시 뛰어남)

Implement

```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 is not 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)

        print(ind)
        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
```

My Experiments-Ablation Studies

	AFGRL	local	global
computers	88.3156	88.3156	88.3156
wikics	74.6032	74.5912	74.5989

1.전반적으로 논문에서의 값보다 성능은 좋지 않았음

2.Wikics 데이터의 경우 논문에서와 같이 AFGRL>global>local순서로 성능이 좋았음

3.Computers 데이터의 경우 성능의 차이가 없었음