Augmentation-Free Self-Supervised Learning on Graph

2023 Winter DSAIL Internship

한재민

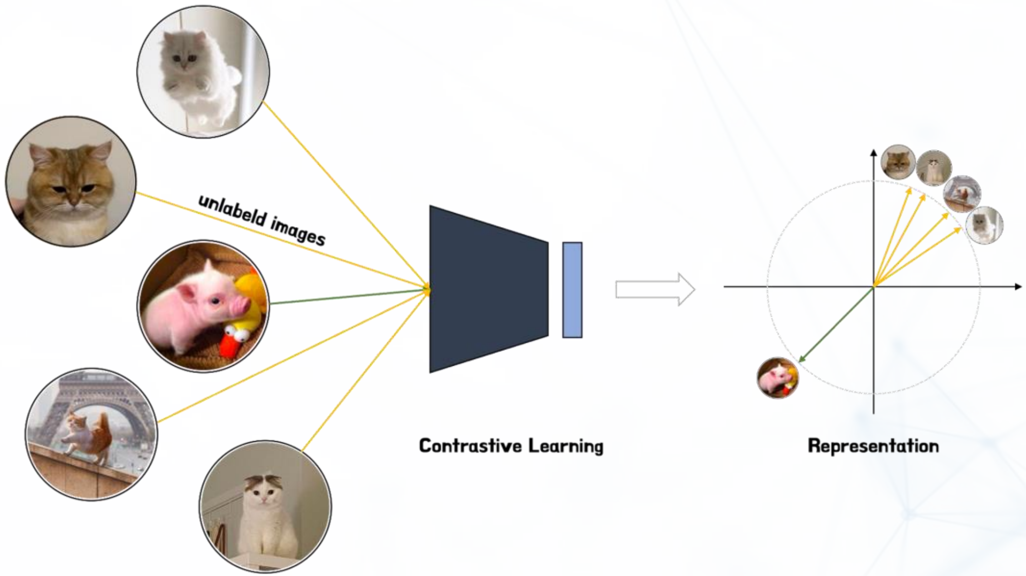
ISysE KAIST
Industrial & Systems Engineering

Contents

Introduction

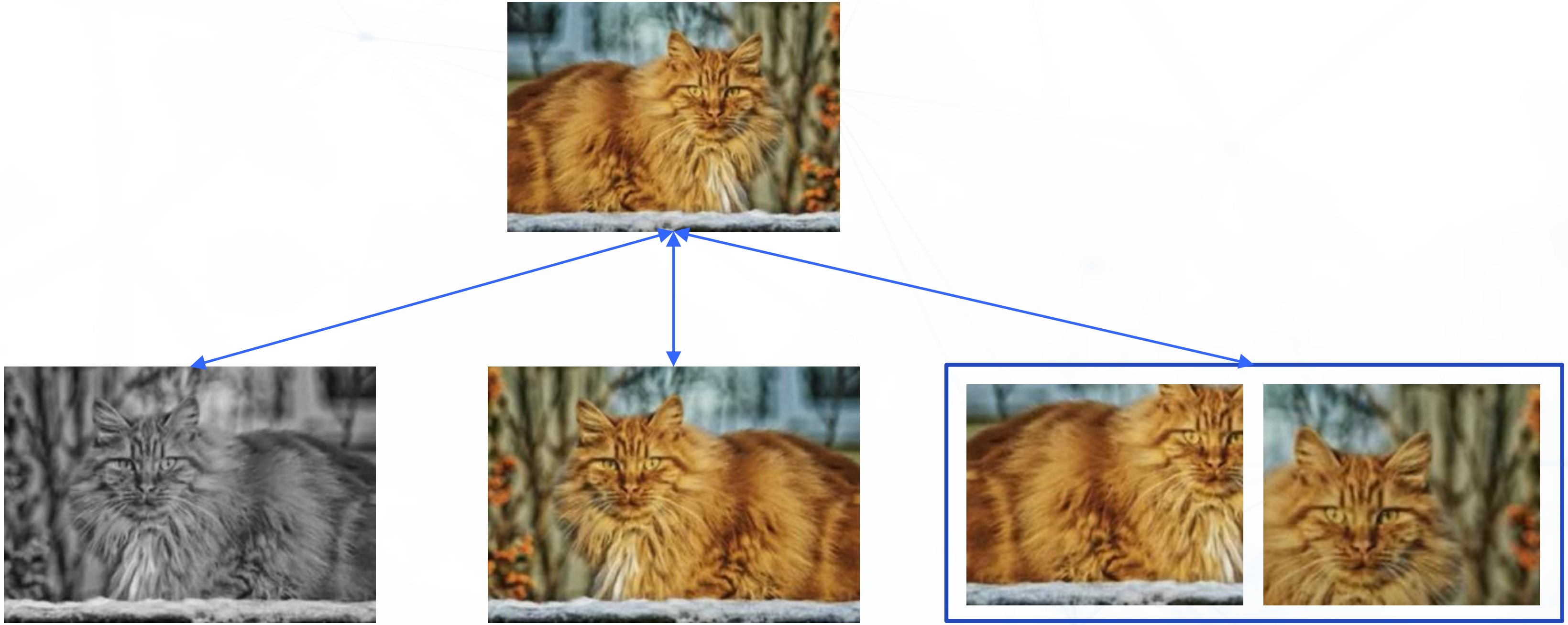
Contrastive Learning

Contrastive Learning is a type of self-supervised representation learning



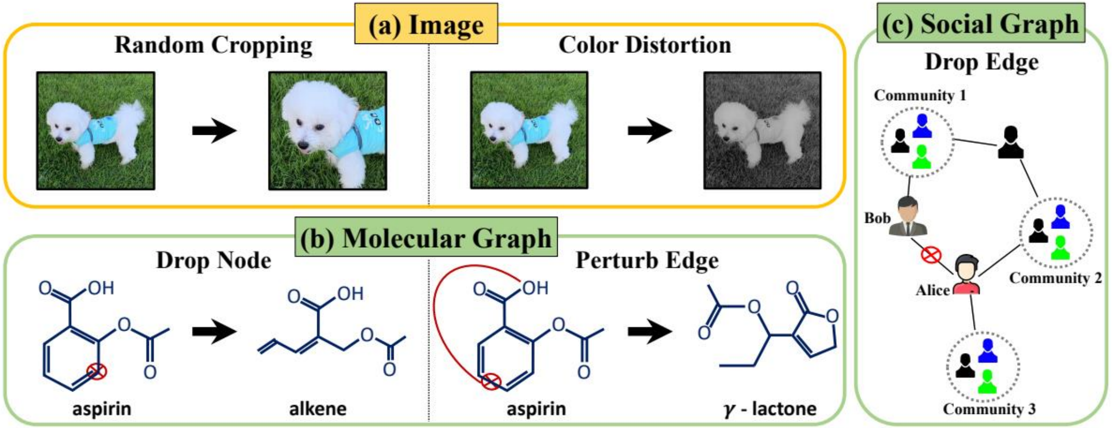
Introduction

Augmentation



Introduction

Augmentation on Graph



- Augmentation may behave arbitrarily on graphs
- Graphs contain not only the semantic but also the structural information

Introduction

Problem

- The quality is dependent on the choice of the augmentation scheme

- Limitation of inherent philosophy of contrastive learning
  → Overlooks the structural information of graphs

- Requirements of a large amount of negative samples
  → High computational and memory costs, impractical in reality

Background

Related Work-Contrastive Methods on Graphs

DGI

Learn node representations by maximizing the mutual information between the local patch of a graph

GRACE

Creates two augmented view of a graph(for the first)

However, they have sampling bias problem → BGRL

Background

Related Work-Augmentations on Graphs

GRACE

Randomly drops edges and masks node features

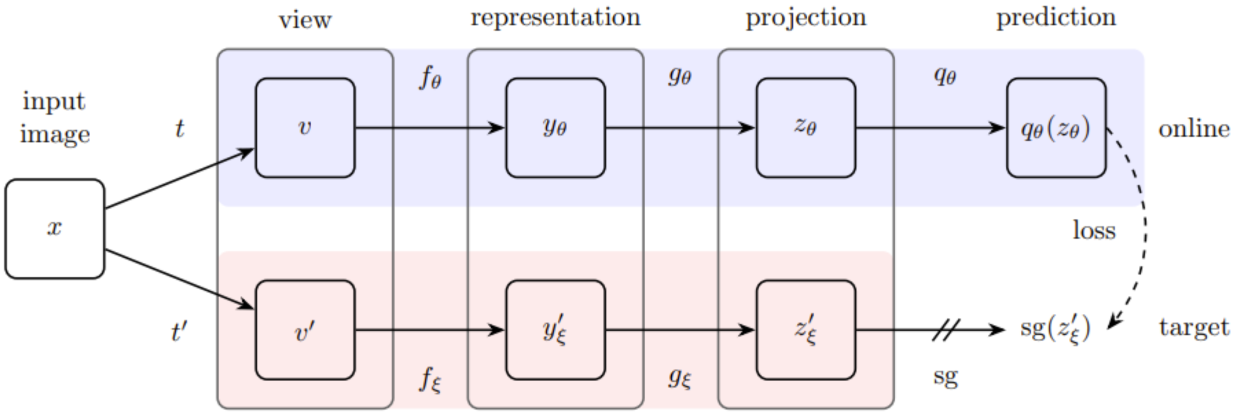
GCA

GRACE + advanced adaptive augmentation techniques that consider both structural and attribute information

There is no universally outperforming data augmentation scheme for graphs.

Background

BYOL(Bootstrap Your Own Latent)



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

Background

BYOL(Bootstrap Your Own Latent)

$$\mathcal{L}_{\theta,\xi} = \left\|\bar{q}_\theta(\mathbf{z_1}) - \bar{\mathbf{z}}_2\right\|^2$$

$\mathcal{L}^{\text{BYOL}}_{\theta,\xi} = \mathcal{L}_{\theta,\xi} + \widetilde{\mathcal{L}}_{\theta,\xi}.$ symmetric(+x_1, x_2 change loss)

Each training

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

$\xi \leftarrow \tau\xi + (1-\tau)\theta$ EMA(Exponential Moving Average)

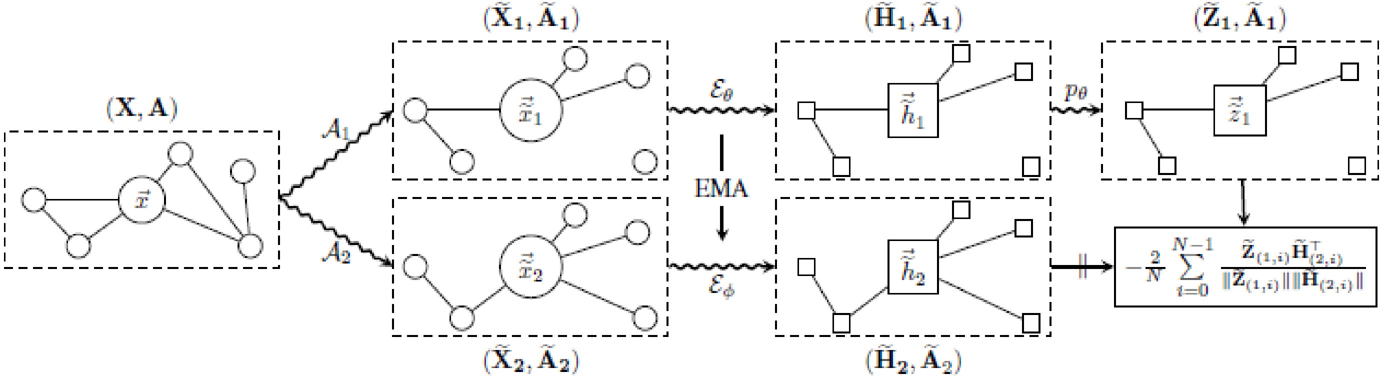
η : learning rate for online network

γ : decay rate that controls how close ξ remains to θ

Background

BGRL(Bootstrap Your Own Latent)

- Fully non-contrastive method for learning node representations
- Do not need negative sample
- Simple Augmentation(node feature masking, edge masking)



Background

Limitation of BGRL

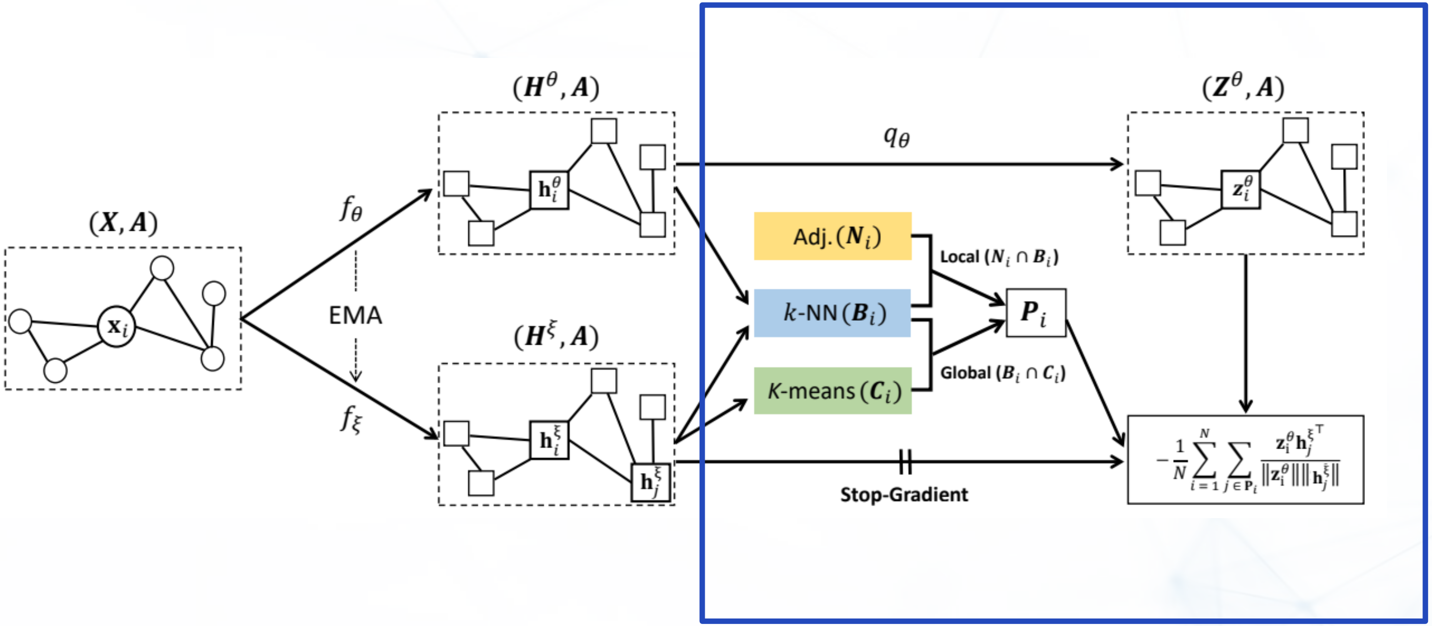
		Comp.	Photo	CS	Physics
Node Classi.	BGRL	-4.00%	-1.06%	-0.20%	-0.69%
	GCA	-19.18%	-5.48%	-0.27%	OOM
Node Clust.	BGRL	-11.57%	-13.30%	-0.78%	-6.46%
	GCA	-26.28%	-23.27%	-1.64%	OOM

$$-\frac{(\text{best}-\text{worst})}{\text{best}} \times 100$$

Performance on downstream tasks learned by BGRL varies greatly according to the choice of hyperparameters associated with augmentations

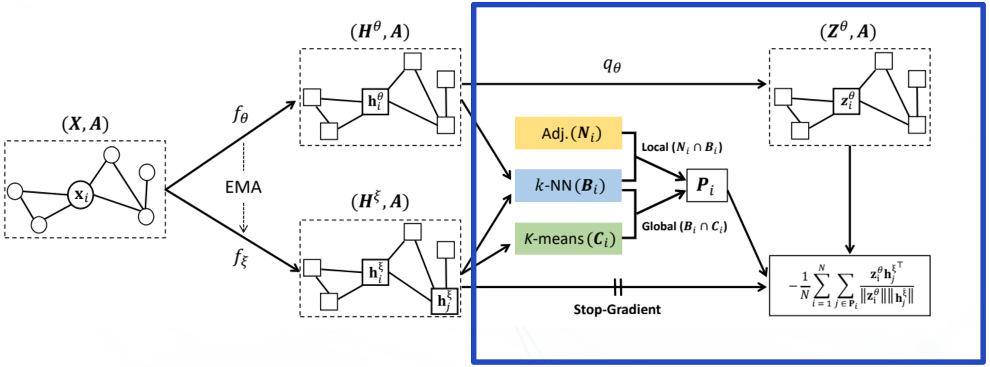
Method

AFGRL

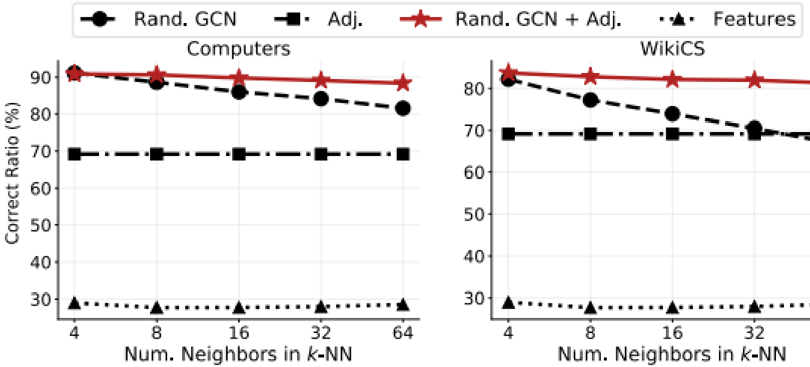


Method

AFGRL

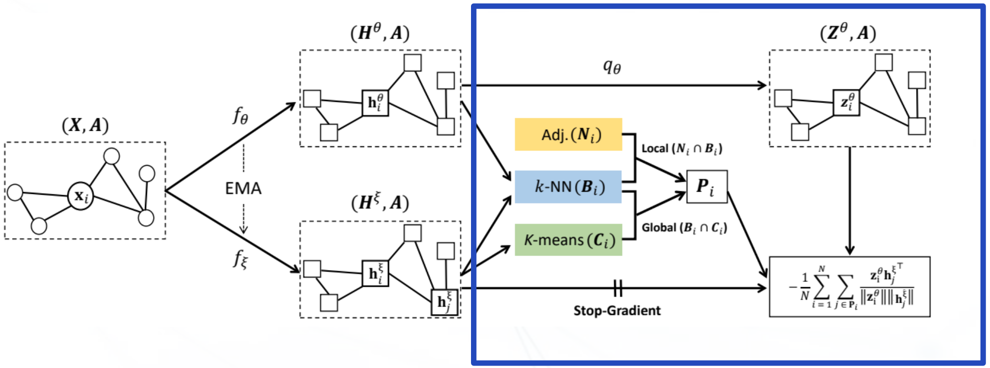


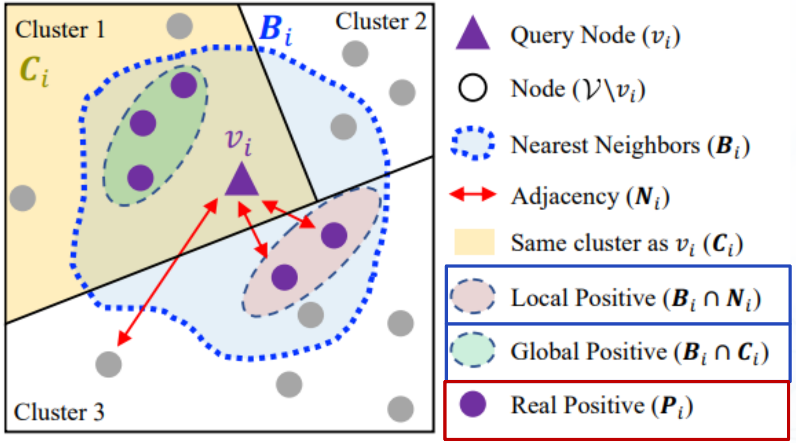
$$sim(v_i, v_j) = \frac{\mathbf{h}_i^\theta \cdot \mathbf{h}_j^\xi}{\|\mathbf{h}_i^\theta\|\|\mathbf{h}_j^\xi\|}, \forall v_j \in \mathcal{V}$$



Method

AFGRL

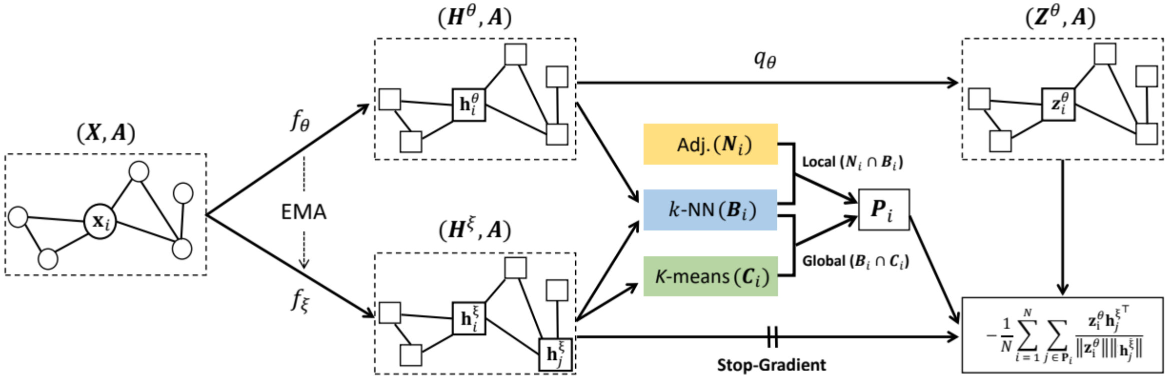




$$\mathbf{P}_i = (\mathbf{B}_i \cap \mathbf{N}_i) \cup (\mathbf{B}_i \cap \mathbf{C}_i)$$

Method

AFGRL



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

Experiments

Datasets

	# Nodes	# Edges	# Feat.	# Cls.
WikiCS	11,701	216,123	300	10
Amazon-Computers	13,752	245,861	767	10
Amazon-Photo	7,650	119,081	745	8
Coauthor-CS	18,333	81,894	6,805	15
Coauthor-Physics	34,493	247,962	8,415	5

Table 5: Statistics for datasets used in this paper.

5 datasets(WikiCS, Amazon-Computers, Amazon-Photo, Coauthor-CS, Coauthor-Physics)

Experiments

Evaluation protocol

Three node-level tasks

- Node classification
- Node clustering
- Node similarity search

Encoder : GCN

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**

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

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**

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

Experiments

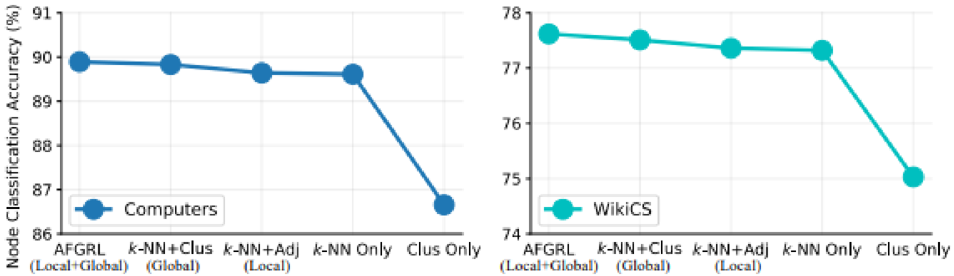
Performance on similarity search

		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**

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

Experiments

Ablation Study



AFGRL gives competitive performance even when the adjacency matrix is sparse

→ Practical

Experiments

Hyperparameter Analysis

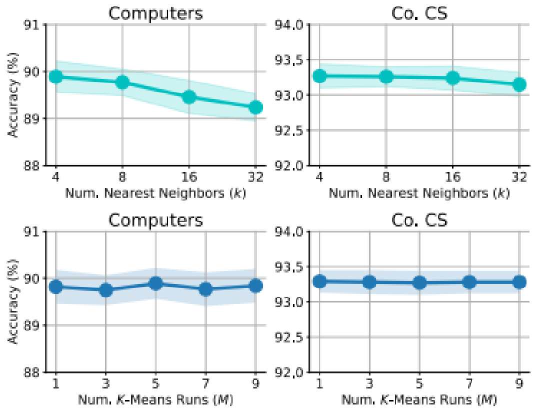
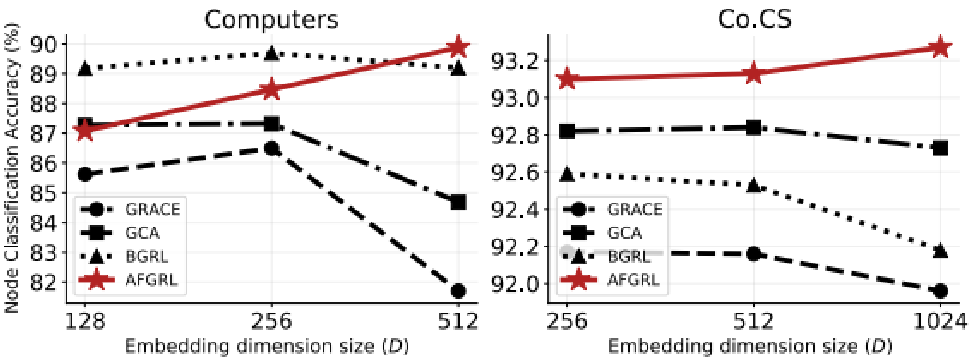


Figure 5: Sensitivity analysis.



Performance is stable over various Ms(M : number of clustering)

AFGRL benefits from high-dimensional embeddings

Experiments

Visualization of embeddings

Nodes are more tightly grouped in AFGRL than GCA

→ AFGRL captures more fine-grained class information

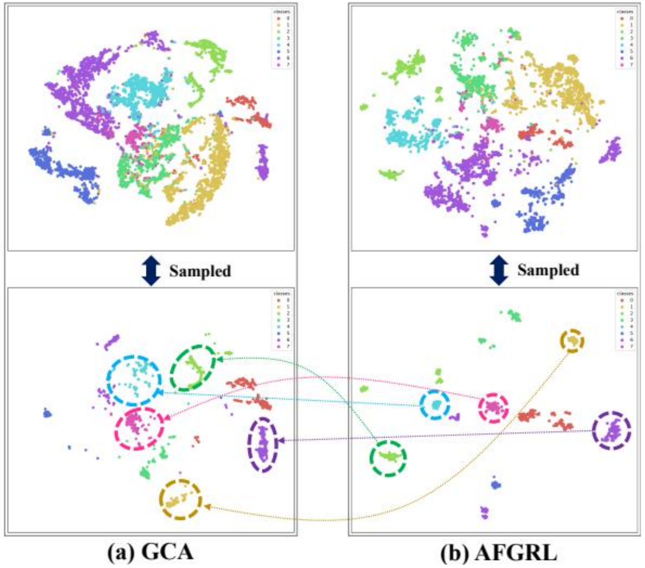


Figure 8: t-SNE embeddings of nodes in *Photo* dataset.

Conclusion

- Neither using augmentation techniques nor negative samples
- Discovers nodes that can serve as positive samples considering the local structural information and the global semantics of graphs
- Stability over hyperparameters(even without using negative sample)

Implementation

```python
[11] 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
```

```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,
                                                        neighbor=[batch_data.neighbor_index, batch_data.neigh
                                                        edge_weight=batch_data.edge_attr, epoch=epoch)

            emb = emb.detach()
            y = batch_data.y.detach()
            if self._embeddings is None:
                self._embeddings, self._labels = emb, y
            else:
```

Graphs

Wiki-CS

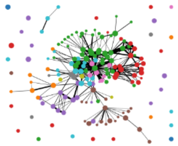
Edit

Introduced by Mernyei et al. in Wiki-CS: A Wikipedia-Based Benchmark for Graph Neural Networks

Wiki-CS is a Wikipedia-based dataset for benchmarking Graph Neural Networks. The dataset is constructed from Wikipedia categories, specifically 10 classes corresponding to branches of computer science, with very high connectivity. The node features are derived from the text of the corresponding articles. They were calculated as the average of pretrained GloVe word embeddings (Pennington et al., 2014), resulting in 300-dimensional node features.

The dataset has 11,701 nodes and 216,123 edges.

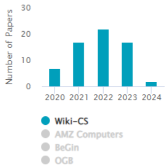
Source: Wiki-CS: A Wikipedia-Based Benchmark for Graph Neural Networks



Source: https://arxiv.org/pdf/2007.02901v1.pdf.

Homepage

Usage



Benchmarks

Edit

Trend	Task	Dataset Variant	Best Model	Paper	Code
	Node Classification	Wiki-CS	CGT		
	Node Clustering	Wiki-CS	CGT		

Implementation

	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	$\boxed{77.62 \pm 0.49}$	**89.88 ± 0.33**	**93.22 ± 0.28**	**93.27 ± 0.17**	**95.69 ± 0.10**

WikiCS에서는 0.77 정도로 논문에 나와있는데 1500 epoch에도 0.68정도 나옴

Implementation

- Related work, 선행 연구가 많아 어떤 점이 왜 바뀌었는지 찾는게 어려웠음
- 논문을 보다 헷갈릴 때, 코드를 보고 이해하는 경우가 있었음