Deep Graph Infomax

Petar Veličković et al.

박상우

2023.07.11

CONTENTS

Introduction

Graph Representation is current major challenges of ML

GCN is novel, but supervised

Dominant algorithms for unsupervised representation learning is based on random walk

But It has a limitation

- Over emphasize proximity information at the expense of structural information
- Highly dependent on hyperparameter
- With encoder, unclear whether it actually provide any useful signal

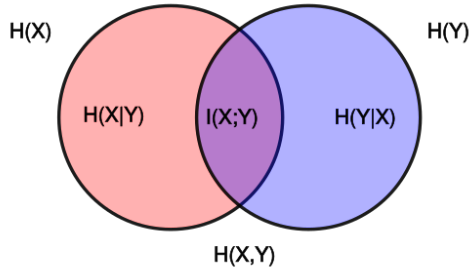
Introduction - Background

Mutual Information

$$I(X;Y) = D_{KL}(P_{(X,Y)} || P_X * P_Y)$$

$$D_{KL}(P||Q) = E_P[log\frac{dP}{dQ}] = \sum P(x)log(\frac{P(x)}{Q(x)})$$

$$I(X;Y) = \sum\sum p(x,y)log(\frac{p(x,y)}{p(x)p(y)})$$



$$I(X;Y) = I(Y;X) \qquad I(X;Y) \geq 0 \qquad I(X_1, X_2; Y) \geq I(X_1; Y)$$

$$\mathbb{I}(X;Y) = \mathbb{H}(X) - \mathbb{H}(X|Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X)$$

But It is hard to find **real** distribution

Introduction - Background

Preliminaries

Denser Varadhan representation

$$D_{KL}(\mathbb{P} \,||\, \mathbb{Q}) \geq \sup_{T \in \mathcal{F}} \mathbb{E}_{\mathbb{P}}[T] - \log(\mathbb{E}_{\mathbb{Q}}[e^T]).$$

MINE

$$I(X;Z) \geq I_\Theta(X,Z), \quad I_\Theta(X,Z) = \sup_{\theta \in \Theta} \mathbb{E}_{\mathbb{P}_{XZ}}[T_\theta] - \log(\mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Z}[e^{T_\theta}]).$$

JSD

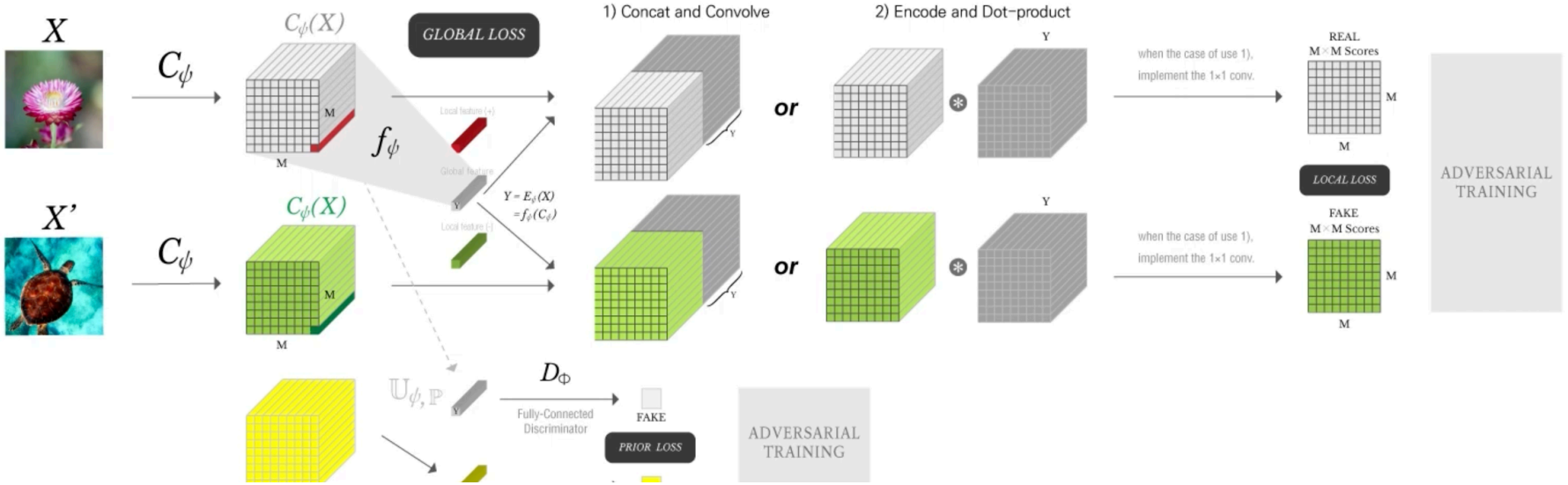
$$\widehat{\mathcal{I}}^{(\mathrm{JSD})}_{\omega,\psi}(X; E_\psi(X)) := \mathbb{E}_{\mathbb{P}}[-\mathrm{sp}(-T_{\psi,\omega}(x, E_\psi(x)))] - \mathbb{E}_{\mathbb{P}\times\tilde{\mathbb{P}}}[\mathrm{sp}(T_{\psi,\omega}(x', E_\psi(x)))],$$

infoNCE

$$\widehat{\mathcal{I}}^{(\mathrm{infoNCE})}_{\omega,\psi}(X; E_\psi(X)) := \mathbb{E}_{\mathbb{P}}\left[T_{\psi,\omega}(x, E_\psi(x)) - \mathbb{E}_{\tilde{\mathbb{P}}}\left[\log \sum_{x'} e^{T_{\psi,\omega}(x', E_\psi(x))}\right]\right]. \tag{5}$$

Introduction - Background

Deep Infomax



$$\arg\max_{\omega_1,\omega_2,\psi} \left(\alpha \widehat{\mathcal{I}}_{\omega_1,\psi}(X; E_\psi(X))\right) + \frac{\beta}{M^2} \sum_{i=1}^{M^2} \widehat{\mathcal{I}}_{\omega_2,\psi}(X^{(i)}; E_\psi(X))\Big) + \arg\min_{\psi} \arg\max_{\phi} \gamma \widehat{\mathcal{D}}_\phi(\mathbb{V} || \mathbb{U}_{\psi,\mathbb{P}}), \quad (8)$$

Maximize global MI | Maximize local MI | Match prior distribution

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximiza- tion. arXiv preprint arXiv:1808.06670, 2018.

Introduction - Related Work

Contrastive Learning

- Scoring function
- Train representation by contrast real and fake example

Sampling Strategies

- How to draw positive and negative samples
- Draw negative sample by sampling random pairs

Predictive Coding

- Suggest by Representation learning with contrastive predictive coding (CPC)
- 현재 분포와 미래 분포의 mutual information을 최대한 보존하는 방식으로 representation 학습

DGI Methodology

Graph-based unsupervised learning

Node Features : $X = \{x_1, x_2, \ldots, x_N\}$ $\quad x_n \in R^F$

Adjacency Matrix : $A \in R^{N \times N}$

Encoder $E : R^{N \times F} \times R^{N \times N} \rightarrow R^{N \times F'}$

High level representation: $E(X, A) = H = \{h_1, h_2 \ldots, h_N\}$ $\quad h_n \in R^{F'}$

h_i is a aggregation over local node neighbourhoods of i

Commonly use Encoder GCN

-> we refer h_i patch representation

DGI Methodology

Local Global Mutual Information Maximization

Readout function $\mathscr{R} : R^{N \times F'} \rightarrow R^{F'}$

Summary vectors : $\vec{s} = \mathscr{R}(E(X, A)) = \mathscr{R}(H)$

Discriminator $\mathscr{D} : R^{F'} \rightarrow R$

Probability scores patch-summary pair : $\mathscr{D}(h_i, \vec{s})$

Corruption function $C : R^{N \times F} \times R^{N \times N} \rightarrow R^{M \times F} \times R^{M \times M}$

Negative Sample : $(\tilde{X}, \tilde{A}) = C(X, A)$

Negative patch representation : $E(\tilde{X}, \tilde{A}) = \tilde{H} = \left\{\tilde{h_1}, \tilde{h_2} \ldots\right\}$

DGI Methodology

Local Global Mutual Information Maximization

$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N}\mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log \mathcal{D}\left(\vec{h}_i, \vec{s}\right)\right] + \sum_{j=1}^{M}\mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{A}})}\left[\log\left(1-\mathcal{D}\left(\vec{\widetilde{h}}_j, \vec{s}\right)\right)\right]\right) \quad (1)$$

Maximize probability scores between positive-summary pair

Minimize probability scores between negative-summary pair

DGI Methodology

Theoretical Motivation

Lemma 1. *Let* $\{\mathbf{X}^{(k)}\}_{k=1}^{|\mathbf{X}|}$ *be a set of node representations drawn from an empirical probability distribution of graphs,* $p(\mathbf{X})$*, with finite number of elements,* $|\mathbf{X}|$*, such that* $p(\mathbf{X}^{(k)}) = p(\mathbf{X}^{(k')})\ \forall k, k'$*. Let* $\mathcal{R}(\cdot)$ *be a deterministic readout function on graphs and* $\vec{s}^{(k)} = \mathcal{R}(\mathbf{X}^{(k)})$ *be the summary vector of the* k*-th graph, with marginal distribution* $p(\vec{s})$*. The optimal classifier between the joint distribution* $p(\mathbf{X}, \vec{s})$ *and the product of marginals* $p(\mathbf{X})p(\vec{s})$*, assuming class balance, has an error rate upper bounded by* $\text{Err}^* = \frac{1}{2}\sum_{k=1}^{|\mathbf{X}|} p(\vec{s}^{(k)})^2$*. This upper bound is achieved if* $\mathcal{R}$ *is injective.*

Corollary 1. *From now on, assume that the readout function used,* $\mathcal{R}$*, is injective. Assume the number of allowable states in the space of* $\vec{s}$*,* $|\vec{s}|$*, is greater than or equal to* $|\mathbf{X}|$*. Then, for* $\vec{s}^{\star}$*, the optimal summary under the classification error of an optimal classifier between the joint and the product of marginals, it holds that* $|\vec{s}^{\star}| = |\mathbf{X}|$*.*

DGI Methodology

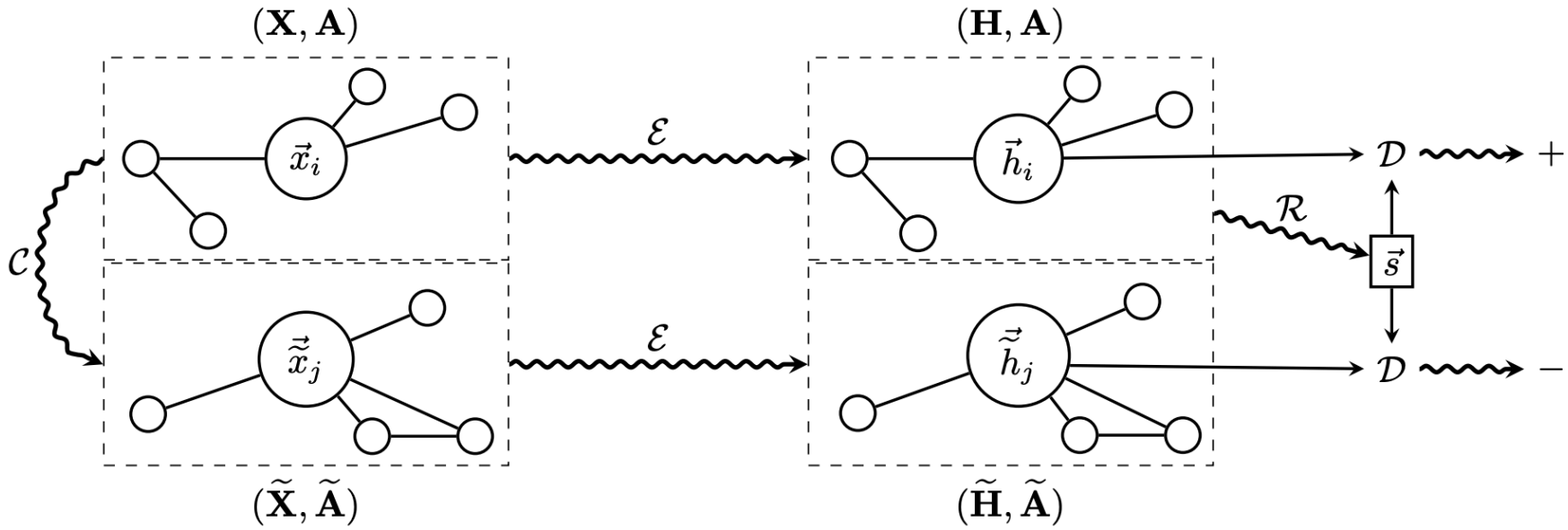
Theoretical Motivation

Theorem 1. $\vec{s}^{\star} = \text{argmax}_{\vec{s}} \text{MI}(\mathbf{X}; \vec{s})$, *where* MI *is mutual information.*

Theorem 2. *Let* $\mathbf{X}_i^{(k)} = \{\vec{x}_j\}_{j \in n(\mathbf{X}^{(k)}, i)}$ *be the neighborhood of the node* i *in the* k*-th graph that collectively maps to its high-level features,* $\vec{h}_i = \mathcal{E}(\mathbf{X}_i^{(k)})$*, where* n *is the neighborhood function that returns the set of neighborhood indices of node* i *for graph* $\mathbf{X}^{(k)}$*, and* $\mathcal{E}$ *is a deterministic encoder function. Let us assume that* $|\mathbf{X}_i| = |\mathbf{X}| = |\vec{s}| \geq |\vec{h}_i|$*. Then, the* $\vec{h}_i$ *that minimizes the classification error between* $p(\vec{h}_i, \vec{s})$ *and* $p(\vec{h}_i)p(\vec{s})$ *also maximizes* $\text{MI}(\mathbf{X}_i^{(k)}; \vec{h}_i)$.

DGI Methodology

Overview of DGI



DGI Methodology

Overview of DGI

1. Sample a negative example: $(\tilde{X}, \tilde{A}) = C(X, A)$
2. Obtain positive patch representation : $E(X, A) = H = \left\{h_1, h_2 \ldots, h_N\right\}$
3. Obtain negative patch representation : $E(\tilde{X}, \tilde{A}) = \tilde{H} = \left\{\tilde{h}_1, \tilde{h}_2 \ldots\right\}$
4. Create summary vector by readout function : $\vec{s} = \mathscr{R}(E(X, A)) = \mathscr{R}(H)$
5. Update Parameter $E, \mathscr{D}, \mathscr{R}$

Classification Performance

Dataset

Cora, Citeseer, Pubmed

- Classifying research papers
- Transductive learning

Reddit

- Predicting the community structure of a social network modelled with Reddit Post
- Inductive learning on large graphs

PPI

- Classifying protein roles within protein-protein interaction networks
- Inductive learning on Multiple graphs

Different experimental Setting!

Classification Performance

Experimental Setup - Transductive

- Use Encoder one-layer GCN

$$\mathcal{E}(\mathbf{X}, \mathbf{A}) = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{\Theta}\right)$$

- Applied parametric ReLU for activation
- For negative sample, preserve original Adjacency matrix, shuffle row-wise shuffling of X

Classification Performance

Experimental Setup - Inductive learning on large graphs

- Use Encoder mean-pooling propagation (GraphSAGE-GCN)

$$\mathbf{MP}(\mathbf{X}, \mathbf{A}) = \hat{\mathbf{D}}^{-1}\hat{\mathbf{A}}\mathbf{X}\mathbf{\Theta}$$

$$\widetilde{\mathbf{MP}}(\mathbf{X}, \mathbf{A}) = \sigma\left(\mathbf{X}\mathbf{\Theta}' \| \mathbf{MP}(\mathbf{X}, \mathbf{A})\right) \qquad \mathcal{E}(\mathbf{X}, \mathbf{A}) = \widetilde{\mathbf{MP}}_3(\widetilde{\mathbf{MP}}_2(\widetilde{\mathbf{MP}}_1(\mathbf{X}, \mathbf{A}), \mathbf{A}), \mathbf{A})$$

- While equation specifies adjacency and degree matrix, they are not needed
- Three-layer mean-pooling model with skip connection (|| is feature wise concatenation)

- Make subgraph centered by sampling node neighborhoods with replacement
- Use similar corruption function at transductive setting

Classification Performance

Experimental Setup - Inductive learning on multiple graphs

- Use three-layer mean-pooling model with dense skip connections

$$\mathbf{H}_1 = \sigma\left(\mathrm{MP}_1(\mathbf{X}, \mathbf{A})\right)$$
$$\mathbf{H}_2 = \sigma\left(\mathrm{MP}_2(\mathbf{H}_1 + \mathbf{X}\mathbf{W}_{\text{skip}}, \mathbf{A})\right)$$
$$\mathcal{E}(\mathbf{X}, \mathbf{A}) = \sigma\left(\mathrm{MP}_3(\mathbf{H}_2 + \mathbf{H}_1 + \mathbf{X}\mathbf{W}_{\text{skip}}, \mathbf{A})\right)$$

- W_{skip} is a learnable projection matrix
- For negative sample, randomly sample a different graph from the training set
- Apply dropout

Classification Performance

Experimental Setup - Additional training details

- For the readout function, use simple averaging of all the node's features (can be replaced by set2vec or DiffPool)

$$\mathcal{R}(\mathbf{H}) = \sigma\left(\frac{1}{N}\sum_{i=1}^{N}\vec{h}_i\right)$$

- For discriminator, applying simple bilinear scoring function (learnable)

$$\mathcal{D}(\vec{h}_i, \vec{s}) = \sigma\left(\vec{h}_i^T \mathbf{W} \vec{s}\right)$$

- Glorot initialization with Adam optimizer

Classification Performance

Result

Transductive

Available data	Method	Cora	Citeseer	Pubmed
X	Raw features	47.9 ± 0.4%	49.3 ± 0.2%	69.1 ± 0.3%
A, **Y**	LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
A	DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
X, **A**	DeepWalk + features	70.7 ± 0.6%	51.4 ± 0.5%	74.3 ± 0.9%
X, **A**	Random-Init (ours)	69.3 ± 1.4%	61.9 ± 1.6%	69.6 ± 1.9%
X, **A**	**DGI** (ours)	**82.3** ± 0.6%	**71.8** ± 0.7%	**76.8** ± 0.6%
X, **A**, **Y**	GCN (Kipf & Welling, 2016a)	81.5%	70.3%	79.0%
X, **A**, **Y**	Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%

Inductive

Available data	Method	Reddit	PPI
X	Raw features	0.585	0.422
A	DeepWalk (Perozzi et al., 2014)	0.324	—
X, **A**	DeepWalk + features	0.691	—
X, **A**	GraphSAGE-GCN (Hamilton et al., 2017a)	0.908	0.465
X, **A**	GraphSAGE-mean (Hamilton et al., 2017a)	0.897	0.486
X, **A**	GraphSAGE-LSTM (Hamilton et al., 2017a)	0.907	0.482
X, **A**	GraphSAGE-pool (Hamilton et al., 2017a)	0.892	0.502
X, **A**	Random-Init (ours)	0.933 ± 0.001	0.626 ± 0.002
X, **A**	**DGI** (ours)	**0.940** ± 0.001	**0.638** ± 0.002
X, **A**, **Y**	FastGCN (Chen et al., 2018)	0.937	—
X, **A**, **Y**	Avg. pooling (Zhang et al., 2018)	0.958 ± 0.001	0.969 ± 0.002

Qualitative Analysis

Figure 3: t-SNE embeddings of the nodes in the Cora dataset from the raw features (**left**), features from a randomly initialized DGI model (**middle**), and a learned DGI model (**right**). The clusters of the learned DGI model's embeddings are clearly defined, with a Silhouette score of 0.234.

Implementation

```python
def load_dataset(args):

    if args.dataset_name == 'Cora':
        dataset = Planetoid(root='/tmp/Cora', name='Cora')
    elif args.dataset_name == 'Pubmed':
        dataset = Planetoid(root='/tmp/Pubmed', name='Pubmed')
    elif args.dataset_name == 'Citeseer':
        dataset = Planetoid(root='/tmp/Citeseer', name='Citeseer')
    else:
        raise ValueError('Dataset name must be one of Cora, Pubmed, Citeseer')

    graph = dataset[0]
    adj = csr_matrix((np.ones(graph.edge_index.shape[1]), (graph.edge_index[0].numpy(), graph.edge_index[1].numpy())))
    feature = graph.x.numpy()

    return adj, feature, graph
```

```python
def sparse_mx_to_torch(sparse_mx):
    sparse_mx = sparse_mx.tocoo().astype(np.float32)
    indices = torch.from_numpy(
        np.vstack((sparse_mx.row, sparse_mx.col)).astype(np.int64))
    values = torch.from_numpy(sparse_mx.data)
    shape = torch.Size(sparse_mx.shape)
    return torch.sparse.FloatTensor(indices, values, shape)
```

```python
def preprocess_adj_feature(adj, feature):

    adj = sp.coo_matrix(adj)
    rowsum = np.array(adj.sum(1))
    d_inv_sqrt = np.power(rowsum, -0.5).flatten()
    d_inv_sqrt[np.isinf(d_inv_sqrt)] = 0.
    d_mat_inv_sqrt = sp.diags(d_inv_sqrt)
    adj = adj.dot(d_mat_inv_sqrt).transpose().dot(d_mat_inv_sqrt).tocoo()
    adj = adj + sp.eye(adj.shape[0])

    adj = sparse_mx_to_torch(adj)

    rowsum = np.sum(feature, axis=1)
    rowsum_diag = np.diag(rowsum)
    rowsum_inv = np.power(rowsum_diag, -1)
    rowsum_inv[np.isinf(rowsum_inv)] = 0.0
    feature = np.dot(rowsum_inv, feature)

    feature = torch.FloatTensor(feature[np.newaxis])

    return adj, feature
```

Implementation

```python
class GCN(nn.Module):
    def __init__(self, in_features, out_features):
        super(GCN, self).__init__()
        self.linear = nn.Linear(in_features, out_features)
        self.PRelu = nn.PReLU()

        nn.init.xavier_uniform_(self.linear.weight.data)

    def forward(self, x, a):
        xtheta = self.linear(x)
        output = torch.unsqueeze(torch.spmm(a, torch.squeeze(xtheta, 0)), 0)
        output = self.PRelu(output)

        return output

class Discriminator(nn.Module):
    def __init__(self, in1_features, in2_features, out_features):
        super(Discriminator, self).__init__()
        self.linear = nn.Bilinear(in1_features, in2_features, out_features)
        # self.logsigmoid = nn.LogSigmoid()

        nn.init.xavier_uniform_(self.linear.weight.data)

    def forward(self, hi, s):


        hiWs = self.linear(hi, s)
        # output = self.logsigmoid(hiWs)

        return hiWs

class Readout(nn.Module):
    def __init__(self):
        super(Readout, self).__init__()
        # self.logsigmoid = nn.LogSigmoid()

    def forward(self, H):
        output = torch.mean(H, dim=1)
        # output = self.logsigmoid(output)

        return output
```

```python
class DGI(nn.Module):
    def __init__(self, in_features, hidden_dim):
        super(DGI, self).__init__()
        self.GCN = GCN(in_features, hidden_dim)
        self.Discriminator = Discriminator(hidden_dim, hidden_dim, 1)
        self.Readout = Readout()
        self.Sigmoid = nn.Sigmoid()

    def forward(self, pos, neg, a):
        pos_H = self.GCN(pos, a)
        neg_H = self.GCN(neg, a)

        s = self.Readout(pos_H)
        s = self.Sigmoid(s)
        s = torch.unsqueeze(s, 1).expand_as(pos_H)

        pos_score = self.Discriminator(pos_H, s).squeeze(2)
        neg_score = self.Discriminator(neg_H, s).squeeze(2)
        logits = torch.cat((pos_score, neg_score), dim=1)

        return logits

    def get_embedding(self, x, a):
        return self.GCN(x, a)
```

Implementation

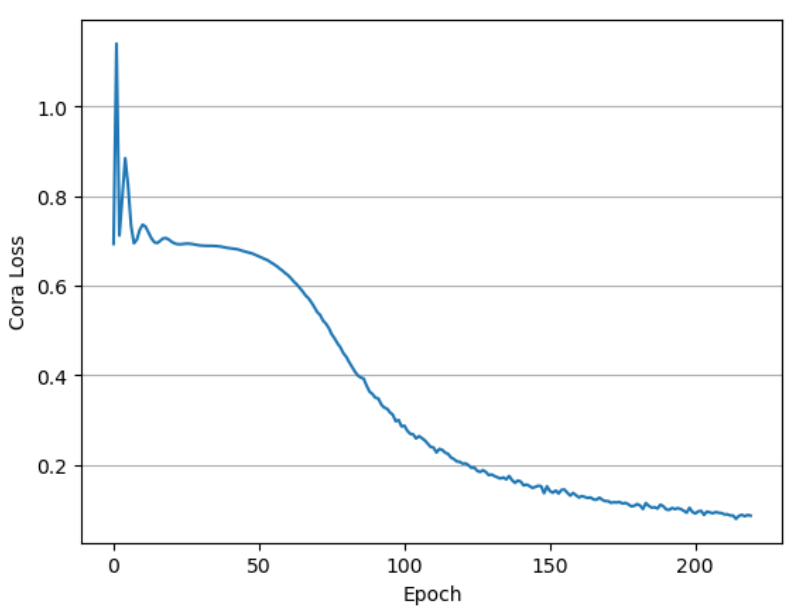
```python
def train(args, model, criterion, optimizer, adj, feature, num_node):
    summary = pd.DataFrame(columns=['Epoch', 'Loss'])
    best_loss = 1000000

    for epoch in range(args.epochs):
        model.train()
        optimizer.zero_grad()

        pos_lb = torch.ones(args.batch_size, num_node)
        neg_lb = torch.zeros(args.batch_size, num_node)
        pos_neg_lb = torch.cat((pos_lb, neg_lb), dim=1)

        corrupted_node = np.random.permutation(num_node)
        corrupted_features = feature[:, corrupted_node, :]
        logit = model(feature, corrupted_features, adj)
        loss = criterion(logit, pos_neg_lb)

        if best_loss > loss.item():
            best_loss = loss.item()
            torch.save(model.state_dict(), f'best_{args.dataset_name}.pt')

        loss.backward()
        optimizer.step()

        print('Epoch: {:03d}, Loss: {:.4f}'.format(epoch, loss.item()))

        summary = pd.concat([summary, pd.DataFrame([[epoch, loss.item()]], columns=['Epoch', 'Loss'])], ignore_index=True)

    summary.to_csv(f'{args.dataset_name}_summary.csv', index=False)
```

```python
def test(args, embedding, label):
    result = pd.DataFrame(columns=['Dataset', 'Accuracy', 'Macro-F1', 'Micro-F1'])
    train_x, test_x, train_y, test_y = train_test_split(embedding, label, test_size=args.test_size, random_state=42)

    LR = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial')
    LR.fit(train_x, train_y)

    pred = LR.predict(test_x)
    acc = accuracy_score(test_y, pred)
    macro_f1 = f1_score(test_y, pred, average='macro')
    micro_f1 = f1_score(test_y, pred, average='micro')

    print('Accuracy: {:.4f}, Macro-F1: {:.4f}, Micro-F1: {:.4f}'.format(acc, macro_f1, micro_f1))
    result = pd.concat([result, pd.DataFrame([[args.dataset_name, acc, macro_f1, micro_f1]], columns=['Dataset', 'Accuracy', 'Macro-F1', 'Micro-F1'])],

    result.to_csv(f'{args.dataset_name}_result.csv', index=False)
```

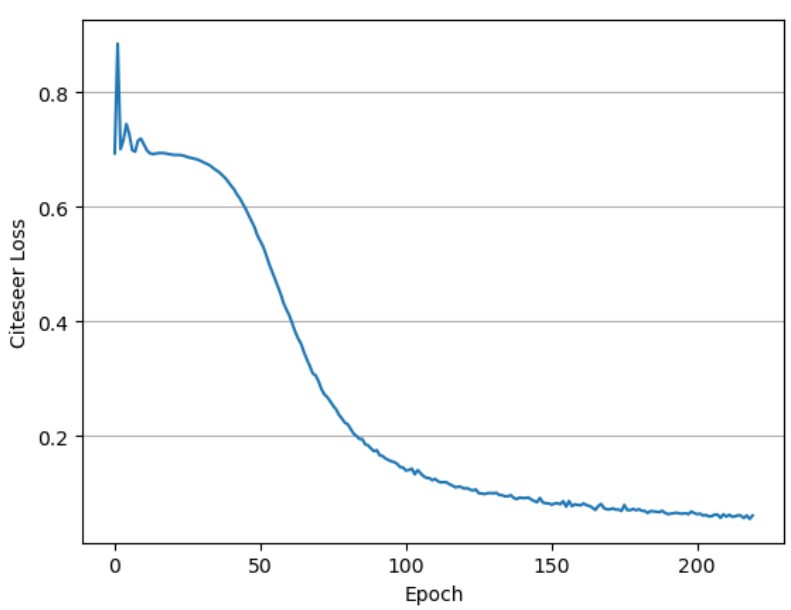
```python
def main(args):
    adj, feature, graph = load_dataset(args)
    adj, feature = preprocess_adj_feature(adj, feature)

    num_node, num_feature = graph.x.size()

    model = DGI(num_feature, args.hidden_dim)

    optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)
    criterion = nn.BCEWithLogitsLoss()

    train(args, model, criterion, optimizer, adj, feature, num_node)

    model.load_state_dict(torch.load(f'best_{args.dataset_name}.pt'))

    embedding = model.get_embedding(feature, adj)
    embedding = embedding.squeeze(0).detach().numpy()
    label = graph.y.numpy()

    test(args, embedding, label)
```

```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='DGI')

    parser.add_argument('--dataset_name', type=str, default='Cora', help='Dataset name')

    parser.add_argument('--epochs', type=int, default=200, help='Number of epochs to train')
    parser.add_argument('--batch_size', type=int, default=1, help='Number of batch size')
    parser.add_argument('--lr', type=float, default=0.001, help='Learning rate')
    parser.add_argument('--hidden_dim', type=int, default=512, help='Hidden dimension')
    parser.add_argument('--test_size', type=float, default=0.2, help='Test size ratio')

    args = parser.parse_args()

    main(args)
```

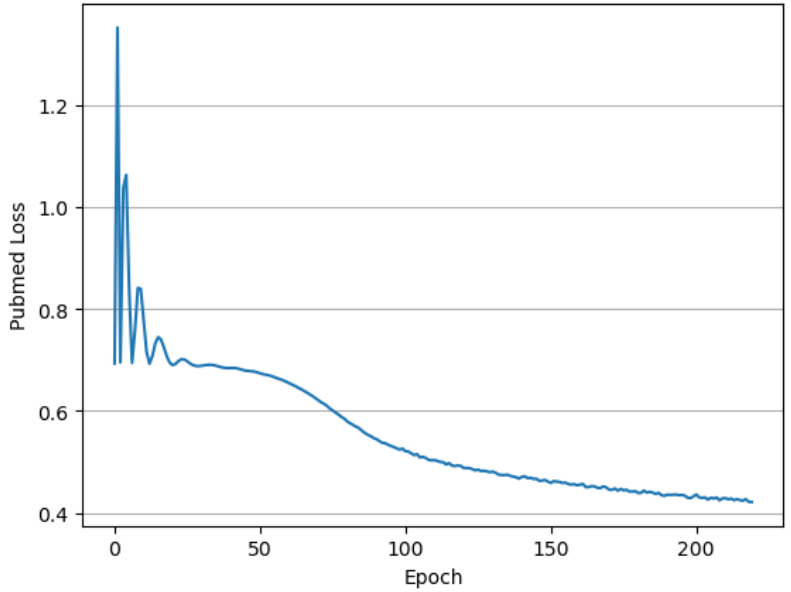
Implementation



Cora



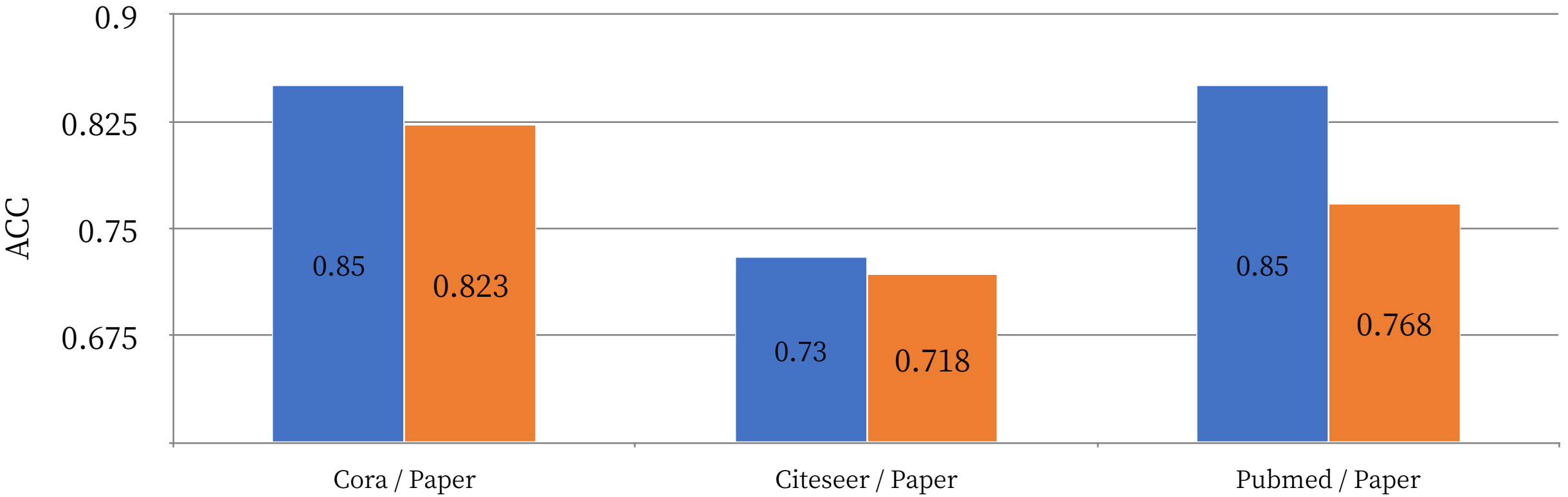
Citeseer



Pubmed

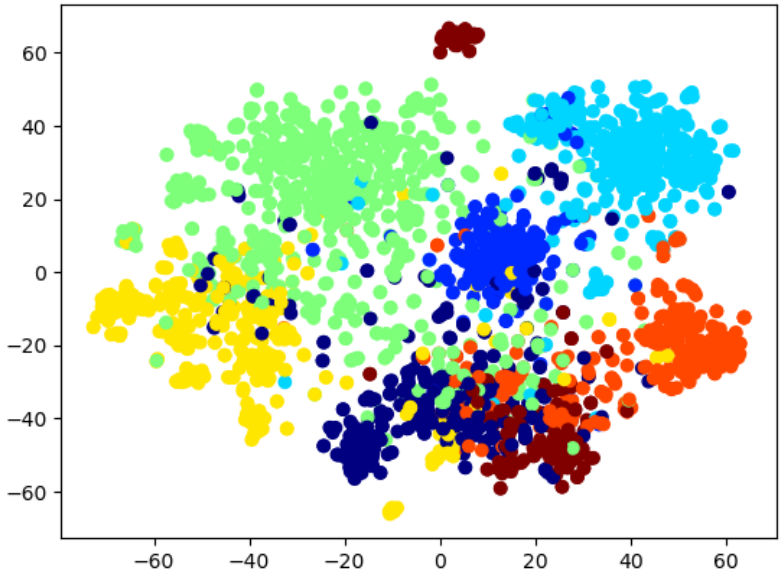
Implementation

Result

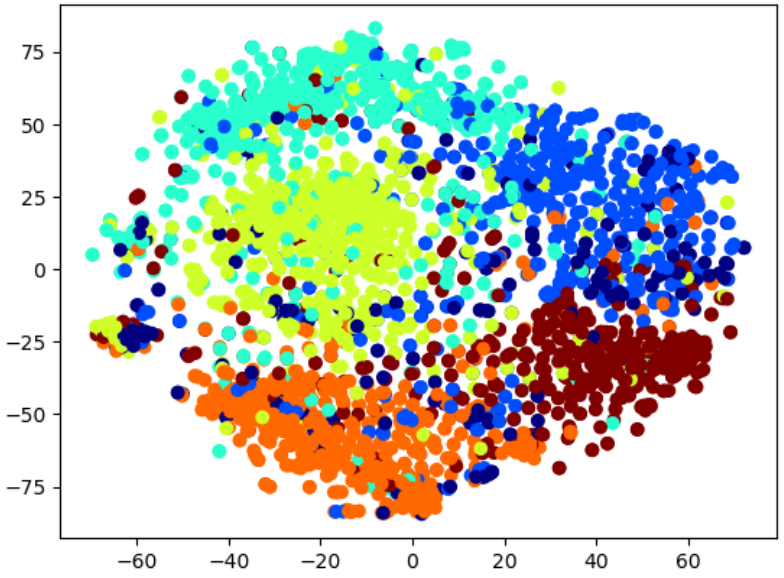


Implementation

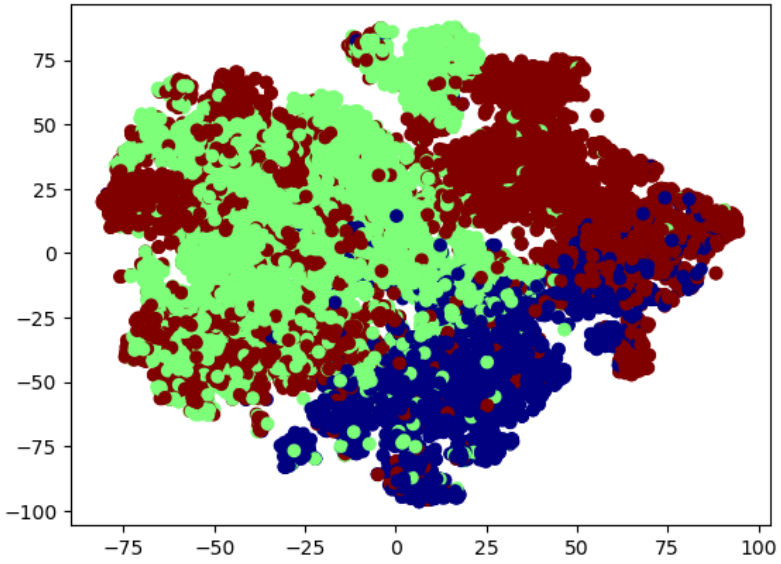
Result



Cora



Citeseer



Pubmed

Review

(Highlight)

- Performance is similar to results of paper

(Question)

- Global ?

(Insight)

- Readout function
- Corrupt function

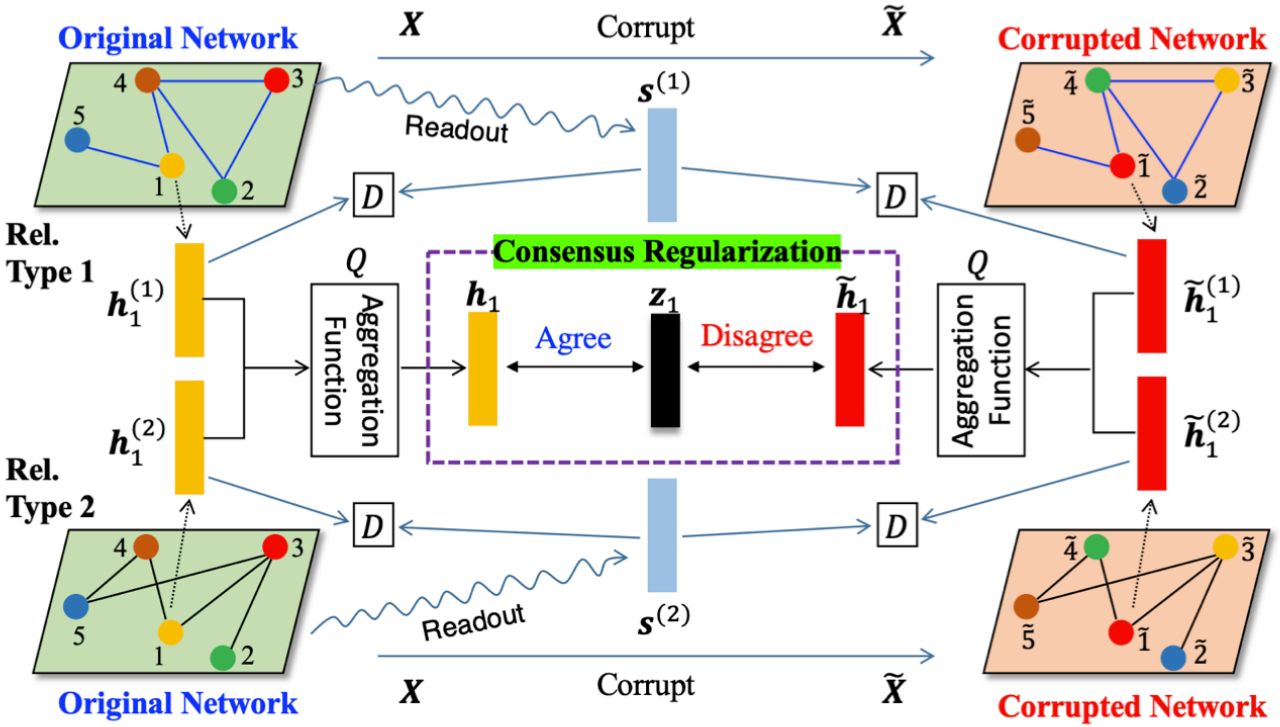
Review

DMGI

- Jointly integrate the embeddings from multiple types of relations between nodes

- Use consensus regularization framework and universal discriminator

- If use attention mechanism, It outperforms most dataset (except IMDB)

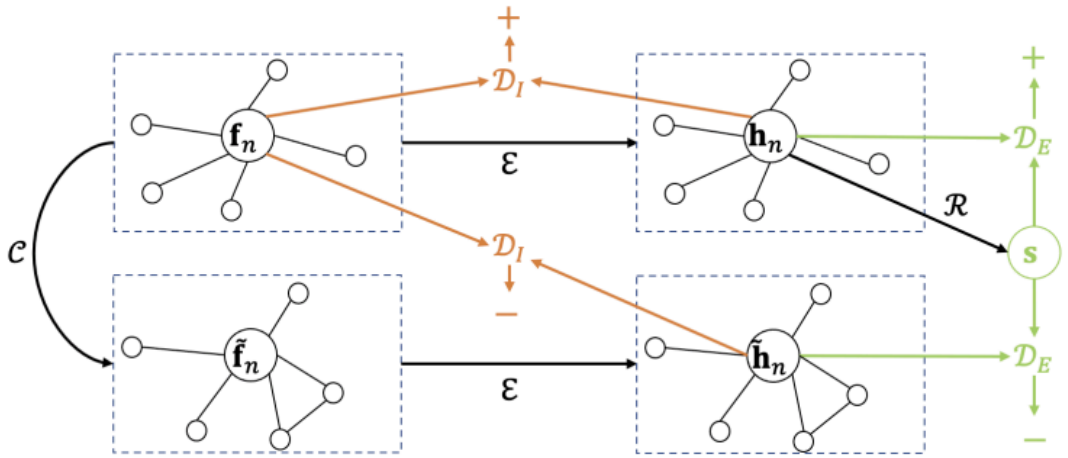


Chanyoung Park, Donghyun Kim, Jiawei Han, Hwanjo Yu. 2019. Unsupervised Attributed Multiplex Network Embedding. (AAAI '19),

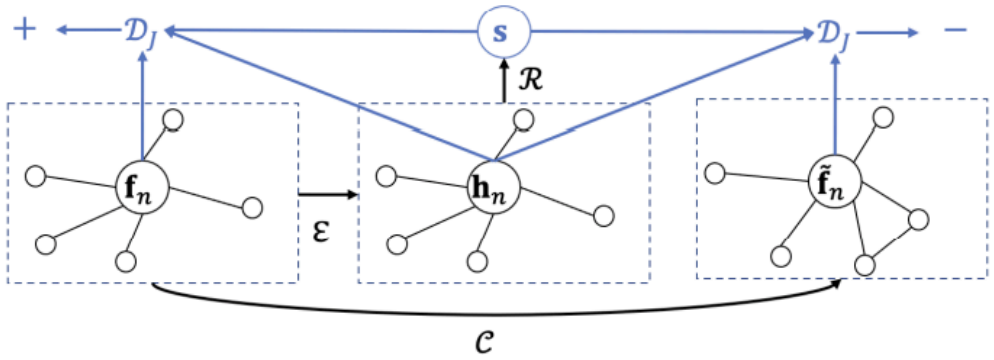
Review

HDMI

- Use high order mutual information
- Simultaneously captures the extrinsic signal, the intrinsic signal
- Combine node embedding from different layers



(a) Illustration of the extrinsic and intrinsic supervision.



(b) Illustration of the joint supervision.

Baoyu Jing, Chanyoung Park, and Hanghang Tong. 2021. HDMI: Highorder Deep Multiplex Infomax. In Proceedings of the Web Conference 2021 (WWW '21),

Review

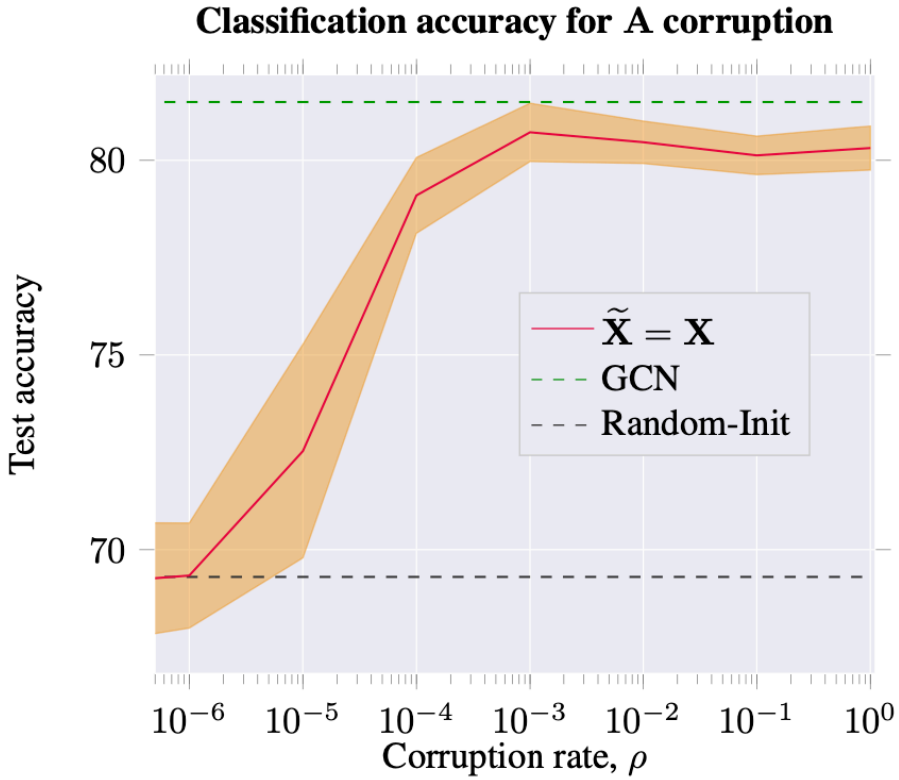
How about corrupted adjacency matrix?

$$\Sigma_{ij} \sim \text{Bernoulli}(\rho)$$

$$\widetilde{\mathbf{A}} = \mathbf{A} \oplus \mathbf{\Sigma}$$

Corrupted adjacency matrix - Cora ACC 0.84

```python
def preprocess_adj_feature(adj):

    pos_adj = sp.coo_matrix(adj)
    rowsum = np.array(pos_adj.sum(1))
    d_inv_sqrt = np.power(rowsum, -0.5).flatten()
    d_inv_sqrt[np.isinf(d_inv_sqrt)] = 0.
    d_mat_inv_sqrt = sp.diags(d_inv_sqrt)
    pos_adj = pos_adj.dot(d_mat_inv_sqrt).transpose().dot(d_mat_inv_sqrt).tocoo()
    pos_adj = pos_adj + sp.eye(pos_adj.shape[0])

    pos_adj = sparse_mx_to_torch(pos_adj)

    corrupted_adj = np.random.binomial(1, 0.001, size=adj.shape)
    corrupted_adj = corrupted_adj + adj
    corrupted_adj = np.where(corrupted_adj > 1, 1, corrupted_adj)

    neg_adj = sp.coo_matrix(adj)
    rowsum = np.array(neg_adj.sum(1))
    d_inv_sqrt = np.power(rowsum, -0.5).flatten()
    d_inv_sqrt[np.isinf(d_inv_sqrt)] = 0.
    d_mat_inv_sqrt = sp.diags(d_inv_sqrt)
    neg_adj = neg_adj.dot(d_mat_inv_sqrt).transpose().dot(d_mat_inv_sqrt).tocoo()
    neg_adj = neg_adj + sp.eye(pos_adj.shape[0])

    neg_adj = sparse_mx_to_torch(neg_adj)

    return pos_adj, neg_adj
```



Review

Using learnable readout function - Cora ACC 0.84

```python
class CustomReadout(nn.Module):
    def __init__(self, in_feature):
        super(CustomReadout, self).__init__()
        self.linear = nn.Linear(in_feature, 1)

    def forward(self, H):
        H = H.transpose(2, 1)
        output = self.linear(H)

        return output
```

How about discriminate between summary vector?

$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N}\mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log \mathcal{D}\left(\vec{h}_i, \vec{s}\right)\right] + \sum_{j=1}^{M}\mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{A}})}\left[\log\left(1-\mathcal{D}\left(\left(\vec{h}_i, \vec{\tilde{s}}\right)\right)\right)\right]\right) \qquad (1)$$

Only work with learnable readout function - Cora ACC 0.835

```python
class Custom_DGI(nn.Module):
    def __init__(self, in_features, hidden_dim, num_node):
        super(Custom_DGI, self).__init__()
        self.GCN = GCN(in_features, hidden_dim)
        self.Discriminator = Discriminator(hidden_dim, hidden_dim, 1)
        self.Readout = CustomReadout(num_node, require_grad = False)
        self.Sigmoid = nn.Sigmoid()

    def forward(self, pos, neg, a):
        pos_H = self.GCN(pos, a)
        neg_H = self.GCN(neg, a)

        pos_s = self.Readout(pos_H)
        pos_s = self.Sigmoid(pos_s)
        pos_s = pos_s.transpose(1,2).expand_as(pos_H)

        neg_s = self.Readout(neg_H)
        neg_s = self.Sigmoid(neg_s)
        neg_s = neg_s.transpose(1,2).expand_as(pos_H)

        pos_score = self.Discriminator(pos_H, pos_s).squeeze(2)
        neg_score = self.Discriminator(pos_H, neg_s).squeeze(2)
        logits = torch.cat((pos_score, neg_score), dim=1)

        return logits

    def get_embedding(self, x, a):
        return self.GCN(x, a)
```

Appendix

Mutual Information

$$I(X;Z) = H(X) - H(X|Z)$$

$$= D_{KL}\left(P(X,Z) \,\|\, P(X)\cdot P(Z)\right) \rightarrow D_{KL}(P\|Q) = E_P\left[\log \frac{P(X)}{Q(X)}\right]$$

$$= \int P(x) \log \frac{P(x)}{Q(x)}\, dx$$

Donsker-Varadhan representation

$$D_{KL}(P\|Q) \geq \sup_{T\in F} E_P[T] - \log\left(E_Q[e^T]\right)$$

$T \Rightarrow$ function $\Omega \rightarrow \mathbb{R}$

$$Z = E_Q[e^T] \qquad dG = \frac{1}{Z} e^T dQ$$

$$\log \frac{dG}{dQ} = \log \frac{1}{Z} e^T = \log \frac{1}{Z} + T = T - \log Z$$

$$E_P[T] - \log Z = E_P\left[\log \frac{dG}{dQ}\right]$$

$$D \leq \sup A = D_{KL}(P\|Q) - \left[E_P[T] - \log\left(E_Q[e^T]\right)\right]$$

$$= E_P\left[\log \frac{dP}{dQ}\right] - E_P\left[\log \frac{dG}{dQ}\right] = E_P\left[\log \frac{dP}{dG}\right]$$

$$= D_{KL}(P\|G) \geq 0$$

$$I(X;Z) \geq I_\theta(X,Z) = \sup E_{P_{XZ}}[T_\theta] - \log\left(E_{P_X \times P_Z}[e^{T_\theta}]\right)$$

$$I(X;Z) = D_{KL}\left(P(X,Z) \,\|\, P(X)\cdot P(Z)\right)$$

$$D_{KL}(P\|Q) \geq \sup_{T\in F} E_P[T] - \log\left(E_Q[e^T]\right)$$

MINE Lemma 1

$$|I(X,Z) - I_\theta(X,Z)| \leq \epsilon, \quad a.e.$$

If $T^* = \log \frac{dP}{dQ}$ $= D_{KL}(P\|Q)$ optimal!

$$\Rightarrow E_P[T^*] = I(X,Z), \quad E_Q[e^{T^*}] = 1$$

$$\Rightarrow E_Q\left[e^{\log \frac{dP}{dQ}}\right] = E_Q\left[\frac{dP}{dQ}\right] = \int_\Omega \frac{P(x)}{Q(x)} dx = 1$$

$$Gap = I(X,Z) - \hat{I}(T) = E_P[T^* - T] + \log E_Q[e^T]$$

$$\leq E_P[T^* - T] + E_Q[e^T - 1] \quad \log x \leq x - 1$$

$$= E_P[T^* - T] + E_Q[e^T - e^{T^*}] \quad \log e^T \leq e^T - 1$$

① If $T_\theta \leq M$, $E_P|T^* - T_\theta| \leq \frac{\eta}{2}$, $E_Q|T^* - T_\theta| \leq \frac{\eta}{2} e^{-M}$

$$\Rightarrow E_Q|e^{T^*} - e^{T_\theta}| \leq e^M E_Q|T^* - T_\theta| \leq \frac{\eta}{2}$$

$$T^* \leq M, \quad T_\theta \leq M$$

$$e^{T^*} \leq e^M \cdot T^*$$

$$e^{T_\theta} \leq e^M \cdot T_\theta$$

$$|I(X,Z) - \hat{I}(T_\theta)| \leq E_P|T^* - T_\theta| + E_Q|e^{T^*} - e^{T_\theta}| \leq \frac{\eta}{2} + \frac{\eta}{2} \leq \eta$$

Appendix

Lemma 1

$Q^{(k)}$ = all possible set X^i that $R(X^{(i)}) = \vec{S}^{(k)}$

$P(\vec{S}^{(k)}) P(X^{(k)}) = P(\vec{S}^{(k)}) \sum_{S} P(X^{(k)}, \vec{S}^{(k)})$

$= P(\vec{S}^{(k)}) P(X^{(k)}, \vec{S}^{(k)})$

$= P(\vec{S}^{(k)}) P(X^{(k)} | \vec{S}^{(k)}) P(\vec{S}^{(k)})$

$= \frac{P(X^{(k)})}{\sum_{X^i \in Q^{(k)}} P(X^i)} P(\vec{S}^{(k)})^2$

$= \rho^{(k)} P(\vec{S}^{(k)})^2$ $\quad X^{(k)} \in Q^{(k)}$ $\quad \rho^{(k)} \le 1$

when $Q^{(k)} = \{X^{(k)}\}$ $\quad \rho^{(k)} = 1$

(일대일 대응)

Corollary 1.

$\vec{S}^{(k)} = \arg\min_{\vec{S}} Err^*$, $\quad Err^* \le \frac{1}{2} \sum_{k=1}^{|X|} P(\vec{S}^{(k)})^2$

Err^* is geometric sum

⇒ when $P(\vec{S}^{(k)})$ is uniform, H is minimized

$R(\cdot)$ is deterministic (입력이 같으면, return이 같음)

$|\vec{S}^{(k)}| = |X|$

Theorem 1

$I(X;S) = H(X) - H(X|\vec{S}) \le H(X)$

$= I(X;X)$

$X \to R(X)$ is injective $\quad = I(X;R(X))$

$|X| = |\vec{S}^{(k)}|$ $\quad = I(X;\vec{S}^{(k)})$

$R^{-1}(\vec{S}) = X$

Theorem 2

If $|X_i| = |\vec{S}|$, $\quad X_i = R^{-1}(\vec{S})$ $\quad (\Rightarrow h_i = E(R^{-1}(\vec{S}))$

Optimal classifier $P(h_i, \vec{S})$, $P(h_i)P(S) \Rightarrow Err^* = \frac{1}{2} \sum_{k=1}^{|X|} P(h_i^{(k)})^2$

$\Rightarrow |h_i| = |X_i|$

by Theorem 1, $\vec{h}_i^* = \arg\max_{\vec{h}_i} MI(X_i; \vec{h}_i)$