Deep Graph Infomax

Petar Veličković et al.

안중찬

2023. 02. 07.

CONTENTS

01 INTRODUCTION

- Unsupervised graph learning ⇨ **Random walk-based** algorithms are dominant.

(nodes that are 'close' → 'close' in the representation space)

Known limitations

1. Over-emphasize proximity information at the expense of structural information
2. Performance is highly dependent on hyperparameter choice
3. Unclear whether random-walk objectives actually provide any useful signal with stronger encoder models

Alternative objective for unsupervised graph learning :

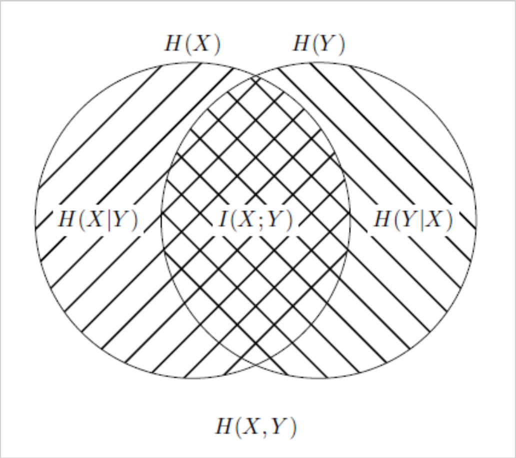
Mutual Information !

01 INTRODUCTION

· Mutual Information

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

$$I(X;Y) = \int_Y \int_X p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right) dx\, dy$$



cf. Donsker-Varadhan representation

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

Source:
https://arxiv.org/abs/1304.2333, Felix Effenberger, "A primer on information theory, with applications to neuroscience"
https://arxiv.org/abs/1801.04062, Mohamed Ishmael Belghazi et al., "Mutual Information Neural Estimation"

01 INTRODUCTION

Mutual Information Neural Estimation(MINE, Belghazi et al., 2018)

- Statistical network as a classifier
- Joint distribution of two random variables and their product of marginals

⇩

Deep InfoMax(DIM, Hjelm et al., 2018)

- Representation learning of high-dimensional data
- Maximize the mutual information between a “global” representation and “local” parts of the input

⇩ (to the graph domain)

Deep Graph InfoMax (DGI)

01 INTRODUCTION

· Related Work

<u>Contrastive methods</u>	Sampling strategies	Predictive coding
- *scoring function* - increase the score on “real” input (positive examples) - decrease the score on “fake” input (negative samples)	- how to draw positive and negative samples - local contrastive loss - positive = short random walks - negative = sampling random pairs	- Contrastive predictive coding: structurally-specified parts - contrast global/local parts of a graph simultaneously - matrix factorization-style losses

02 DGI METHODOLOGY

2.1 Graph-based unsupervised learning

Set of node features, $\boldsymbol{X} = \{\vec{x}_1, \vec{x}_2, \vec{x}_3, \dots, \vec{x}_N\}$

Adjacency matrix, $\boldsymbol{A} \in \mathbb{R}^{N*N}$ $\quad A_{ij} = 1\ if\ there\ exists\ an\ edge\ i \rightarrow j\ in\ the\ graph,\ \ A_{ij} = 0\ otherwise$

Encoder, $\mathcal{E}\colon \mathbb{R}^{N*F} \times \mathbb{R}^{N*N} \rightarrow \mathbb{R}^{N*F'}$, $\mathcal{E}(\boldsymbol{X}, \boldsymbol{A}) = \boldsymbol{H} = \{\vec{h}_1, \vec{h}_2, \vec{h}_3, \dots, \vec{h}_N\}$

2.2 Local-Global mutual information maximization

Summary vectors, $\vec{\boldsymbol{s}} = \mathcal{R}\big(\mathcal{E}(X, A)\big)$

Readout function, $\mathcal{R} : \mathbb{R}^{N*F} \rightarrow \mathbb{R}^{F}$

Discriminator, $\mathcal{D} : \mathbb{R}^{F} \times \mathbb{R}^{F} \rightarrow \mathbb{R}$ $\quad proxy\ for\ maximizing\ the\ local\ mutual\ information$

Corruption function, $\mathcal{C} : \mathbb{R}^{N*F} \times \mathbb{R}^{N*N} \rightarrow \mathbb{R}^{M*F} \times \mathbb{R}^{M*M}$

Negative samples: $\mathcal{C}(\boldsymbol{X}, \boldsymbol{A}) = (\widetilde{\boldsymbol{X}}, \widetilde{\boldsymbol{A}})$, $\overrightarrow{\widetilde{h}_j}$

objective (Jensen-Shannon divergence)

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

02 DGI METHODOLOGY

2.3 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* $\mathrm{Err}^* = \frac{1}{2}\sum_{k=1}^{|\mathbf{X}|} p(\vec{s}^{(k)})^2$*. 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* $|\vec{s}^{\star}| = |\mathbf{X}|$.

02 DGI METHODOLOGY

2.3 Theoretical Motivation

Theorem 1. $\vec{s}^{\star} = \operatorname{argmax}_{\vec{s}} \mathrm{MI}(\mathbf{X}; \vec{s})$, *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* $\mathrm{MI}(\mathbf{X}_i^{(k)}; \vec{h}_i)$.

- Classifier between samples from the joint and the product of marginals

- Binary cross-entropy(BCE) loss to optimize

02 DGI METHODOLOGY

2.4 Overview of DGI

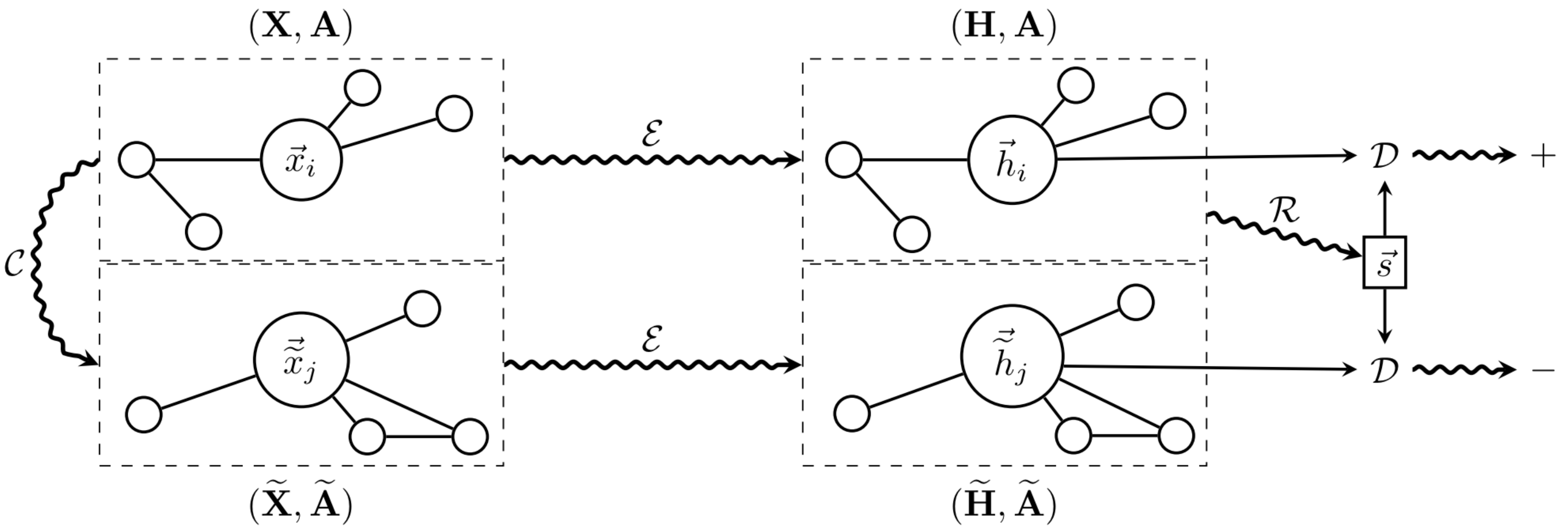


Figure 1: A high-level overview of Deep Graph Infomax. Refer to Section 3.4 for more details.

03 CLASSIFICATION PERFORMANCE

3.1 Dataset

Dataset	Task	Nodes	Edges	Features	Classes	Train/Val/Test Nodes
Cora	Transductive	2,708	5,429	1,433	7	140/500/1,000
Citeseer	Transductive	3,327	4,732	3,703	6	120/500/1,000
Pubmed	Transductive	19,717	44,338	500	3	60/500/1,000
Reddit	Inductive	231,443	11,606,919	602	41	151,708/23,699/55,334
PPI	Inductive	56,944 (24 graphs)	818,716	50	121 (multilbl.)	44,906/6,514/5,524 (20/2/2 graphs)

- Transductive learning:

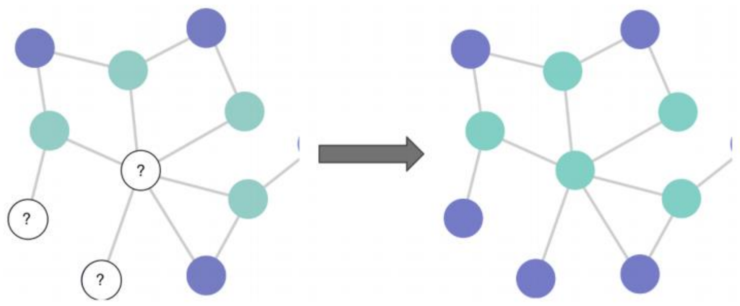
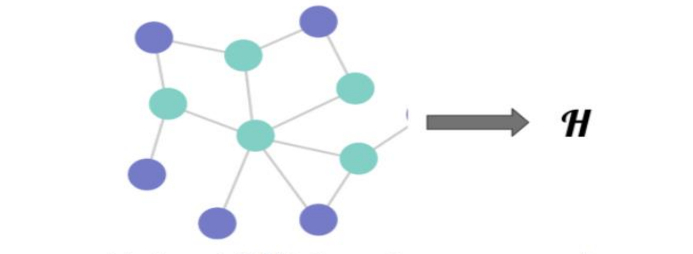


Figure 1. Node classification in transductive setting. At training time, the learning algorithm has access to all the nodes and edges including nodes for which labels are to be predicted.

- Inductive learning:



(a) A model $\mathcal{H}$ is learned over some graph

Source: https:// https://arxiv.org/abs/2001.07524, Pushkar Mishra et al., "Node Masking: Making Graph Neural Networks Generalize and Scale

03 CLASSIFICATION PERFORMANCE

3.2 Experimental Setup – encoder & corruption functions

(1) Transductive learning(Cora, Citeseer, and Pubmed)

one-layer GCN

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

$\widehat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$

$\widehat{\boldsymbol{D}}$: $degree\ matrix$

σ : $parametric\ ReLU(PReLU)$

$\boldsymbol{\Theta}$: $learnable\ linear\ transformation$

$F' = 512\ (\ 256\ on\ Pubmed)$

- preserves the original adjacency matrix ($\widetilde{\boldsymbol{A}} = \boldsymbol{A}$)
- corrupted features($\widetilde{\boldsymbol{X}}$) obtained by row-wise shuffling of $\boldsymbol{X}$

03 CLASSIFICATION PERFORMANCE

3.2 Experimental Setup – encoder & corruption functions

(2) Inductive learning on large graphs(Reddit)

three-layer mean-pooling model with skip connections

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

$$\widetilde{\mathrm{MP}}(\mathbf{X}, \mathbf{A}) = \sigma\left(\mathbf{X}\mathbf{\Theta}' \| \mathrm{MP}(\mathbf{X}, \mathbf{A})\right)$$

$$\mathcal{E}(\mathbf{X}, \mathbf{A}) = \widetilde{\mathrm{MP}}_3(\widetilde{\mathrm{MP}}_2(\widetilde{\mathrm{MP}}_1(\mathbf{X}, \mathbf{A}), \mathbf{A}), \mathbf{A})$$

(GraphSAGE-GCN(Hamilton et al.. 2017)
(Const-GAT(Veličković et al.. 2017)

$\widehat{D}^{-1}$: $performs\ normalized\ sum$

- row-wise shuffle the feature matrices within a subsampled patch

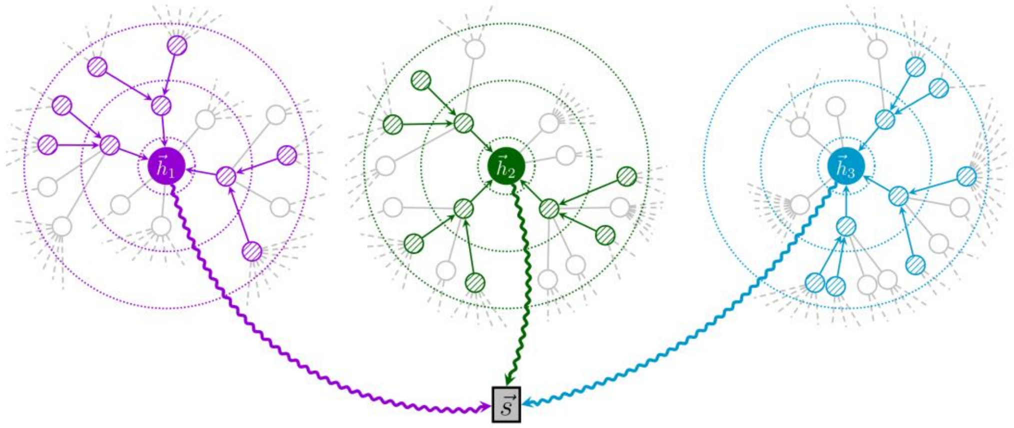


Figure 2: The DGI setup on large graphs (such as Reddit). Summary vectors, $\vec{s}$, are obtained by combining several subsampled patch representations, $\vec{h}_i$ (here obtained by sampling three and two neighbors in the first and second level, respectively).

03 CLASSIFICATION PERFORMANCE

3.2 Experimental Setup – encoder & corruption functions

(3) Inductive learning on multiple graphs(PPI)

three-layer mean-pooling model with dense skip connections

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

$W_{skip}: learnable\ projection\ matrix$

- randomly sampled training graphs

03 CLASSIFICATION PERFORMANCE

3.2 Experimental Setup – Readout, discriminator, and additional training details

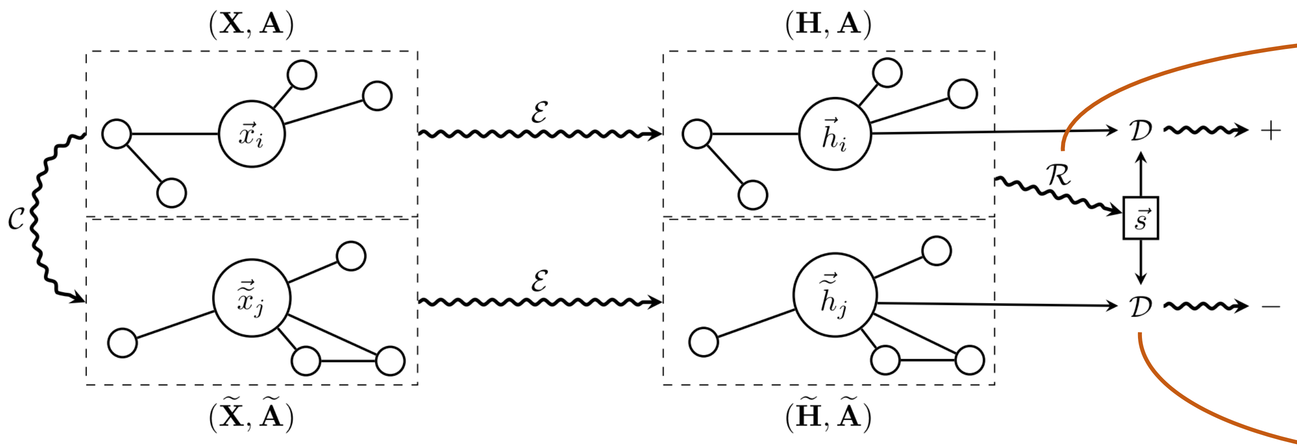


Figure 1: A high-level overview of Deep Graph Infomax. Refer to Section 3.4 for more details.

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

$\boldsymbol{W}$: *learnablie scoring matrix*
σ : *logistic sigmoid nonlinearity*

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

- Glorot initialization
- Adam SGD (learning rate = 0.001)
- Transductive: early stopping with a patience of 20 epochs
- 150 on Reddit, 20 on PPI

03 CLASSIFICATION PERFORMANCE

3.3 Results

Table 2: Summary of results in terms of classification accuracies (on transductive tasks) or micro-averaged F_1 scores (on inductive tasks). In the first column, we highlight the kind of data available to each method during training ($\mathbf{X}$: features, $\mathbf{A}$: adjacency matrix, $\mathbf{Y}$: labels). "GCN" corresponds to a two-layer DGI encoder trained in a supervised manner.

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

04 CONCLUSION

4.1 Qualitative analysis

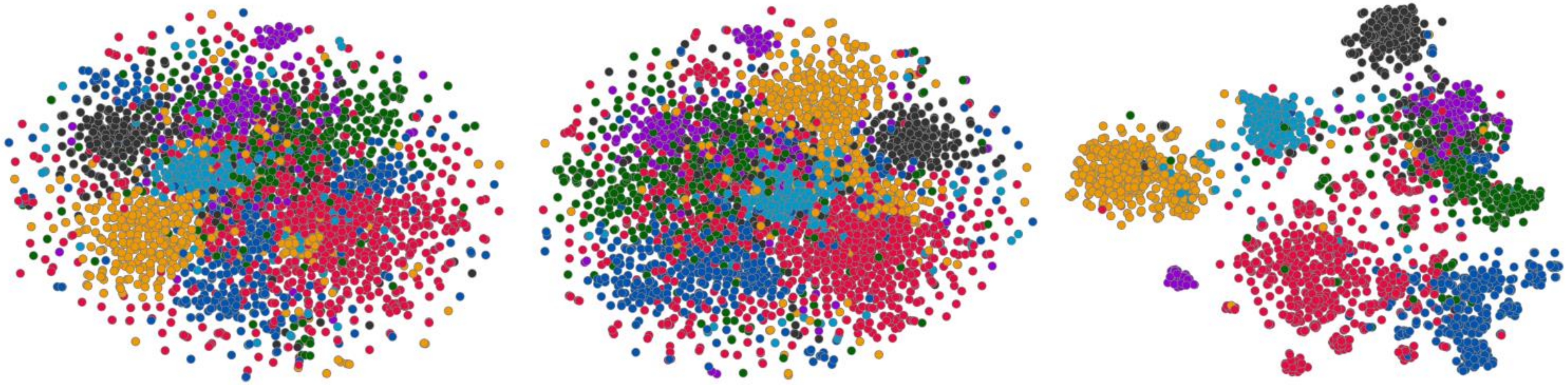


Figure 3: t-SNE embeddings of the nodes in the Cora dataset from the raw features (**left**), features from a randomly initialized DGI model (**middle**), and a learned DGI model (**right**). The clusters of the learned DGI model's embeddings are clearly defined, with a Silhouette score of 0.234.

04 CONCLUSION

Deep Graph Infomax

- New approach for learning unsupervised representations on graph-structured data
- Leveraging local mutual information maximization across the graph’s patch representations
- Obtain node embeddings that are mindful of the global structural properties of the graph
- Competitive performance across a variety of tasks

05 ADDITIONAL INFORMATION

· **Robustness to choice of corruption function**

$$\boldsymbol{\Sigma}_{ij} \sim \text{Bernoulli}(\rho)$$
$$\widetilde{\mathbf{A}} = \mathbf{A} \oplus \boldsymbol{\Sigma}$$

ρ : *corruption rate*

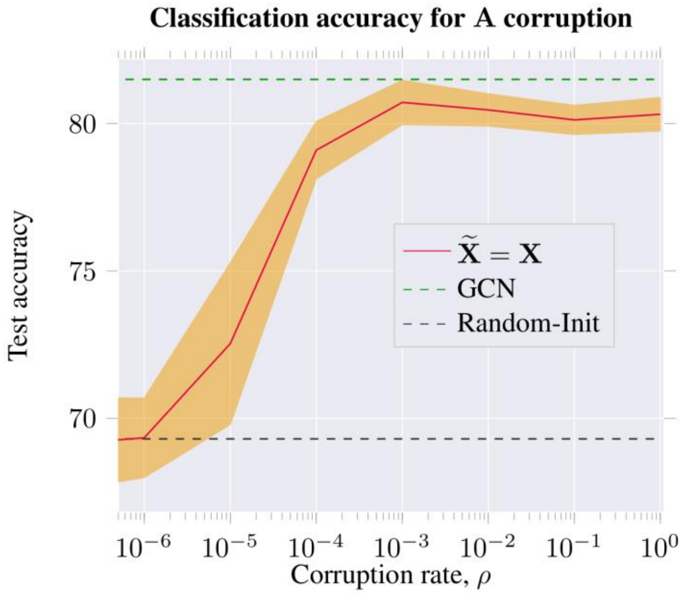


Figure 7: DGI also works under a corruption function that modifies only the adjacency matrix ($\widetilde{\mathbf{A}} \neq \mathbf{A}$) on the Cora dataset. The left range ($\rho \to 0$) corresponds to no modifications of the adjacency matrix—therein, performance approaches that of the randomly initialized DGI model. As ρ increases, DGI produces more useful features, but ultimately fails to outperform the feature-shuffling corruption function. **N.B.** log scale used for ρ.

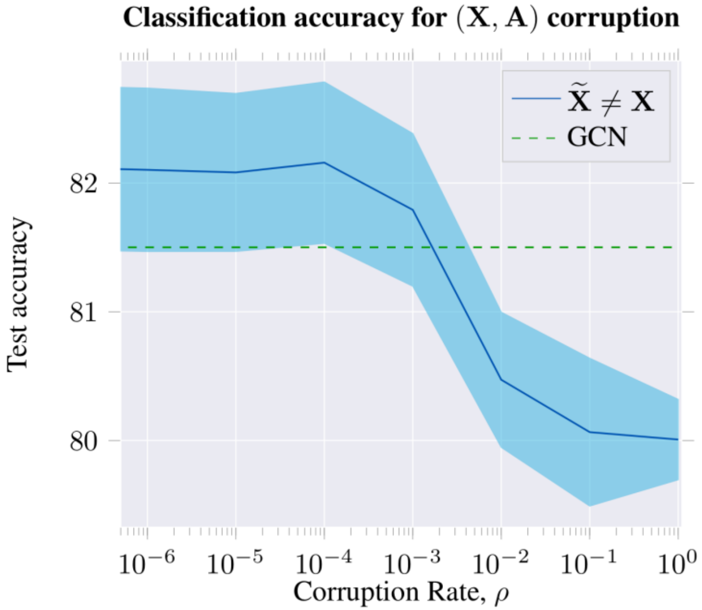


Figure 8: DGI is stable and robust under a corruption function that modifies *both* the feature matrix ($\mathbf{X} \neq \widetilde{\mathbf{X}}$) and the adjacency matrix ($\widetilde{\mathbf{A}} \neq \mathbf{A}$) on the Cora dataset. Corruption functions that preserve sparsity ($\rho \approx \frac{1}{N}$) perform the best. However, DGI still performs well even with large disruptions (where edges are added or removed with probabilities approaching 1). **N.B.** log scale used for ρ.

06 IMPLEMENTATION

- PyTorch, PyTorch geometric

```python
In [6]:
class GCN(nn.Module):
    def __init__(self, ft_in, n_fts):
        super(GCN, self).__init__()
        self.conv = GCNConv(ft_in, n_fts)
        self.act = nn.PReLU(n_fts)

    def forward(self, x, edge_index):
        x = self.conv(x, edge_index)
        x = self.act(x)
        return x
```

```python
In [7]:
def corruption(x, edge_index):
    return x[torch.randperm(x.size(0))], edge_index
```

```python
In [8]:
def summary(h, *args, **kwargs):
    return torch.mean(h, dim=0)
```

```python
In [9]:
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = DeepGraphInfomax(hidden_channels=512,
                         encoder=GCN(data.num_features, 512),
                         summary=summary,
                         corruption=corruption).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)
```

```python
In [10]:
def train():
    model.train()
    optimizer.zero_grad()
    pos_z, neg_z, summary = model(data.x, data.edge_index)
    loss = model.loss(pos_z, neg_z, summary)
    loss.backward()
    optimizer.step()
    return loss.item()
```

06 IMPLEMENTATION

· **Results**

Name	#nodes	#edges	#features	#classes	#train/val/test
Cora	2,708	10,556	1,433	7	140/500/1000
CiteSeer	3,327	9,104	3,703	6	120/500/1000
PubMed	19,717	88,648	500	3	60/500/1000

Cora

```
Epoch: 10, Loss: 0.9537
Epoch: 20, Loss: 0.6639
Epoch: 30, Loss: 0.4968
Epoch: 40, Loss: 0.3886
Epoch: 50, Loss: 0.3447
Epoch: 60, Loss: 0.3199
Epoch: 70, Loss: 0.3020
Epoch: 80, Loss: 0.2888
Epoch: 90, Loss: 0.2401
Epoch: 100, Loss: 0.2267
Accuracy: 0.8290
```

CiteSeer

```
Epoch: 10, Loss: 1.2206
Epoch: 20, Loss: 0.8322
Epoch: 30, Loss: 0.4920
Epoch: 40, Loss: 0.3300
Epoch: 50, Loss: 0.2627
Epoch: 60, Loss: 0.2438
Epoch: 70, Loss: 0.2997
Epoch: 80, Loss: 0.2786
Epoch: 90, Loss: 0.3120
Epoch: 100, Loss: 0.2219
Accuracy: 0.7140
```

PubMed

```
Epoch: 10, Loss: 1.3811
Epoch: 20, Loss: 1.3460
Epoch: 30, Loss: 1.2444
Epoch: 40, Loss: 1.0822
Epoch: 50, Loss: 0.9635
Epoch: 60, Loss: 0.9102
Epoch: 70, Loss: 0.8739
Epoch: 80, Loss: 0.8437
Epoch: 90, Loss: 0.8260
Epoch: 100, Loss: 0.8061
Accuracy: 0.7160
```

06 IMPLEMENTATION

- PyTorch PyTorch geometric

```python
class GCNlayer(nn.Module):
    def __init__(self, in_ftr, out_ftr):
        super(GCNlayer, self).__init__()
        self.conv = GCNConv(in_ftr, out_ftr)
        self.activation = nn.PReLU(out_ftr)

    def forward(self, x, edge_index):
        x = self.conv(x, edge_index)
        x = self.activation(x)
        return x
```

```python
class DGI(nn.Module):
    def __init__(self, data, dim):
        super(DGI, self).__init__()
        self.dim = dim
        self.data = Data(data.x, data.edge_index)
        self.loss = nn.BCEWithLogitsLoss()
        self.weight = nn.Parameter(torch.Tensor(self.dim, self.dim))
        nn.init.xavier_uniform_(self.weight)

    def discriminator(self, h, summary):
        value = torch.matmul(h, torch.matmul(self.weight, summary))
        return torch.sigmoid(value)

    def corruption(self, data):
        return Data(self.data.x[torch.randperm(self.data.x.size(0))], self.data.edge_index)
```

```python
def forward(self):
    pos_x = self.data
    neg_x = self.corruption(pos_x)
    encoder = GCNlayer(self.data.num_features, self.dim)
    pos_h = encoder(pos_x.x, pos_x.edge_index)
    neg_h = encoder(neg_x.x, neg_x.edge_index)
    summary = torch.sigmoid(torch.mean(pos_h, dim = 0))

    pos_h = self.discriminator(pos_h,summary)
    neg_h = self.discriminator(neg_h,summary)
    loss_pos = self.loss(pos_h, torch.ones_like(pos_h))
    loss_neg = self.loss(neg_h, torch.zeros_like(neg_h))
    return loss_pos + loss_neg

def predict(self, data):
    pos_x = data
    neg_x = self.corruption(pos_x)
    encoder = GCNlayer(self.data.num_features, self.dim)
    pos_h = encoder(pos_x.x, pos_x.edge_index)
    neg_h = encoder(neg_x.x, neg_x.edge_index)
    summary = torch.sigmoid(torch.mean(pos_h, dim = 0))
    return pos_h, neg_h, summary
```

06 IMPLEMENTATION

· **Discussion**

```python
def loss(self, pos_h, neg_h, summary):
    pos_loss = -torch.log(self.discriminator(pos_h, summary)).mean()
    neg_loss = -torch.log(1-self.discriminator(neg_h,summary)).mean()
    return pos_loss + neg_loss
```

Cora

```
Epoch: 1, Loss: 1.4378
Epoch: 2, Loss: 1.4025
Epoch: 3, Loss: 1.3923
Epoch: 4, Loss: 1.3890
Epoch: 5, Loss: 1.3885
Epoch: 6, Loss: 1.3872
Epoch: 7, Loss: 1.3864
Epoch: 8, Loss: 1.3865
Epoch: 9, Loss: 1.3864
Epoch: 10, Loss: 1.3864
Accuracy: 0.7710
```

CiteSeer

```
Epoch: 1, Loss: 1.4346
Epoch: 2, Loss: 1.4010
Epoch: 3, Loss: 1.3916
Epoch: 4, Loss: 1.3889
Epoch: 5, Loss: 1.3874
Epoch: 6, Loss: 1.3865
Epoch: 7, Loss: 1.3866
Epoch: 8, Loss: 1.3864
Epoch: 9, Loss: 1.3864
Epoch: 10, Loss: 1.3863
Accuracy: 0.6230
```

PubMed

```
Epoch: 1, Loss: 1.4483
Epoch: 2, Loss: 1.4351
Epoch: 3, Loss: 1.4335
Epoch: 4, Loss: 1.4239
Epoch: 5, Loss: 1.4214
Epoch: 6, Loss: 1.4175
Epoch: 7, Loss: 1.4145
Epoch: 8, Loss: 1.4116
Epoch: 9, Loss: 1.4085
Epoch: 10, Loss: 1.4059
Accuracy: 0.7320
```

THANK YOU –