InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization

F. Sun, J. Hoffmann, V. Verma, J. Tang
ICLR 2020

Minwoo Lee

Seoul National University

Motivation

- **Goal**: Unsupervised/Semi-supervised learning of **graph representation**
  - Variety of applications such as molecular properties and material science.
  - Usually scarce or no annotated labels

- Existing works on unsupervised graph representation learning
  - **Graph Kernels**
    - relies on **handcrafted features** -> **bad generalization** performance
  - Unsupervised node representation learning + aggregation
    - not designed for learning good graph representation

Methodology (Unsupervised) – InfoGraph

- **Idea**: *learn graph representation so that it encodes aspects of the data that are shared across all substructures (patches)*

- Patch representation: learned through GNN (*encoder*)

$$h_\phi^i = \mathrm{CONCAT}(\{h_i^{(k)}\}_{k=1}^K)$$

- Graph representation: Aggregation of patch representations

$$H_\phi(G) = \mathrm{READOUT}(\{h_\phi^i\}_{i=1}^N)$$

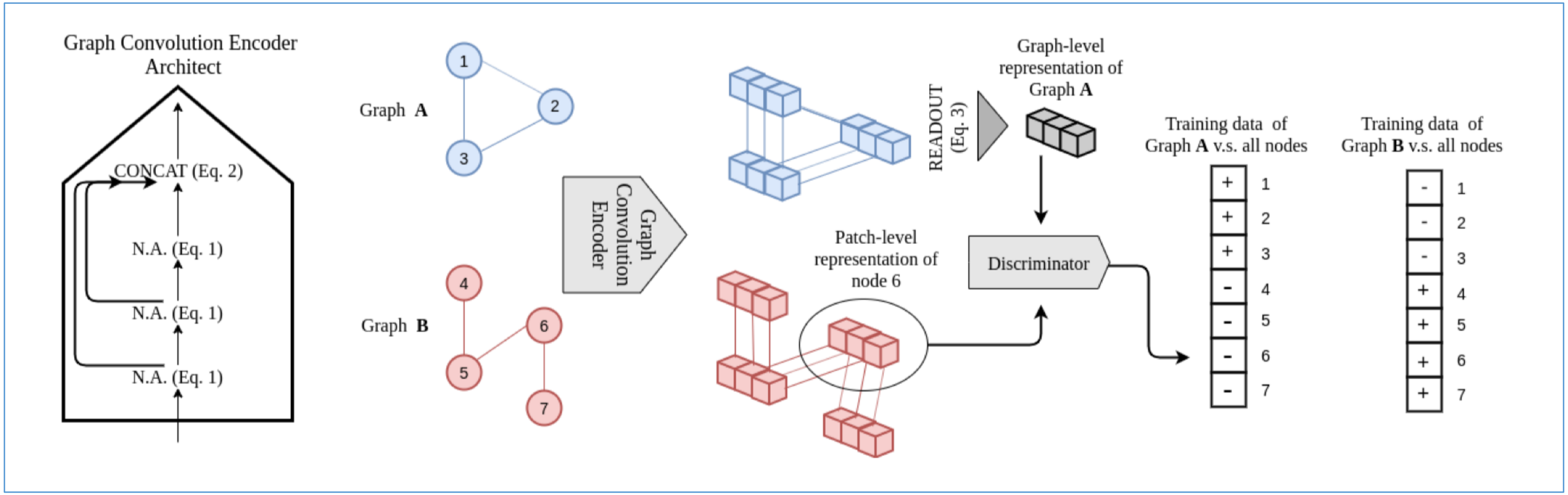
- Objective: Maximize **Mutual information** between patch representation and graph representation

$$\hat{\phi}, \hat{\psi} = \arg\max_{\phi,\psi} \sum_{G \in \mathbf{G}} \frac{1}{|G|} \sum_{u \in G} I_{\phi,\psi}(\vec{h}_\phi^u; H_\phi(G)).$$

Methodology (Unsupervised) – InfoGraph

- How to maximize mutual information?
  - Train **discriminator** to discriminate correct (true) graph-patch representation pairs with random (false) graph-patch combination

$$I_{\phi,\psi}(h_\phi^i(G); H_\phi(G)) :=$$
$$\mathbb{E}_{\mathbb{P}}[-\text{sp}(-T_{\phi,\psi}(\vec{h}_\phi^i(x), H_\phi(x)))] - \mathbb{E}_{\mathbb{P}\times\tilde{\mathbb{P}}}[\text{sp}(T_{\phi,\psi}(\vec{h}_\phi^i(x'), H_\phi(x)))]$$



Methodology (Semi-supervised) – InfoGraph*

- When labels partially exist:
  - Combine unsupervised objective with supervised objective (naive method)

$$L_{\text{total}} = \sum_{i=1}^{|\mathbb{G}^L|} L_{\text{supervised}}(y_\phi(G_i), o_i) + \lambda \sum_{j=1}^{|\mathbb{G}^L|+|\mathbb{G}^U|} L_{\text{unsupervised}}(h_\phi(G_j); H_\phi(G_j))$$

- Naive method could lead to **negative transfer**

Methodology (Semi-supervised) – InfoGraph*

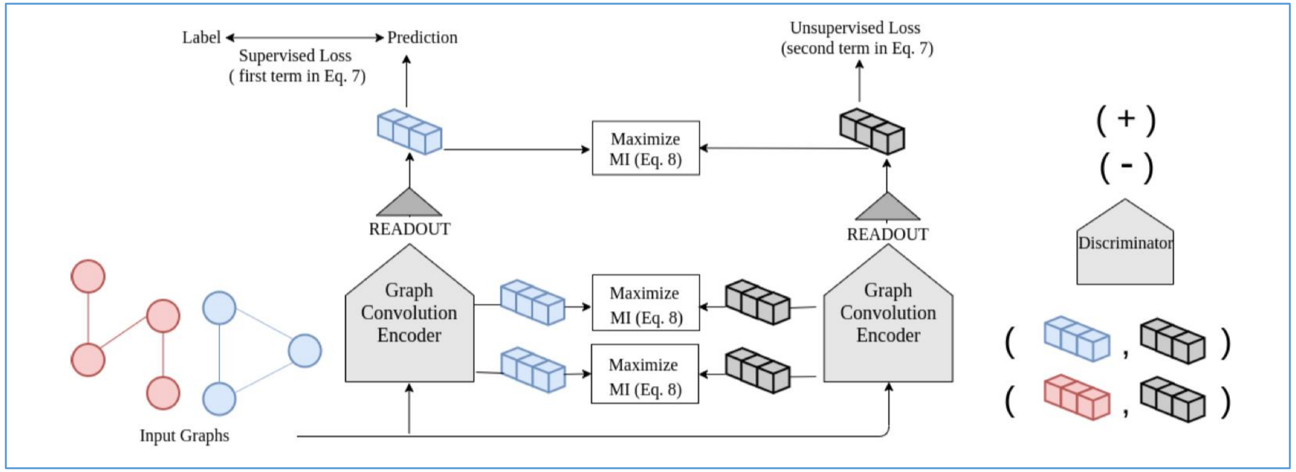
- To alleviate this problem: use **two** encoder models + **transfer** information between the models
  - Transfer method: Maximize Mutual Information between intermediate representations of encoder (GNN)

Model 1 (ϕ) **Model 2 (φ)**

$$L_{\text{total}} = \sum_{i=1}^{|\mathbb{G}^L|} L_{\text{supervised}}(y_\phi(G_i), o_i) + \sum_{j=1}^{|\mathbb{G}^L|+|\mathbb{G}^U|} L_{\text{unsupervised}}(h_\varphi(G_j); H_\varphi(G_j))$$

$$- \lambda \sum_{j=1}^{|\mathbb{G}^L|+|\mathbb{G}^U|} \frac{1}{|G_j|} \sum_{k=1}^{K} I(H_\phi^k(G_j); H_\varphi^k(G_j)).$$

Transfer (ϕ, φ)



Experimental Results – Unsupervised

- **Dataset**: 6 Benchmark graph classification task
- **Configuration**: LIBSVM downstream classifier on 512-dim (unsupervised-learned) feature
- **Result**: Best performance for 4 out of 6 tasks

Dataset	MUTAG	PTC-MR	RDT-B	RDT-M5K	IMDB-B	IMDB-M
(No. Graphs)	188	344	2000	4999	1000	1500
(No. classes)	2	2	2	5	2	3
(Avg. Graph Size)	17.93	14.29	429.63	508.52	19.77	13.00
Graph Kernels						
RW	83.72 ± 1.50	57.85 ± 1.30	OMR	OMR	50.68 ± 0.26	34.65 ± 0.19
SP	85.22 ± 2.43	58.24 ± 2.44	64.11 ± 0.14	39.55 ± 0.22	55.60 ± 0.22	37.99 ± 0.30
GK	81.66 ± 2.11	57.26 ± 1.41	77.34 ± 0.18	41.01 ± 0.17	65.87 ± 0.98	43.89 ± 0.38
WL	80.72 ± 3.00	57.97 ± 0.49	68.82 ± 0.41	46.06 ± 0.21	72.30 ± 3.44	46.95 ± 0.46
DGK	87.44 ± 2.72	60.08 ± 2.55	78.04 ± 0.39	41.27 ± 0.18	66.96 ± 0.56	44.55 ± 0.52
MLG	87.94 ± 1.61	$\mathbf{63.26 \pm 1.48}$	> 1 Day	> 1 Day	66.55 ± 0.25	41.17 ± 0.03
Other Unsupervised Methods						
node2vec	72.63 ± 10.20	58.58 ± 8.00	-	-	-	-
sub2vec	61.05 ± 15.80	59.99 ± 6.38	71.48 ± 0.41	36.68 ± 0.42	55.26 ± 1.54	36.67 ± 0.83
graph2vec	83.15 ± 9.25	60.17 ± 6.86	75.78 ± 1.03	47.86 ± 0.26	71.1 ± 0.54	$\mathbf{50.44 \pm 0.87}$
InfoGraph	$\mathbf{89.01 \pm 1.13}$	61.65 ± 1.43	$\mathbf{82.50 \pm 1.42}$	$\mathbf{53.46 \pm 1.03}$	$\mathbf{73.03 \pm 0.87}$	49.69 ± 0.53

Experimental Results – Semi-supervised

- **Dataset**: QM9 (molecular property prediction)
  - 130,462 molecules
- **Model**: enn-s2s model from (Gilmer et al. 2017)
- **Result**: Best performance for 11 out of 12 targets

Target	Mu (0)	Alpha (1)	HOMO (2)	LUMO (3)	Gap (4)	R2 (5)	ZPVE(6)	U0 (7)	U (8)	H (9)	G(10)	Cv (11)
MAE	0.3201	0.5792	0.0060	0.0062	0.0091	10.0469	0.0007	0.3204	0.2934	0.2722	0.2948	0.2368

Naive method

Semi-Supervised	Error Ratio											
Mean-Teachers	1.09	1.00	**0.99**	1.00	**0.97**	0.52	0.77	1.16	0.93	0.79	0.86	0.86
InfoGraph	1.02	0.97	1.02	**0.99**	1.01	0.71	0.96	0.85	0.93	0.93	0.99	1.00
InfoGraph*	**0.99**	**0.94**	**0.99**	**0.99**	0.98	**0.49**	**0.52**	**0.44**	**0.58**	**0.57**	**0.54**	**0.83**

* Error ratio lower than 1 means better performance than supervised model

Reproducing Results

Reproducing Results

- **Re-implemented Code**: https://github.com/minwhoo/InfoGraph
- **Original Implementation by author**: https://github.com/fanyun-sun/InfoGraph

- **Code modifications**
  - **Remove** unused/baseline code
  - **Clean up and comment** on train/eval code and models
  - **Copy** dataset preprocessing / downstream evaluation metric codes

Reproducing Results

Code summary

- **unsupervised.py**
  - train/eval code for unsupervised setting
  - **InfoGraph, GINEncoder** pytorch modules
- **semisupervised.py**
  - train/eval code for semi-supervised setting
  - **InfoGraphSemi, ENNSet2SetEncoder** pytorch modules
  - QM9 Dataset processing code
- **model.py**
  - **Discriminator** modules shared between unsupervised and semisupervised settings
- **evaluate_embedding.py**
  - downstream Logistic Regression/SVC evaluation code copied from original repo

Code modification structure

Reproducing Results

PyTorch 1.5.0

To install the binaries for PyTorch 1.5.0, simply run

```
$ pip install torch-scatter==latest+${CUDA} -f https://pytorch-geometric.com/whl/torch-1.5.0.html
$ pip install torch-sparse==latest+${CUDA} -f https://pytorch-geometric.com/whl/torch-1.5.0.html
$ pip install torch-cluster==latest+${CUDA} -f https://pytorch-geometric.com/whl/torch-1.5.0.html
$ pip install torch-spline-conv==latest+${CUDA} -f https://pytorch-geometric.com/whl/torch-1.5.0.html
$ pip install torch-geometric
```

where `${CUDA}` should be replaced by either `cpu`, `cu92`, `cu101` or `cu102` depending on your PyTorch installation.

Steps for running the code

1. **Manually install python packages below**
   - pytorch==1.5.0
     - `pip install pytorch==1.5.0`
   - pytorch-geometric==1.5.0
     - Follow instructions [here]
2. **Install remaining required python packages by running:**
   - `pip install -r requirements.txt`
3. **Run training code** (Datasets automatically downloaded when running the code)
   - For unsupervised learning:
     - `python unsupervised.py --dataset=${DATASET}`
       - where `${DATASET}` should be replaced by one of `MUTAG`, `PTC_MR`, `REDDIT-BINARY`, `REDDIT-MULTI-5K`, `IMDB-BINARY`, `IMDB-MULTI`
   - For semi-supervised learning:
     - `python semisupervised.py --target=${TARGET}`
       - where `${TARGET}` denotes the target predicting property and should be one of 0,1,…,11

Reproducing Results – Unsupervised

- Results show slightly worse average performance
  - Possibly due to fixed hyperparameter setting and high variance of the task

Dataset (No. Graphs) (No. classes) (Avg. Graph Size)	MUTAG 188 2 17.93	PTC-MR 344 2 14.29	RDT-B 2000 2 429.63	RDT-M5K 4999 5 508.52	IMDB-B 1000 2 19.77	IMDB-M 1500 3 13.00
Graph Kernels						
RW	83.72 ± 1.50	57.85 ± 1.30	OMR	OMR	50.68 ± 0.26	34.65 ± 0.19
SP	85.22 ± 2.43	58.24 ± 2.44	64.11 ± 0.14	39.55 ± 0.22	55.60 ± 0.22	37.99 ± 0.30
GK	81.66 ± 2.11	57.26 ± 1.41	77.34 ± 0.18	41.01 ± 0.17	65.87 ± 0.98	43.89 ± 0.38
WL	80.72 ± 3.00	57.97 ± 0.49	68.82 ± 0.41	46.06 ± 0.21	72.30 ± 3.44	46.95 ± 0.46
DGK	87.44 ± 2.72	60.08 ± 2.55	78.04 ± 0.39	41.27 ± 0.18	66.96 ± 0.56	44.55 ± 0.52
MLG	87.94 ± 1.61	$\mathbf{63.26 \pm 1.48}$	> 1 Day	> 1 Day	66.55 ± 0.25	41.17 ± 0.03
Other Unsupervised Methods						
node2vec	72.63 ± 10.20	58.58 ± 8.00	-	-	-	-
sub2vec	61.05 ± 15.80	59.99 ± 6.38	71.48 ± 0.41	36.68 ± 0.42	55.26 ± 1.54	36.67 ± 0.83
graph2vec	83.15 ± 9.25	60.17 ± 6.86	75.78 ± 1.03	47.86 ± 0.26	71.1 ± 0.54	$\mathbf{50.44 \pm 0.87}$
InfoGraph	$\mathbf{89.01 \pm 1.13}$	61.65 ± 1.43	$\mathbf{82.50 \pm 1.42}$	$\mathbf{53.46 \pm 1.03}$	$\mathbf{73.03 \pm 0.87}$	49.69 ± 0.53
Re-implementation Results	88.33 ↓	60.17 ↓	82.2 ↓	54.19 ↑	71.40 ↓	48.00 ↓

* higher is better
* Results are from single-run

Reproducing Results – Semi-supervised

- Results show there is some discrepancy between results in paper and semi-supervised training code.
- The cause of error wasn't found in time of submission of this project.
- Only 3 out of 12 target properties were trained and evaluated due to time constraints.

Target	Mu (0)	Alpha (1)	HOMO (2)	LUMO (3)	Gap (4)	R2 (5)	ZPVE(6)	U0 (7)	U (8)	H (9)	G(10)	Cv (11)
MAE	0.3201	0.5792	0.0060	0.0062	0.0091	10.0469	0.0007	0.3204	0.2934	0.2722	0.2948	0.2368

Semi-Supervised	Error Ratio											
Mean-Teachers	1.09	1.00	**0.99**	1.00	**0.97**	0.52	0.77	1.16	0.93	0.79	0.86	0.86
InfoGraph	1.02	0.97	1.02	**0.99**	1.01	0.71	0.96	0.85	0.93	0.93	0.99	1.00
InfoGraph*	**0.99**	**0.94**	**0.99**	**0.99**	0.98	**0.49**	**0.52**	**0.44**	**0.58**	**0.57**	**0.54**	**0.83**

Test error ratio	0.66	-	26.55	26.20	-	-	-	-	-	-	-	-
Test MAE	**0.2126**	-	**0.1593**	**0.1572**	-	-	-	-	-	-	-	-

* lower is better
* Results are from single-run