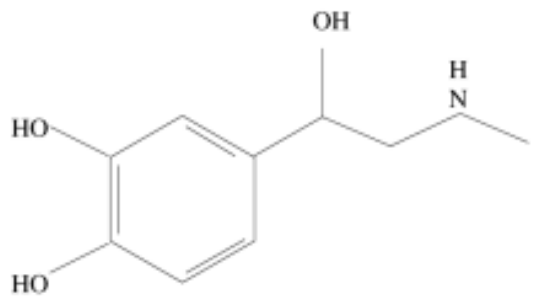
5. Graph Convolutional Neural Network

What is a graph convolutional neural network(GCNN)?

- Consider a machine learning problem with graph data
- Type 1. Graph classification
  - input: $\{(G_1(V_1, E_1, X_1, Z_1), Y_1), \ldots, (G_1(V_n, E_n, Z_n, X_n), Y_n)\}$
    n samples of graphs with nodes V_i, edges E_i, node features X_1, edge weights Z_1, graph label Y_1
  - goal: find $f_W(G_i(V_i, E_i, X_i, Z_i)) = \hat{Y}$ to predict the graph label
- Type 2. Semi-supervised node classification
  - input: Single graph $G(V, E)$, labeled nodes $\{(X_1, Y_1), \ldots, (X_L, Y_L)\}$, unlabeled nodes $\{X_{L+1}, \ldots, X_n\}$
  - goal: find a function $f_W(G, X_{1:n}) = \hat{Y}_{1:n}$ to predict the node label
- Type 3. Unsupervised node embedding (graph auto-encoder)
  - input: Single graph $G(V, E)$
  - goal: find an encoder $f_W(G) = \hat{X}_{1:n}$ and a decoder $g_W(\hat{X}_{1:n}) = \hat{G}$ to predict the edges
- Main challenge: graphs change in size and connections, and it is not clear how to input it to a neural network, as opposed to typical datasets that are set of fixed size real-valued vectors.

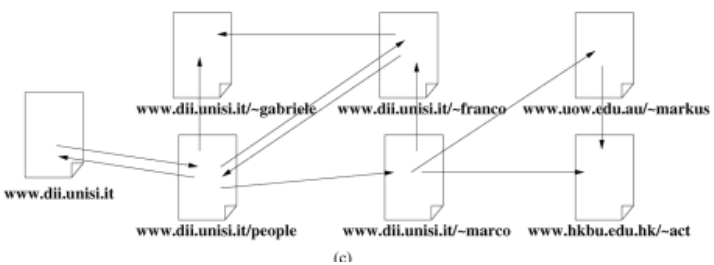
Examples of practical problems tackled with GCNN

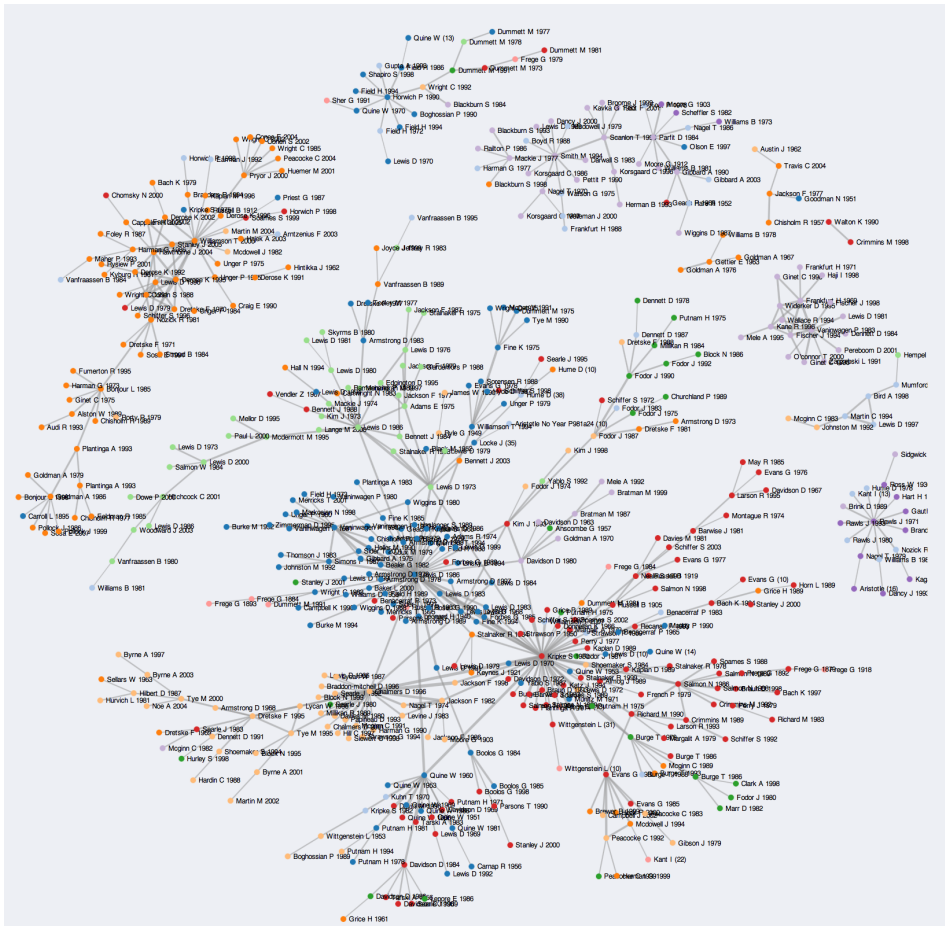
- Type 1. supervised classification of molecular network for drug discovery



Examples of practical problems tackled with GCNN

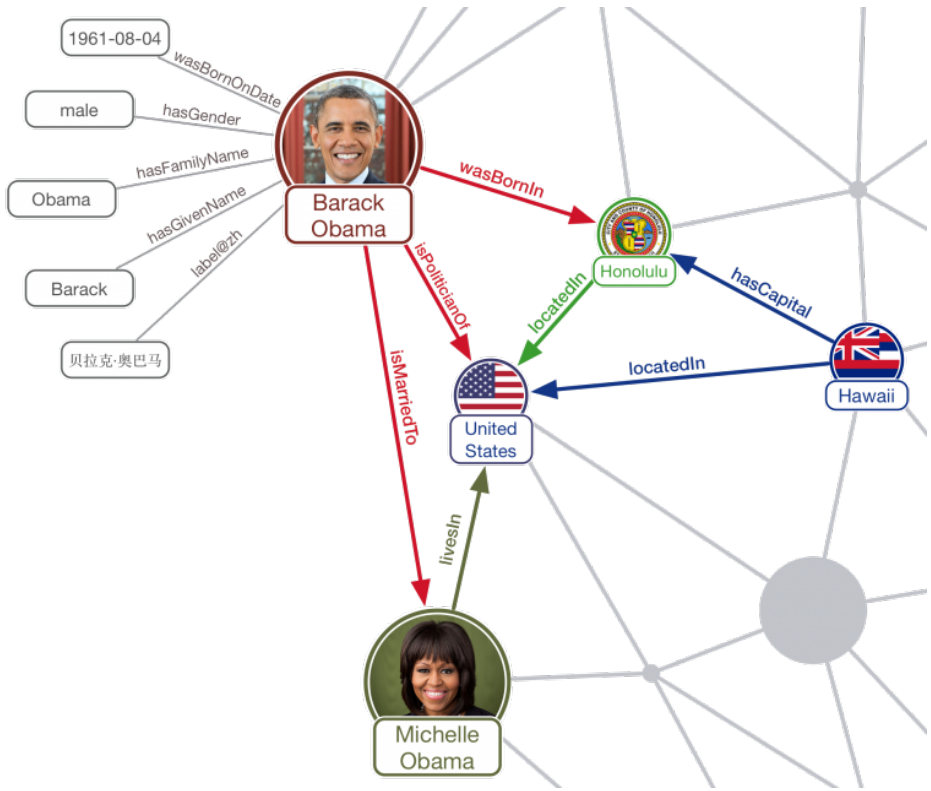
- Type 2. semisupervised classification of documents in citation network





Examples of practical problems tackled with GCNN

- Type 3. unsupervised link prediction on knowledge graph



Examples of (practical?) problems we can tackle

- Example 1. Detecting clusters (exponential but groundtruth available)
  - Input: $\{G_i(V_i, E_i)\}_{i=1}^{n}$, $\{Y_i = 0\}_{i=1}^{n}$, $\{G_i(V_i, E_i)\}_{i=n+1}^{2n}$, $\{Y_i = 1\}_{i=n+1}^{2n}$ one set of samples generated from Erdos-Renyi graph, and another set from Stochastic Block Model
  - Goal: classify a graph whether it is from ER or SBM
  - Research question
    - ★ which graph neural network architecture/loss should we use?
    - ★ which parameters for ER and SBM should we use?
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one parameters and test it on another?
    - ★ how does it compare against other (non-neural network) methods that use the knowledge of SBM explicitly?

Examples of (practical?) problems we can tackle

- Example 2. estimating minimum spanning tree (polynomial)
  - Input: $\{G_i(V_i, E_i, Z_i)\}_{i=1}^n$, $\{Y_i\}_{i=1}^n$
    generate weighted graphs, and corresponding value of minimum spanning trees
  - Goal: estimate the value of the minimum spanning tree
  - Research question
    - ★ which graph neural network architecture/loss should we use?
    - ★ which input graphs should we use? (random graph with random weights is not good)
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ how does it compare against the exact algorithm?

Examples of (practical?) problems we can tackle

- Example 3. estimating PageRank scores (polynomial)
  - Input: $\{G_i(V_i, E_i)\}_{i=1}^n$, $\{Y_i\}_{i=1}^n$
    generate directed graphs, and corresponding pagerank scores for all nodes
  - Goal: estimate the pagerank score
  - Research question
    - ★ which graph neural network architecture/loss should we use?
    - ★ which input graphs should we use? (random graph is not good, perhaps preferential attachment graph is better)
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ how does it compare against the exact algorithm?

Examples of (practical?) problems we can tackle

- Example 4. detecting Eulerian cycle (polynomial)
  - Input: $\{G_i(V_i, E_i)\}_{i=1}^n$, $\{Y_i\}_{i=1}^n$
    generate directed graphs, and label it as Eulerian or not
  - Goal: detect Eulerian graphs
  - Research question
    - ★ which graph neural network architecture/loss should we use?
    - ★ which input graphs should we use?
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ how does it compare against the exact algorithm?

Examples of (practical?) problems we can tackle

- Example 5. detecting Hamiltonian cycle (exponential)
  - Input: $\{G_i(V_i, E_i)\}_{i=1}^n$, $\{Y_i\}_{i=1}^n$
    generate directed graphs, and label it as Hamiltonian or not
  - Goal: detect Hamiltonian graphs
  - Research question
    - ★ How do we find the labels of the training examples??
    - ★ which graph neural network architecture/loss should we use?
    - ★ which input graphs should we use?
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ how does it compare against other heuristics?

Examples of (practical?) problems we can tackle

- Example 6. finding maximum cut (exponential)
  - Input: $\{G_i(V_i, E_i, Z_i)\}_{i=1}^n$, $\{Y_i\}_{i=1}^n$
    generate weighted undirected graphs, and label it with its maximum cut
  - Goal: estimate max cut
  - Research question
    - ★ How do we find the labels of the training examples??
    - ★ which graph neural network architecture/loss should we use?
    - ★ which input graphs should we use?
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ how does it compare against other heuristics?

Examples of (practical?) problems we can tackle

- Example 7. finding minimum cut (polynomial)
  - Input: $\{G_i(V_i, E_i, Z_i)\}_{i=1}^n$, $\{Y_i\}_{i=1}^n$
    generate weighted undirected graphs, and label it with its minimum cut
  - Goal: estimate min cut
  - Research question
    - ★ which graph neural network architecture/loss should we use?
    - ★ which input graphs should we use?
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ how does it compare against exact algorithm?

Examples of (practical?) problems we can tackle

- Example 8. finding max-weight matching (polynomial)
  - Input: $\{G_i(V_i, E_i, Z_i)\}_{i=1}^n$, $\{Y_i\}_{i=1}^n$
    generate weighted undirected graphs, and label it with its edges that are matching
  - Goal: estimate the set of edges in the matching
  - Research question
    - ★ which graph neural network architecture should we use?
    - ★ which input graphs should we use?
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ how does it compare against exact algorithm?

Examples of (practical?) problems we can tackle

- Example 9. graph coloring (exponential)
  - Input: $G(V, E)$,
  - Goal: for one given graph, learn the coloring of nodes such that adjacent nodes have different colors.
  - Research question
    - ★ which graph neural network architecture should we use?
    - ★ which loss function should we use?

Examples of (practical?) problems we can tackle

- Example 10. estimate shortest paths (polynomial)
  - Input: $\{G_i(V_i, E_i, Z_i)\}, \{Y_i\}$,
    set of directed graphs with a single source and single target with distances on the edges, labeled by the length of the shortest path
  - Goal: estimate the shortest path length
  - Research question
    - ★ which graph neural network architecture should we use?
    - ★ which input graphs should we use?
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ how does it compare against exact algorithm?

Examples of (practical?) problems we can tackle

- Example 11. semi-supervised learning with stochastic block models (exponential)
  - Input: $G(V, E, X)$, Y_L
    generate one single graph from a stochastic block model, so that we know the labels of all the nodes
    generate multi-dimensional features X_i for each node i from some distribution conditioned on the true label, e.g. $X_i \sim N(\mu_{Y_i}, \Sigma_{Y_i})$
    reveal some of the labels of the nodes, perhaps 3% of the nodes
  - Goal: find the labels of all the nodes
  - Research question
    - which graph neural network architecture should we use?
    - does the architecture scale? can we train on small graphs and test on large graphs?
    - is it robust? can we train on one type of graphs/weights and test it on another?

Examples of (practical?) problems we can tackle

- Example 12. semi-supervised learning on citation networks (?)
  - Input: $G(V, E, X)$, Y_L
    use the benchmark citation network datasets from https://linqs.soe.ucsc.edu/node/236 called CiteSeer, CORA, and PubMed
  - Goal: find the labels of all the nodes
  - Research question
    - ★ which graph neural network architecture should we use?
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ Can we beat the state-of-the-art?

Examples of (practical?) problems we can tackle

- Example 13. supervised learning on molecular network (?)
  - Input: $\{G_i(V_i, E_i, X_i, Z_i)\}$, $\{Y_i\}$
    use the benchmark citation network datasets
  - Goal: classify the graph
  - Research question
    - ★ which graph neural network architecture should we use?
    - ★ does the architecture scale? can we train on small graphs and test on large graphs?
    - ★ is it robust? can we train on one type of graphs/weights and test it on another?
    - ★ Can we beat the state-of-the-art?

Concrete examples of GNN in action: citation network

- Citation Network Benchmark Dataset

Table: Citation Network Dataset

Dataset	Nodes	Edges	Classes	Features	Labeled nodes
CiteSeer	3,327	4,732	6	3,703	120
Cora	2,708	5,429	7	1,433	140
PubMed	19,717	44,328	3	500	60

Graph Convolutional Network (GCN) by Kipf and Welling [2017 ICLR]

- Input:
  - graph: $G(V, E)$ or equivalently $A \in \{0, 1\}^{n \times n}$
  - node features: $X \in \mathbb{R}^{n \times d_x}$
  - labeled nodes: $\{Y_i\}_{i \in L}$
- Output:
  - estimated classes: $Z = f_W(X, A) \in \mathbb{R}^{n \times d_y}$
- goal: graph-based semisupervised learning

- How would you attack this problem?

Leader board

Table 2: Classification accuracy in percent with a fixed split of data from (Yang et al., 2016).

Input	Method	Citeseer	Cora	PubMed
y_1^L, x_1^L	Singlelayer Perceptron	57.2	57.4	69.8
	Multilayer Perceptron	64.0	57.5	71.4
y_1^L, x_1^{L+U}	T-SVM (Joachims, 1999)	64.0	57.5	62.2
y_1^L, G	DeepWalk (Perozzi et al., 2014)	43.2	67.2	65.3
y_1^L, x_1^{L+U}, G	LP (Zhu et al., 2003)	45.3	68.0	63.0
	ICA (Lu & Getoor, 2003)	69.1	75.1	73.9
	ManiReg (Belkin et al., 2006)	60.1	59.5	70.7
	SemiEmb (Weston et al., 2012)	59.6	59.0	71.1
	DCNN (Atwood & Towsley, 2016)		76.8	73.0
	Planetoid (Yang et al., 2016)	64.7	75.7	77.2
	MoNet (Monti et al., 2016)		81.7	78.8
	Graph-CNN (Such et al., 2017)		76.3	
	DynamicFilter (Verma et al., 2017)		81.6	79.0
	Bootstrap (Buchnik & Cohen, 2017)	53.6	78.4	78.8
	GCN (Kipf & Welling, 2016)	70.3	81.5	79.0
	GLN	70.9±.05	81.2±.05	78.9±.05
	AGNN (this paper)	**71.7**±.08	**82.6**±.09	**79.9**±.07

- GCN:
  [Input]-[Propagation]-[Perceptron]-[Propagation]-[Perceptron]- $\cdots$ -[SoftMax]-[Output]
- Forward Pass:
  - $H^{(0)}) = X$
  - Repeat for $t = 1, 2, \ldots, T$
    - $\tilde{H}^{(t)} = PH^{(t-1)}$, with $P = D^{-1}A$
    - $H^{(t)} = \mathrm{ReLU}(\tilde{H}^{(t)} W^{(t)})$
  - $\mathrm{SoftMax}(H^{(T)} W^{(T+1)})$
- Training:
  - Let $Z = f_W(X, A)$
  - the weights $W^{(1)}$, $W^{(2)}$, ..., $W^{(T+1)}$ are trained on the cross entropy loss

$$\mathcal{L}_{X,A,Y_L}(W) = -\sum_{i \in [n]} \sum_{j=1}^{d_y} Y_{ij} \ln Z_{ij}$$

- Why cross entropy loss?
  it measures distance between Y_i and Z_i, e.g.
  $-\sum_j Y_{ij} \ln Z_{ij} = 0$ if $Y_i = [1, 0, 0]$ and $Z_i = [1, 0, 0]$, and
  $-\sum_j Y_{ij} \ln Z_{ij} = -\ln(1/3)$ if $Y_i = [1, 0, 0]$ and $Z_i = [1/3, 1/3, 1/3]$.
- What is GCN doing?
  summarizing the neighborhood and extracting sufficient statistics
- Naive approach: store all neighborhood information
  - computationally intractable
  - memory blows up
  - varying dimensions
- as a solution GCN summarizes the local neighborhood by *local averaging* in propagation layer, and attempts to find the sufficient statistics via *perceptron layer*, recursively.

- some interpretation of the learned embeddings:

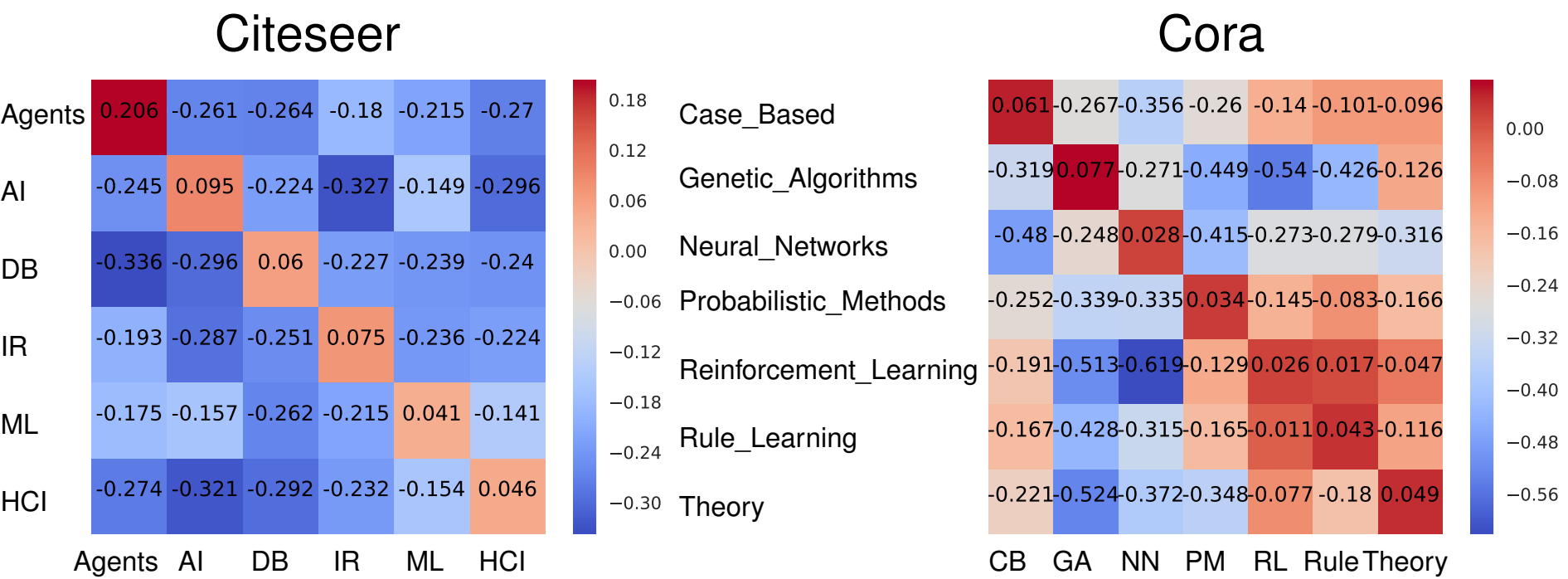


Figure: Average influence from a column class to a row class

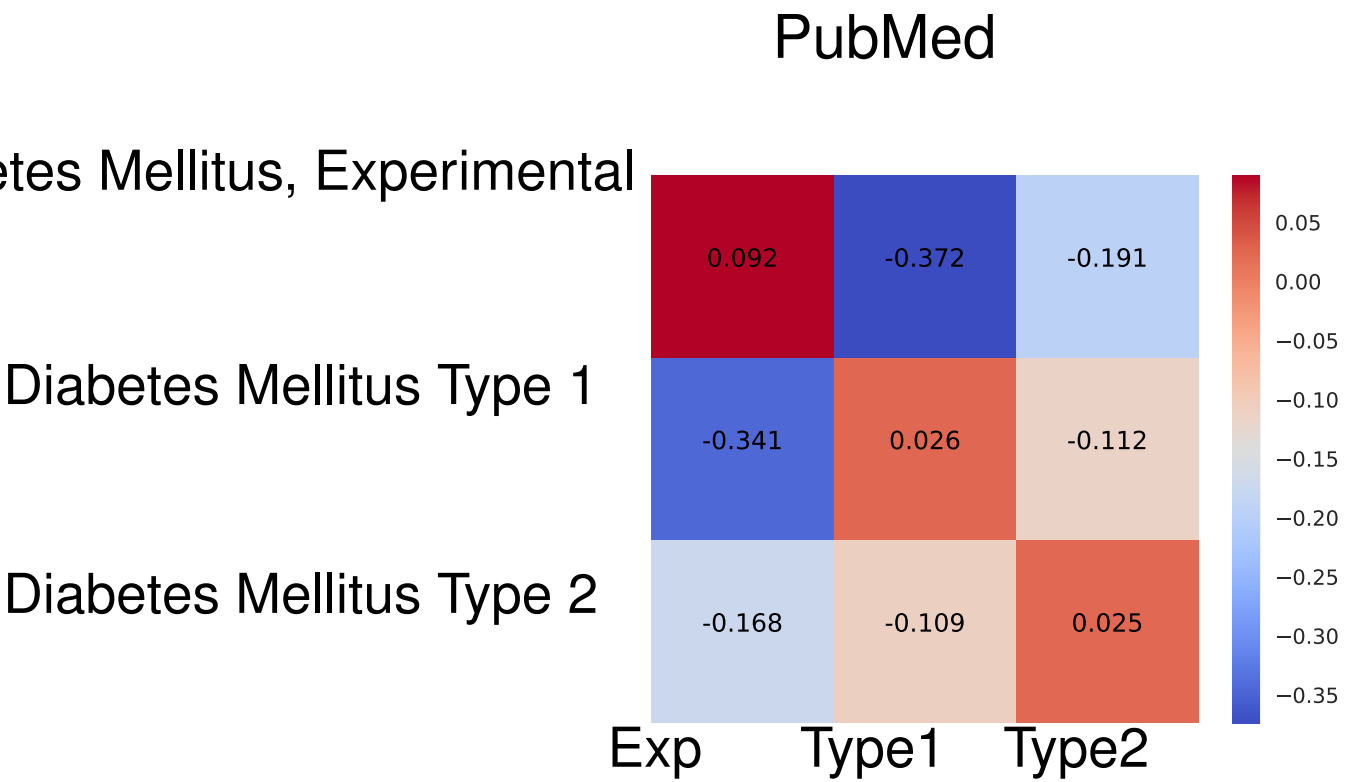


Figure: Average influence from a column class to a row class

Other approaches for graph-based semi-supervised learning

- main question is "how do you encode the graph information into a learning?"

- graph as a regularizer

$$\mathcal{L}_{X,A,Y_L}(W) = \underbrace{\sum_{i \in L} \ell(f_W(X_i), Y_i)}_{\mathcal{L}_{\text{supervised}}(X_L, Y_L)} + \lambda \underbrace{\sum_{(j,k) \in E} \|f_W(X_j) - f_W(X_k)\|^2}_{\mathcal{L}_{\text{regularizer}}(A,X)}$$

- this is a natural *parametric approach* that can be learned via back-propagation [“Deep Learning via Semi-supervised embedding " by Weston et al. 2012]
- two parts in the loss
- encode the graph as a part of the loss, forcing nearby nodes to have similar labels (or embeddings)

- There are earlier regularization approaches that are non-parametric

$$\mathcal{L}_{X,A,Y_L}(f) = \underbrace{\sum_{i \in L} \ell(f(i), Y_i)}_{\mathcal{L}_{\text{supervised}}(Y_L)} + \lambda \underbrace{\sum_{(j,k) \in E} B_{ij} \|f(j) - f(k)\|^2}_{\mathcal{L}_{\text{regularizer}}(A,X)}$$

  with $B_{ij} = e^{-(X_i - X_j)^2/\sigma^2}$.
- This is a very popular approach known as **Label Propagation** [“Learning from labeled and unlabeled data with label propagation”, X Zhu, Z Ghahramani, 2002]
- admits a closed form solution for $\lambda \to 0$

- we force true labels on the known nodes: $f(i) = Y_i$, for $i \in L$
- build a similarity matrix B with $B_{ij} = e^{-(X_i - X_j)^2/\sigma^2}$
- build a **graph Laplacian** $A = \mathrm{diag}(B\mathbb{1}) - B$
- and consider binary classification where $f = [f(1), \ldots, f(n)] \in \{0,1\}^n$
- then the loss from previous slide becomes

$$\text{minimize}_f \;\; f^T A f$$

subject to $f(L) = Y_L$.

- This gives

$$\mathcal{L} = \begin{bmatrix} f_L & f_U \end{bmatrix} \begin{bmatrix} A_{LL} & A_{LU} \\ A_{UL} & A_{UU} \end{bmatrix} \begin{bmatrix} f_L \\ f_U \end{bmatrix}$$

  and we can minimize $\mathcal{L}(f_U) = f_L^T A_{LL} f_L + 2 f_L^T A_{LU} f_U + f_U^U A_{UU} f_U$ with $f_U = A_{UU}^{-1} A_{UL} f_L$
- as it admits this closed-form solution, it is very popular, but ...

Concrete examples of GNN in action: community detection

- Input: graph $G(V, E)$ or A
- Output: $Z = f_W(A) \in \mathbb{R}^{n \times k}$ clustering of the node sinto k classes
-
- Spectral clustering:
  consider binary classification for now and let $Z_i \in \{-1, +1\}$

$$\text{minimize} \quad \sum_{i,j} A_{ij}(1 - Z_i Z_j)$$

  minimizes the **cut** between two classes
- and this is $\sum_j A_{ij}(1 - Z_i Z_j) = D_i - \sum_j A_{ij} Z_i Z_j$, and hence

$$\text{minimize} \quad \sum_{i,j} Z_i L_{ij} Z_j \quad = \quad Z^T L Z \ ,$$

  where $L = D - A$ is the graph Laplacian of G, D_i is the degree of node i, $D = \text{diag}([D_1, \ldots, D_n])$

- We want to solve

$$\text{minimize} \quad Z^T L Z$$

  subject to $Z_i \in \{+1, -1\}$
- which is hard as it is a combinatorial problem. A common heuristic is to relax the constraint and solve

$$\text{minimize} \quad Z^T L Z$$

  subject to $\|Z\|^2 = n$.
- however, this has a trivial solution, $Z_i = 1$ for all i, that achieves the minimum. Instead, we add a constraint that Z's have to be orthogonal to $\mathbb{1}$

$$\text{minimize} \quad Z^T L Z$$

  subject to $\sum_i Z_i = 0$
- This has a beautiful analytical solution now:

- GNN approach to community detection
  [Community Detection with Graph Neural Networks, Joan Bruna, Xiang Li, 2017]
- Architecture

$$H^{(0)} = \begin{bmatrix} degree_1 & \dots & degree_n \end{bmatrix}$$

$$H_1^{(t+1)} = \mathrm{ReLU}\left(H^{(t)} W^{(t,1)} + \mathrm{diag}(A\mathbb{1}) H^{(t)} W^{(t,2)} + \frac{1}{n} \mathbb{1}\mathbb{1}^T H^{(t)} W^{(t,3)} + H^{(t)} W^{(t,4+j)} \right)$$

$$H_2^{(t+1)} = H^{(t)} W^{(t,1)} + \mathrm{diag}(A\mathbb{1}) H^{(t)} W^{(t,2)} + \frac{1}{n} \mathbb{1}\mathbb{1}^T H^{(t)} W^{(t,3)} + H^{(t)} W^{(t,4+j)}$$

$$H^{(t+1)} = \begin{bmatrix} H_1^{(t+1)} & H_2^{(t+1)} \end{bmatrix}$$

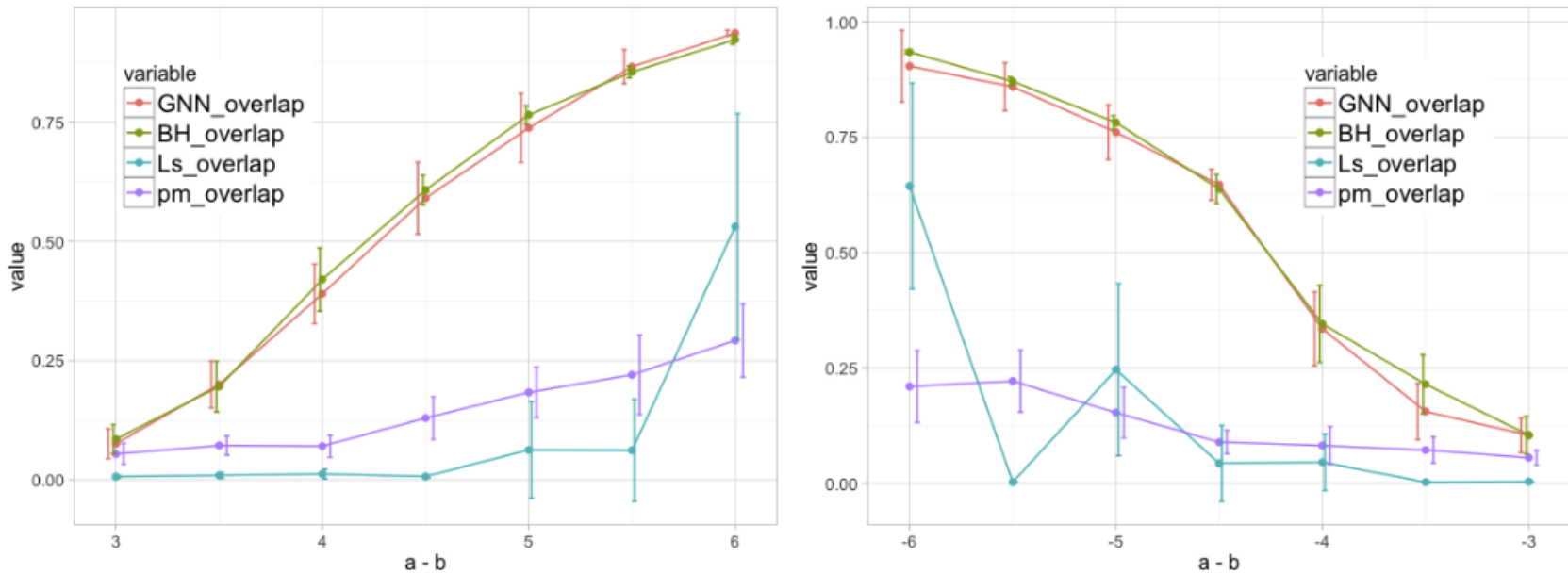
$$O_i = \mathrm{SoftMax}(\theta, H_i^{(T)}),$$

where $O_{i,c} = \frac{e^{H_i^{(T)} \theta_c}}{\sum_a e^{H_i^{(T)}} \theta_a}$ and $\theta \in \mathbb{R}^{d_H \times C}$ where C is the number of classes

$$\mathcal{L} = \sum_i \inf_{\sigma \in \Pi_C} -\log(O_{i,\sigma(y_i)})$$

where Π_C is the set of all permutations over the classes C

- Training data generated from Stochastic Block Model of various parameters
- Testing data also generated from Stochastic Model of the same size



Concrete examples of GNN in action: graph matching

- [A Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks, Alex Nowak, Soledad Villar, Afonso S. Bandeira and Joan Bruna, 2017]
- Architecture

$$H^{(0)} = \begin{bmatrix} degree_1 & \dots & degree_n \end{bmatrix}$$

$$H_1^{(t+1)} = \mathrm{ReLU}\left(H^{(t)} W^{(t,1)} + \mathrm{diag}(A\mathbb{1}) H^{(t)} W^{(t,2)} + \frac{1}{n} \mathbb{1}\mathbb{1}^T H^{(t)} W^{(t,3)} + H^{(t)} W^{(t,4+j)} \right)$$

$$H_2^{(t+1)} = H^{(t)} W^{(t,1)} + \mathrm{diag}(A\mathbb{1}) H^{(t)} W^{(t,2)} + \frac{1}{n} \mathbb{1}\mathbb{1}^T H^{(t)} W^{(t,3)} + H^{(t)} W^{(t,4+j)}$$

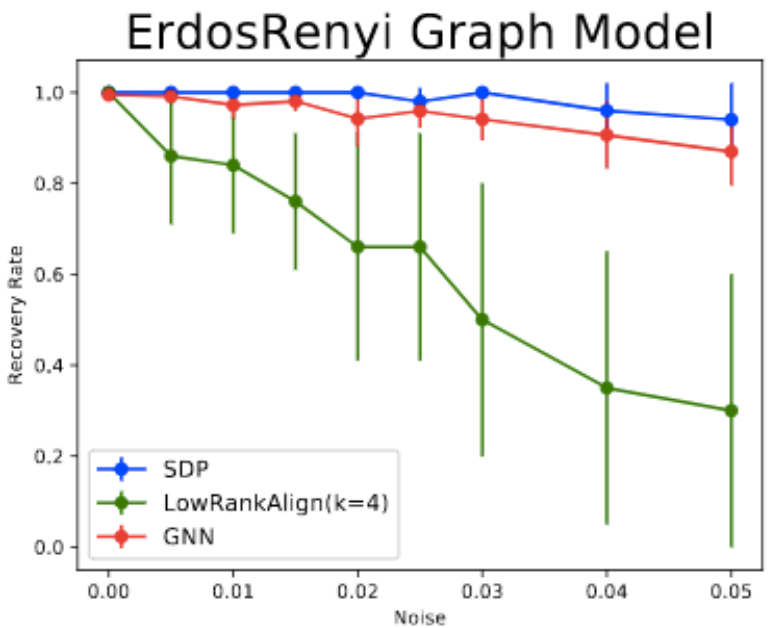
$$H^{(t+1)} = \begin{bmatrix} H_1^{(t+1)} & H_2^{(t+1)} \end{bmatrix}$$

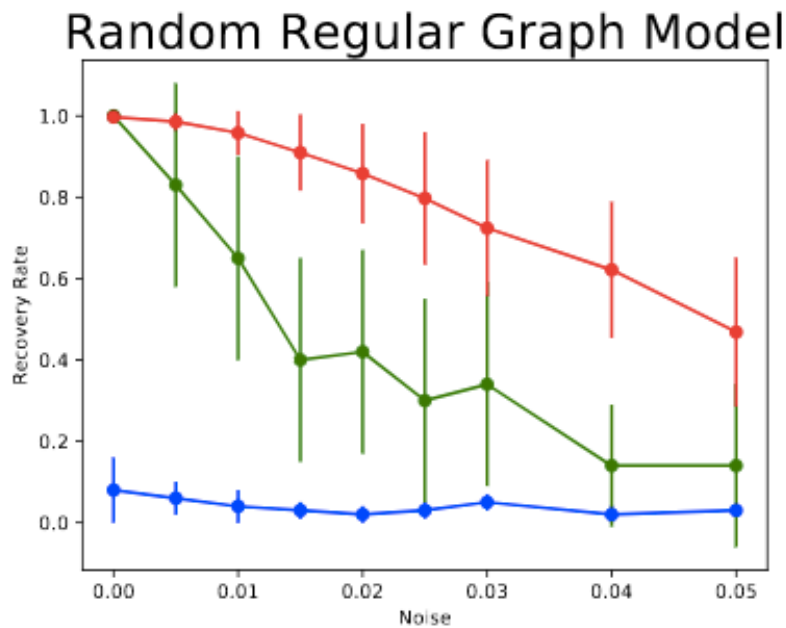
run it on both graphs G_A and G_B and then compute $M = (H_A^{(T)})^T H_B^{(T)} \in \mathbb{R}^{n \times n}$ Then take SoftMax on each row to map from G_A to G_B

$$O_i = \mathrm{SoftMax}(M_{i\cdot}),$$

$$\mathcal{L} = \sum_{i=1}^{n} -\log(O_{i,y_i})$$

- Trained on Erdos-Renyi + Noise and test on the same
- Trained on Random Regular graphs + Noise and tested on the same





Concrete examples of GNN in action: Quantum Chemistry

- [Neural Message Passing for Quantum Chemistry, Gilmer, Schoenholz, Riley, Vinyals, Dahl, 2017]
  - Input: chemical network with node features in {H, C, N, O, F }, and edges of bond types {single,double, triple, or aromatic} and also edge distances.
  - Output: estimated chemical properties
  - Training data:

$$
\begin{aligned}
H_i^{(t+1)} &= \sum_{j \in N(i)} M_t(\tilde{H}_i^{(t)}, \tilde{H}_j^{(t)}, A_{ij}) \\
\tilde{H}_i^{(t+1)} &= U_t(\tilde{H}_i^{(t)}, H_i^{(t+1)})
\end{aligned}
$$

$$
O = R(H^{(T)})
$$

with the choice of

$$
M_t(H_i^{(t)}, H_j^{(t)}, A_{ij}) = f_W(A_{ij}) H_j^{(t)}
$$

$$
R(H^{(T)}) = f_{W'}(\sum_i H_i^{(T)})
$$

- Trained and tested on benchmark dataset

Neural Message Passing for Quantum Chemistry

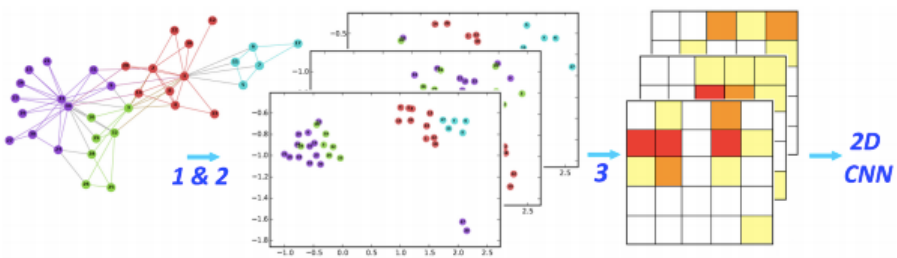
Table 2. Comparison of Previous Approaches (left) with MPNN baselines (middle) and our methods (right)

Target	BAML	BOB	CM	ECFP4	HDAD	GC	GG-NN	DTNN	enn-s2s	enn-s2s-ens5
mu	4.34	4.23	4.49	4.82	3.34	0.70	1.22	-	**0.30**	0.20
alpha	3.01	2.98	4.33	34.54	1.75	2.27	1.55	-	**0.92**	0.68
HOMO	2.20	2.20	3.09	2.89	1.54	1.18	1.17	-	**0.99**	0.74
LUMO	2.76	2.74	4.26	3.10	1.96	1.10	1.08	-	**0.87**	0.65
gap	3.28	3.41	5.32	3.86	2.49	1.78	1.70	-	**1.60**	1.23
R2	3.25	0.80	2.83	90.68	1.35	4.73	3.99	-	**0.15**	0.14
ZPVE	3.31	3.40	4.80	241.58	1.91	9.75	2.52	-	**1.27**	1.10
U0	1.21	1.43	2.98	85.01	0.58	3.02	0.83	-	**0.45**	0.33
U	1.22	1.44	2.99	85.59	0.59	3.16	0.86	-	**0.45**	0.34
H	1.22	1.44	2.99	86.21	0.59	3.19	0.81	-	**0.39**	0.30
G	1.20	1.42	2.97	78.36	0.59	2.95	0.78	.84[2]	**0.44**	0.34
Cv	1.64	1.83	2.36	30.29	0.88	1.45	1.19	-	**0.80**	0.62
Omega	0.27	0.35	1.32	1.47	0.34	0.32	0.53	-	**0.19**	0.15
Average	2.17	2.08	3.37	53.97	1.35	2.59	1.36	-	**0.68**	0.52

ICLR 2018

https://openreview.net/group?id=ICLR.cc/2018/Conference

- "Classifying Graphs as Images with Convolutional Neural Networks"
- Antoine Jean-Pierre Tixier, Giannis Nikolentzos, Polykarpos Meladianos, Michalis Vazirgiannis
  - Graph classification (N graphs with n nodes each)
  - graph kernels are
    - ★ slow: N^2 comparisons, each costing n^4 for shortest paths kernels
    - ★ SVM step can take $N^2 \sim N^3$
    - ★ feature learning and classification is separated
    - ★ Kernels tend to capture local structures (to keep complexity small)
  - the paper proposes [GraphEmbedding]-[2D-PCA]-[Histogram]-[CNN]



    - ★ GNN captures global structure
    - ★ end-to-end training
    - ★ GNN is faster

Dataset	IMDB-B	COLLAB	REDDIT-B	REDDIT-5K	REDDIT-12K
Max # vertices	136	492	3782	3648	3782
Min # vertices	12	32	6	22	2
Average # vertices	19.77	74.49	429.61	508.50	391.40
Max # edges	1249	40120	4071	4783	5171
Min # edges	26	60	4	21	1
Average # edges	96.53	2457.78	497.75	594.87	456.89
# graphs	1000	5000	2000	4999	11929
# classes	2	3	2	5	11
Max class imbalance	1:1	1:3.4	1:1	1:1	1:5

Table 3: 10-fold CV average test set classification accuracy of our proposed method compared to state-of-the-art graph kernels and graph CNN. $\pm$ is standard deviation. Best performance per column in **bold**. *indicates stat. sign. at the $p < 0.05$ level (our 2D CNN vs. WL) using the Mann-Whitney U test (`https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.stats.mannwhitneyu.html`).

Method \ Dataset	REDDIT-B (size=2,000;nclasses=2)	REDDIT-5K (4,999;5)	REDDIT-12K (11,929;11)	COLLAB (5,000;3)	IMDB-B (1,000;2)
Graphlet Shervashidze2009	77.26 ($\pm$ 2.34)	39.75 ($\pm$ 1.36)	25.98 ($\pm$ 1.29)	73.42 ($\pm$ 2.43)	65.40 ($\pm$ 5.95)
WL Shervashidze2011	78.52 ($\pm$ 2.01)	50.77 ($\pm$ 2.02)	34.57 ($\pm$ 1.32)	**77.82*** ($\pm$ 1.45)	**71.60** ($\pm$ 5.16)
Deep GK Yanardag2015	78.04 ($\pm$ 0.39)	41.27 ($\pm$ 0.18)	32.22 ($\pm$ 0.10)	73.09 ($\pm$ 0.25)	66.96 ($\pm$ 0.56)
PSCN $k = 10$ Niepert2016	86.30 ($\pm$ 1.58)	49.10 ($\pm$ 0.70)	41.32 ($\pm$ 0.42)	72.60 ($\pm$ 2.15)	71.00 ($\pm$ 2.29)
2D CNN (our method)	**89.12*** ($\pm$ 1.70)	**52.11** ($\pm$ 2.24)	**48.13*** ($\pm$ 1.47)	70.28 ($\pm$ 1.21)	70.40 ($\pm$ 3.85)

	REDDIT-B	REDDIT-5K	REDDIT-12K	COLLAB	IMDB-B
Size, average (# nodes, # edges)	2000, (430,498)	4999, (509,595)	11929, (391,457)	5000, (74,2458)	1000, (20,97)
Input shapes (for our approach)	(5,62,62)	(2,65,65)	(5,73,73)	(5,36,36)	(5,37,37)
Graphlet Shervashidze2009	551	5,046	12,208	3,238	275
WL Shervashidze2011	645	5,087	20,392	1,579	23
2D CNN (our approach)	6	16	52	5	1

Table 4: Runtimes in seconds, rounded to the nearest integer. For the graph kernel baselines, time necessary to populate the Kernel matrix (8-thread 3.4GHz CPU). For our model, time per epoch (Titan Xp GPU).

- "Graph Attention Networks"
- Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio
- node classification (semi-supervised learning)

$$\alpha_{ij} = \frac{\exp\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)}$$

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\right).$$

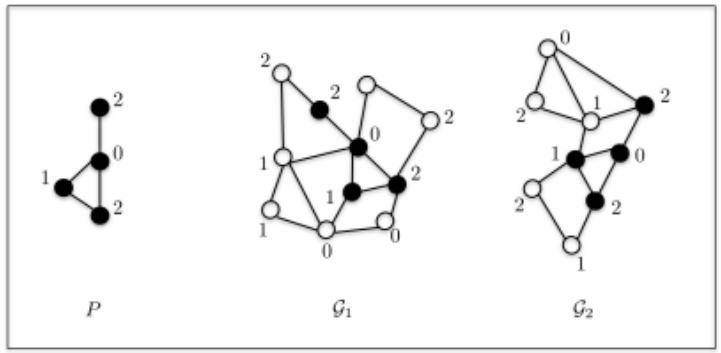
$$\vec{h}'_i = \Big\|_{k=1}^{K}\sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

Transductive

Method	**Cora**	**Citeseer**
MLP	55.1%	46.5%
ManiReg (Belkin et al., 2006)	59.5%	60.1%
SemiEmb (Weston et al., 2012)	59.0%	59.6%
LP (Zhu et al., 2003)	68.0%	45.3%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%
ICA (Lu & Getoor, 2003)	75.1%	69.1%
Planetoid (Yang et al., 2016)	75.7%	64.7%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%
GCN (Kipf & Welling, 2017)	81.5%	70.3%
GAT (ours)	**83.3%**	**74.0%**
improvement w.r.t GCN	1.8%	3.7%

- "RESIDUAL GATED GRAPH CONVNETS"
- subgraph matching [in Scarselli et al. (2009)]



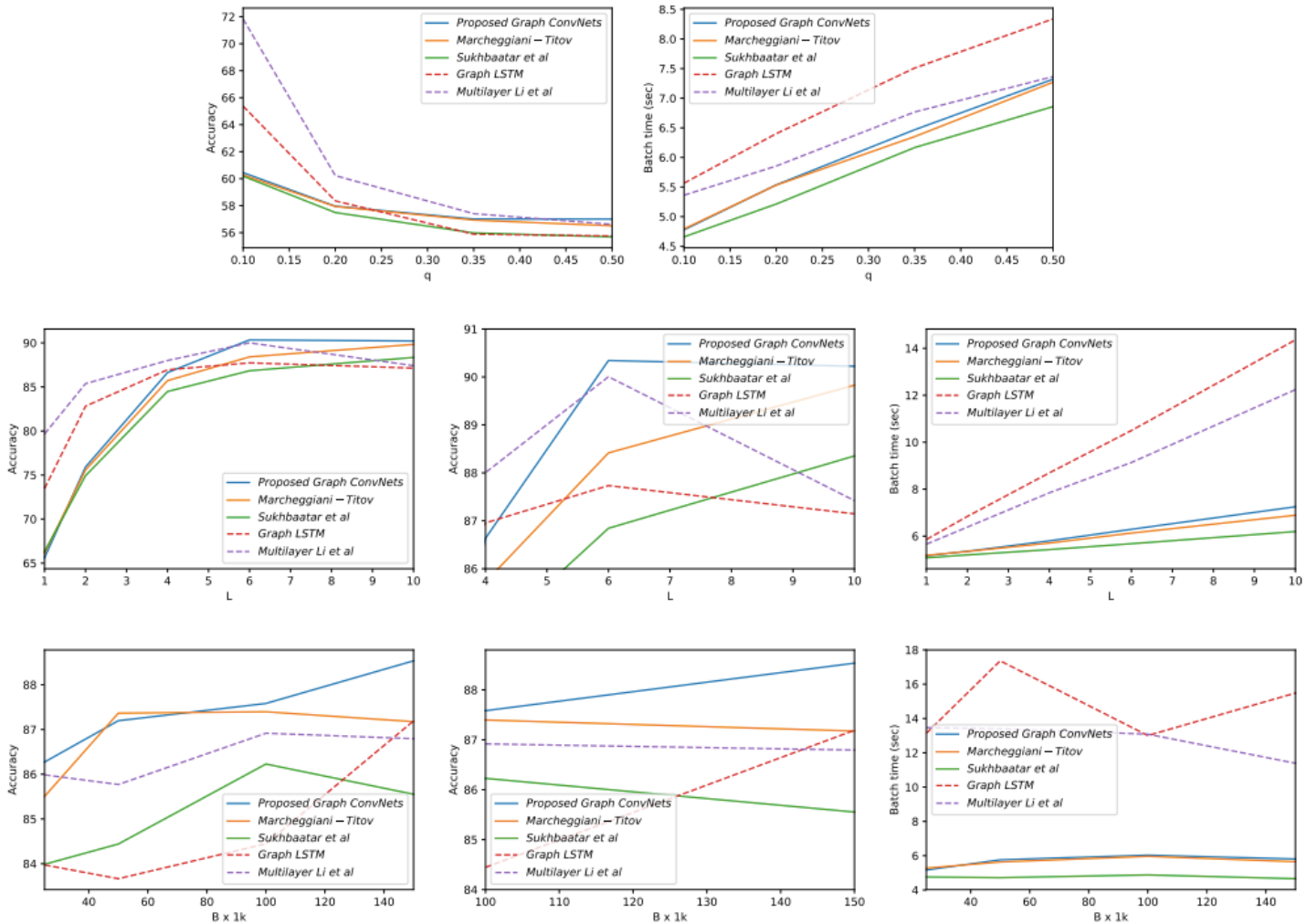
- we generate a subgraph P of 20 nodes with a SBM q = 0.5, and the signal on P is generated with a uniform random distribution with a vocabulary of size 3, i.e. $\{0, 1, 2\}$.
- Larger graphs Gk are composed of 10 communities with sizes randomly generated between 15 and 25. The SBM of each community is p = 0.5. The value of q, which acts as the noise level, is 0.1, unless otherwise specified. Finally, the signal on Gk is also randomly generated between $\{0, 1, 2\}$.

- architecture

$$h_i^{\ell+1} = f_{\text{G-GCNN}}^{\ell}\left(h_i^{\ell} \, , \, \{h_j^{\ell} : j \to i\} \right) \;\; = \;\; \mathbf{ReLU}\left(U^{\ell} h_i^{\ell} + \sum_{j \to i} \eta_{ij} \odot V^{\ell} h_j^{\ell} \right)$$

$$h_i^{\ell+1} \;\; = \;\; f^{\ell}\left(h_i^{\ell} \, , \, \{h_j^{\ell} : j \to i\} \right) + h_i^{\ell}.$$

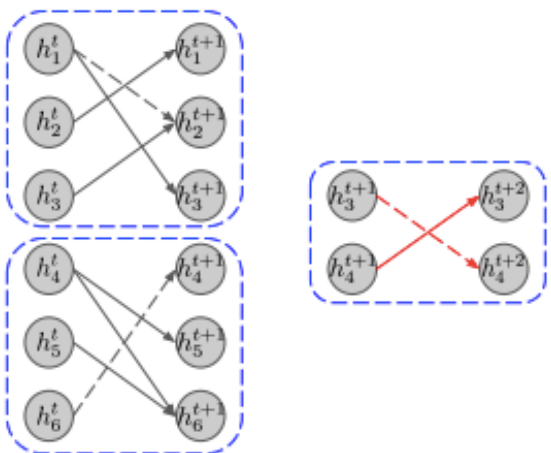
- performance



- "Graph Partition Neural Networks for Semi-Supervised Classification"
- standard graph neural network
  - could take long time to propagate (for a line graph n^2 messages sent)
- proposed GPNN

Algorithm 1 Graph Partition Propagation Schedule.

Input: K subgraphs $\{\mathcal{S}_k | k = 1, \ldots, K\}$, cut $\mathcal{S}_0$, outer propagation step limit T, intra-subgraph and inter-subgraph propagation step limits T_S and T_C.
for $t = 1, \ldots, T$ **do**
    for all $k \in \{1, \ldots, K\}$ **do in parallel**
        Call SYNCPROP within subgraph $\mathcal{S}_k$ for T_S steps.
    Call SYNCPROP within cut $\mathcal{S}_0$ for T_C steps.
function SYNCPROP
    Compute & send messages as in Eq. (1)
    Aggregate messages as in Eq. (2)
    Update states as in Eq. (3)

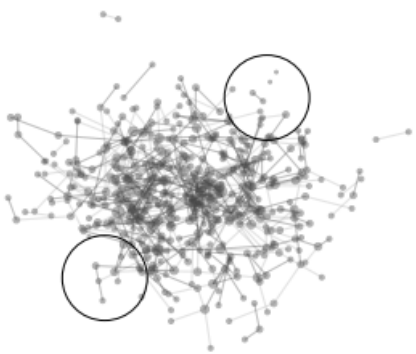


- Performance

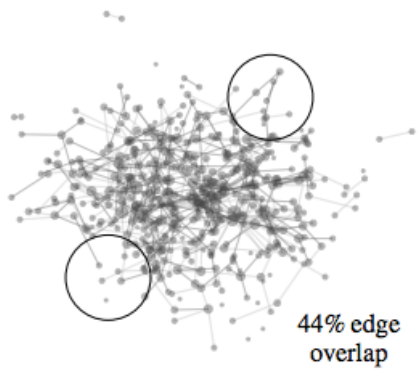
Method	(Source)	Citeseer	Cora	Pubmed	NELL		
					10%	1%	0.1%
Feat	(Yang et al., 2016)	57.2	57.4	69.8	62.1	40.4	21.7
ManiReg	(Belkin et al., 2006)	60.1	59.5	70.7	63.4	41.3	21.8
SemiEmb	(Weston et al., 2012)	59.6	59.0	71.1	65.4	43.8	26.7
LP	(Zhu et al., 2003)	45.3	68.0	63.0	71.4	44.8	26.5
DeepWalk	(Perozzi et al., 2014)	43.2	67.2	65.3	79.5	72.5	58.1
ICA	(Lu & Getoor, 2003)	69.1	75.1	73.9	–	–	–
Planetoid (Transductive)	(Yang et al., 2016)	64.9	75.7	75.7	84.5	**75.7**	61.9
Planetoid (Inductive)	(Yang et al., 2016)	64.7	61.2	77.2	70.2	59.8	45.4
GCN	(Kipf & Welling, 2017)	**70.3**	81.5	79.0	†83.0	†67.0	†54.2
GGNN*	(Li et al., 2016)	68.1	77.9	77.2	**84.6**	66.2	59.1
GPNN	(ours)	69.7	**81.9**	**79.2**	83.7	74.6	**63.1**

Method	Citeseer	Cora	Pubmed
GCN† (Kipf & Welling, 2017)	**68.7** ± 2.0	**80.4** ± 2.8	**77.5** ± 2.1
GGNN* (Li et al., 2016)	66.3 ± 2.0	78.9 ± 2.6	74.7 ± 2.8
GPNN	68.6 ± 1.7	79.9 ± 2.4	76.1 ± 2.0

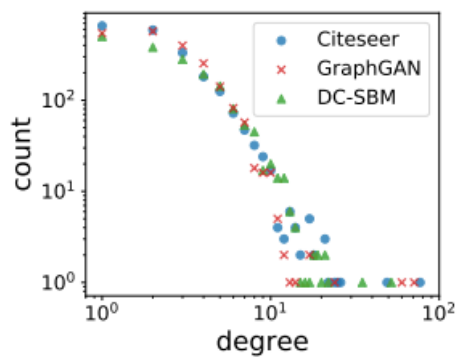
- "GraphGAN: Generating Graphs via Random Walks"
- generate sibling graphs, which have similar properties yet are not exact replicas of the original graph
- challenges
  - handle discrete objects
  - in a typical setting one has to learn from a single graph
  - any model operating on a graph necessarily has to be permutation invariant



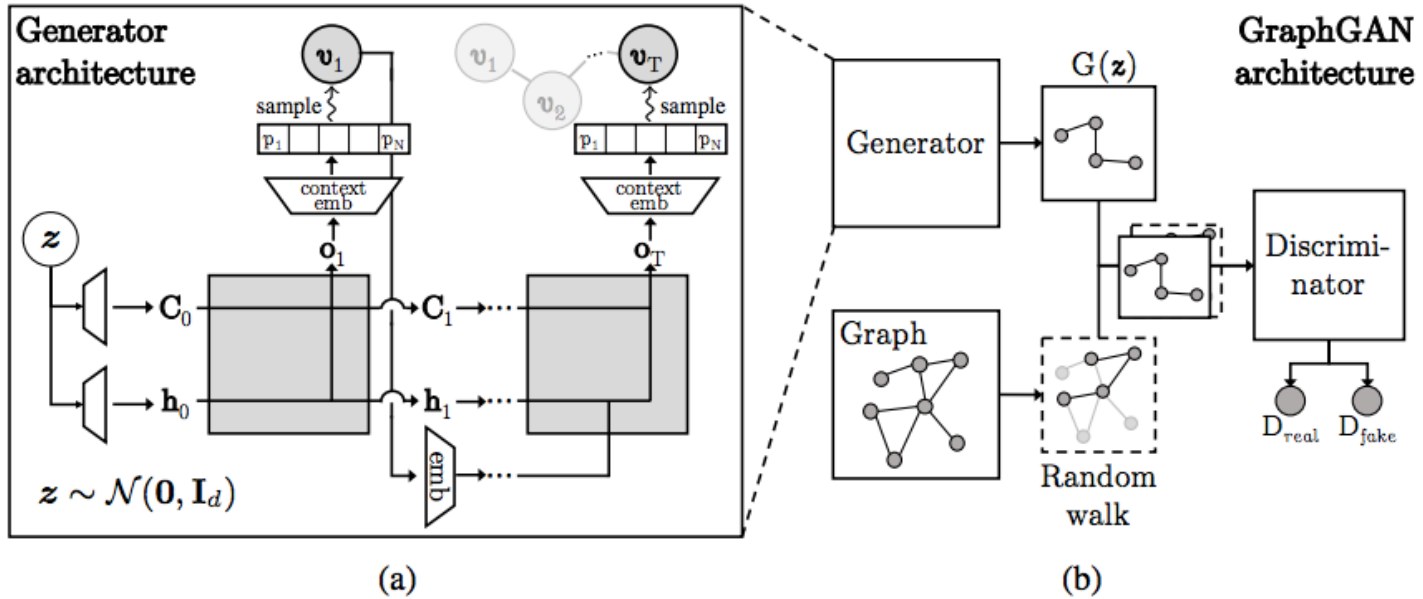
(a) Original graph



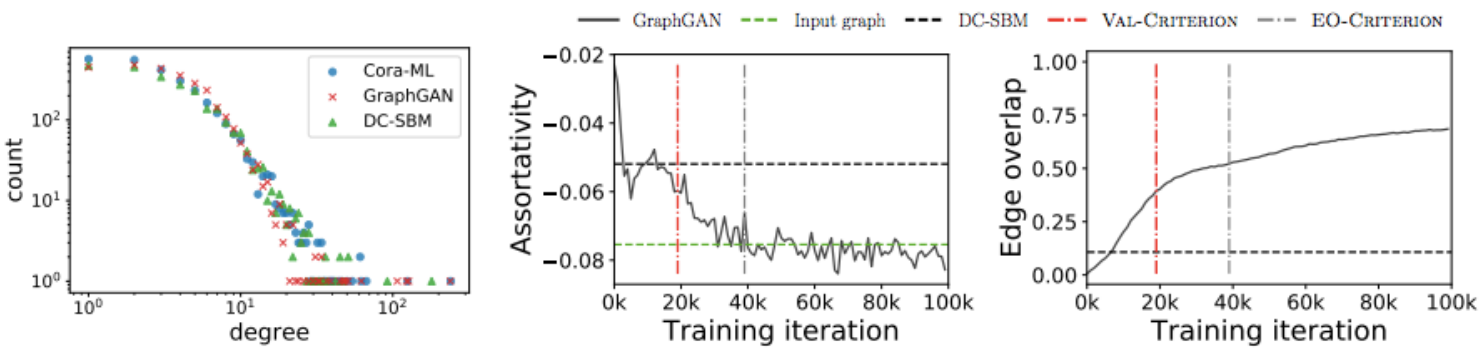
(b) Sibling graph



(c) Degree distribution comparison



Method	CORA-ML		CORA		CITESEER		DBLP		PUBMED		POLBLOGS	
	ROC	AP	ROC	AP	ROC	AP	ROC	AP	ROC	AP	ROC	AP
Adamic/Adar	92.16	85.43	93.00	86.18	88.69	77.82	91.13	82.48	84.98	70.14	85.43	92.16
DC-SBM	**96.03**	95.15	98.01	97.45	94.77	93.13	**97.05**	**96.57**	**96.76**	95.64	95.46	**94.93**
node2vec	92.19	91.76	**98.52**	**98.36**	95.29	94.58	96.41	96.36	96.49	**95.97**	85.10	83.54
GraphGAN (500K)	94.00	92.32	82.31	68.47	95.18	91.93	82.45	70.28	87.39	76.55	95.06	94.61
GraphGAN (100M)	95.19	**95.24**	84.82	88.04	**96.30**	**96.89**	86.61	89.21	93.41	94.59	**95.51**	94.83
GraphGAN (emb.)	90.29	88.29	84.38	79.36	92.95	92.44	86.59	81.96	91.79	89.37	70.01	62.72



(a) Degree distribution

(b) Assortativity over training iterations

(c) Edge overlap (EO) over training iterations

- "SPECTRALNET: SPECTRAL CLUSTERING USING DEEP NEURAL NETWORKS"

GNN with edge features

- original GCN Forward Pass:
  - $H^{(0)} = X$
  - Repeat for $t = 1, 2, \ldots, T$
    - ★ $\tilde{H}^{(t)} = PH^{(t-1)}$, with $P = D^{-1}A$
    - ★ $H^{(t)} = \text{ReLU}(\tilde{H}^{(t)} W^{(t)})$
  - $\text{SoftMax}(H^{(T)} W^{(T+1)})$

$$H_i^{(t+1)} = \text{ReLU}\left(\frac{1}{d_i} \sum_{j \in N(i)} H_j^{(t)} W^{(t+1)}\right)$$

- edge-feature $\{F_{ij}\}$ GCN Forward Pass:
  - $H^{(0)} = X$
  - Define $F_i = \sum_{j \in N(i)} F_{ij}$
  - Let $F = [F_1; F_2; \ldots F_n] \in \mathbb{R}^{n \times d_F}$
  - Repeat for $t = 1, 2, \ldots, T$
    - ★ $\tilde{H}^{(t)} = PH^{(t-1)}$, with $P = D^{-1}A$
    - ★ $H^{(t)} = \text{ReLU}(\tilde{H}^{(t)} W^{(t)} + F\tilde{W}^{(t)})$
  - $\text{SoftMax}(H^{(T)} W^{(T+1)} + F\tilde{W}^{(T+1)})$

$$H_i^{(t+1)} = \text{ReLU}\left(\frac{1}{d_i} \sum_{j \in N(i)} \{H_j^{(t)} W^{(t+1)} + F_{ij} \tilde{W}^{(t+1)}\}\right)$$