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://linq.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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Classes</th>
<th>Features</th>
<th>Labeled nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CiteSeer</td>
<td>3,327</td>
<td>4,732</td>
<td>6</td>
<td>3,703</td>
<td>120</td>
</tr>
<tr>
<td>Cora</td>
<td>2,708</td>
<td>5,429</td>
<td>7</td>
<td>1,433</td>
<td>140</td>
</tr>
<tr>
<td>PubMed</td>
<td>19,717</td>
<td>44,328</td>
<td>3</td>
<td>500</td>
<td>60</td>
</tr>
</tbody>
</table>
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?
Table 2: Classification accuracy in percent with a fixed split of data from (Yang et al., 2016).

<table>
<thead>
<tr>
<th>Input</th>
<th>Method</th>
<th>Citeseer</th>
<th>Cora</th>
<th>PubMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y^L_1, x^L_1$</td>
<td>Singlelayer Perceptron</td>
<td>57.2</td>
<td>57.4</td>
<td>69.8</td>
</tr>
<tr>
<td></td>
<td>Multilayer Perceptron</td>
<td>64.0</td>
<td>57.5</td>
<td>71.4</td>
</tr>
<tr>
<td>$y^L_1, x^{L+U}_1$</td>
<td>T-SVM (Joachims, 1999)</td>
<td>64.0</td>
<td>57.5</td>
<td>62.2</td>
</tr>
<tr>
<td>$y^L_1, G$</td>
<td>DeepWalk (Perozzi et al., 2014)</td>
<td>43.2</td>
<td>67.2</td>
<td>65.3</td>
</tr>
<tr>
<td>$y^L_1, x^{L+U}, G$</td>
<td>LP (Zhu et al., 2003)</td>
<td>45.3</td>
<td>68.0</td>
<td>63.0</td>
</tr>
<tr>
<td></td>
<td>ICA (Lu &amp; Getoor, 2003)</td>
<td>69.1</td>
<td>75.1</td>
<td>73.9</td>
</tr>
<tr>
<td></td>
<td>ManiReg (Belkin et al., 2006)</td>
<td>60.1</td>
<td>59.5</td>
<td>70.7</td>
</tr>
<tr>
<td></td>
<td>SemiEmb (Weston et al., 2012)</td>
<td>59.6</td>
<td>59.0</td>
<td>71.1</td>
</tr>
<tr>
<td></td>
<td>DCNN (Atwood &amp; Towsley, 2016)</td>
<td></td>
<td>76.8</td>
<td>73.0</td>
</tr>
<tr>
<td></td>
<td>Planetoid (Yang et al., 2016)</td>
<td>64.7</td>
<td>75.7</td>
<td>77.2</td>
</tr>
<tr>
<td></td>
<td>MoNet (Monti et al., 2016)</td>
<td>81.7</td>
<td></td>
<td>78.8</td>
</tr>
<tr>
<td></td>
<td>Graph-CNN (Such et al., 2017)</td>
<td></td>
<td>76.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DynamicFilter (Verma et al., 2017)</td>
<td></td>
<td>81.6</td>
<td>79.0</td>
</tr>
<tr>
<td></td>
<td>Bootstrap (Buchnik &amp; Cohen, 2017)</td>
<td>53.6</td>
<td>78.4</td>
<td>78.8</td>
</tr>
<tr>
<td></td>
<td>GCN (Kipf &amp; Welling, 2016)</td>
<td>70.3</td>
<td>81.5</td>
<td>79.0</td>
</tr>
<tr>
<td>GLN</td>
<td></td>
<td>70.9±.05</td>
<td>81.2±.05</td>
<td>78.9±.05</td>
</tr>
<tr>
<td>AGNN (this paper)</td>
<td></td>
<td>71.7±.08</td>
<td>82.6±.09</td>
<td>79.9±.07</td>
</tr>
</tbody>
</table>
GCN:
[Input]-[Propagation]-[Perceptron]-[Propagation]-[Perceptron]- · · · -[SoftMax]-[Output]
Forward Pass:
- \( H^{(0)} = X \)
- Repeat for \( t = 1, 2, \ldots, T \)
  - \( \tilde{H}^{(t)} = PH^{(t-1)} \), with \( P = D^{-1}A \)
  - \( H^{(t)} = \text{ReLU}(\tilde{H}^{(t)}W^{(t)}) \)
- SoftMax\((H^{(T)}W^{(T+1)})\)
Training:
- Let \( Z = f_W(X, A) \)
- the weights \( W^{(1)}, W^{(2)}, \ldots, W^{(T+1)} \) 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

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

- 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

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

with \( B_{ij} = e^{-(X_i - X_j)^2 / \sigma^2} \).

This is a very popular approach known as \textbf{Label Propagation} \cite{label_propagation, zhu2002}

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** \( A = \text{diag}(B 1) - B \)

and consider binary classification where 
\[
\begin{align*}
  f &= [f(1), \ldots, f(n)] 
  \in \{0, 1\}^n 
\end{align*}
\]

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 \( \mathcal{L}(f_U) = f_L^TA_{LL}f_L + 2f_L^TA_{LU}f_U + f_U^TA_{UU}f_U \)

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 s into $k$ classes

Spectral clustering:
consider binary classification for now and let $Z_i \in \{-1, +1\}$

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

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

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

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

\[
\text{minimize } Z^T L Z
\]

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

\[
\text{minimize } Z^T L Z
\]

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 \( \mathbf{1} \)

\[
\text{minimize } Z^T L Z
\]

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

\[ H^{(0)} = \begin{bmatrix} \text{degree}_1 & \ldots & \text{degree}_n \end{bmatrix} \]

\[ H_{1}^{(t+1)} = \text{ReLU} \left( H^{(t)} W^{(t,1)} + \text{diag}(A1) H^{(t)} W^{(t,2)} + \frac{1}{n} 1 1^T H^{(t)} W^{(t,3)} + H^{(t)} W^{(t,4+j)} \right) \]

\[ H_{2}^{(t+1)} = H^{(t)} W^{(t,1)} + \text{diag}(A1) H^{(t)} W^{(t,2)} + \frac{1}{n} 1 1^T H^{(t)} W^{(t,3)} + H^{(t)} W^{(t,4+j)} \]

\[ H^{(t+1)} = \begin{bmatrix} H_{1}^{(t+1)} & H_{2}^{(t+1)} \end{bmatrix} \]

\[ O_i = \text{SoftMax}(\theta, H_i^{(T)}) , \]

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 \( \Pi_C \) 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

\[
\begin{align*}
H^{(0)} &= \begin{bmatrix} \text{degree}_1 & \ldots & \text{degree}_n \end{bmatrix} \\
H_1^{(t+1)} &= \text{ReLU}\left( H^{(t)} W^{(t,1)} + \text{diag}(A1) H^{(t)} W^{(t,2)} + \frac{1}{n} \mathbb{1} \mathbb{1}^T H^{(t)} W^{(t,3)} + H^{(t)} W^{(t,4+j)} \right) \\
H_2^{(t+1)} &= H^{(t)} W^{(t,1)} + \text{diag}(A1) H^{(t)} W^{(t,2)} + \frac{1}{n} \mathbb{1} \mathbb{1}^T H^{(t)} W^{(t,3)} + H^{(t)} W^{(t,4+j)} \\
H^{(t+1)} &= \begin{bmatrix} H_1^{(t+1)} & H_2^{(t+1)} \end{bmatrix}
\end{align*}
\]

Run it on both graphs \( G_A \) and \( G_B \) and then compute

\[
M = \left( H_A^{(T)} \right)^T H_B^{(T)} \in \mathbb{R}^{n \times n}
\]

Then take SoftMax on each row to map from \( G_A \) to \( G_B \)

\[
O_i = \text{SoftMax}(M_{i,\cdot}),
\]

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

  - 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:

\[
\begin{align*}
H_i^{(t+1)} &= \sum_{j \in N(i)} M_t(\tilde{H}_i^{(t)}, \tilde{H}_j^{(t)}, A_{ij}) \\
\tilde{H}_i^{(t+1)} &= U_t(\tilde{H}_i^{(t)}, H_i^{(t+1)}) \\
O &= R(H^{(T)})
\end{align*}
\]

with the choice of

\[
M_t(H_i^{(t)}, H_j^{(t)}, A_{ij}) = f(W(A_{ij})H_j^{(t)})
\]

\[
R(H^{(T)}) = f_W\left(\sum_i H_i^{(T)}\right)
\]
Trained and tested on benchmark dataset

<table>
<thead>
<tr>
<th>Target</th>
<th>BAML</th>
<th>BOB</th>
<th>CM</th>
<th>ECFP4</th>
<th>HDAD</th>
<th>GC</th>
<th>GG-NN</th>
<th>DTNN</th>
<th>enn-s2s</th>
<th>enn-s2s-ens5</th>
</tr>
</thead>
<tbody>
<tr>
<td>mu</td>
<td>4.34</td>
<td>4.23</td>
<td>4.49</td>
<td>4.82</td>
<td>3.34</td>
<td>0.70</td>
<td>1.22</td>
<td>-</td>
<td>0.30</td>
<td>0.20</td>
</tr>
<tr>
<td>alpha</td>
<td>3.01</td>
<td>2.98</td>
<td>4.33</td>
<td>34.54</td>
<td>1.75</td>
<td>2.27</td>
<td>1.55</td>
<td>-</td>
<td>0.92</td>
<td>0.68</td>
</tr>
<tr>
<td>HOMO</td>
<td>2.20</td>
<td>2.20</td>
<td>3.09</td>
<td>2.89</td>
<td>1.54</td>
<td>1.18</td>
<td>1.17</td>
<td>-</td>
<td>0.99</td>
<td>0.74</td>
</tr>
<tr>
<td>LUMO</td>
<td>2.76</td>
<td>2.74</td>
<td>4.26</td>
<td>3.10</td>
<td>1.96</td>
<td>1.10</td>
<td>1.08</td>
<td>-</td>
<td>0.87</td>
<td>0.65</td>
</tr>
<tr>
<td>gap</td>
<td>3.28</td>
<td>3.41</td>
<td>5.32</td>
<td>3.86</td>
<td>2.49</td>
<td>1.78</td>
<td>1.70</td>
<td>-</td>
<td>1.60</td>
<td>1.23</td>
</tr>
<tr>
<td>R2</td>
<td>3.25</td>
<td>0.80</td>
<td>2.83</td>
<td>90.68</td>
<td>1.35</td>
<td>4.73</td>
<td>3.99</td>
<td>-</td>
<td>0.15</td>
<td>0.14</td>
</tr>
<tr>
<td>ZPVE</td>
<td>3.31</td>
<td>3.40</td>
<td>4.80</td>
<td>241.58</td>
<td>1.91</td>
<td>9.75</td>
<td>2.52</td>
<td>-</td>
<td>1.27</td>
<td>1.10</td>
</tr>
<tr>
<td>U0</td>
<td>1.21</td>
<td>1.43</td>
<td>2.98</td>
<td>85.01</td>
<td>0.58</td>
<td>3.02</td>
<td>0.83</td>
<td>-</td>
<td>0.45</td>
<td>0.33</td>
</tr>
<tr>
<td>U</td>
<td>1.22</td>
<td>1.44</td>
<td>2.99</td>
<td>85.59</td>
<td>0.59</td>
<td>3.16</td>
<td>0.86</td>
<td>-</td>
<td>0.45</td>
<td>0.34</td>
</tr>
<tr>
<td>H</td>
<td>1.22</td>
<td>1.44</td>
<td>2.99</td>
<td>86.21</td>
<td>0.59</td>
<td>3.19</td>
<td>0.81</td>
<td>-</td>
<td>0.39</td>
<td>0.30</td>
</tr>
<tr>
<td>G</td>
<td>1.20</td>
<td>1.42</td>
<td>2.97</td>
<td>78.36</td>
<td>0.59</td>
<td>2.95</td>
<td>0.78</td>
<td>.84^2</td>
<td>0.44</td>
<td>0.34</td>
</tr>
<tr>
<td>Cv</td>
<td>1.64</td>
<td>1.83</td>
<td>2.36</td>
<td>30.29</td>
<td>0.88</td>
<td>1.45</td>
<td>1.19</td>
<td>-</td>
<td>0.80</td>
<td>0.62</td>
</tr>
<tr>
<td>Omega</td>
<td>0.27</td>
<td>0.35</td>
<td>1.32</td>
<td>1.47</td>
<td>0.34</td>
<td>0.32</td>
<td>0.53</td>
<td>-</td>
<td>0.19</td>
<td>0.15</td>
</tr>
<tr>
<td>Average</td>
<td>2.17</td>
<td>2.08</td>
<td>3.37</td>
<td>53.97</td>
<td>1.35</td>
<td>2.59</td>
<td>1.36</td>
<td>-</td>
<td>0.68</td>
<td>0.52</td>
</tr>
</tbody>
</table>
"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
<table>
<thead>
<tr>
<th>Dataset</th>
<th>IMDB-B</th>
<th>COLLAB</th>
<th>REDDIT-B</th>
<th>REDDIT-5K</th>
<th>REDDIT-12K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max # vertices</td>
<td>136</td>
<td>492</td>
<td>3782</td>
<td>3648</td>
<td>3782</td>
</tr>
<tr>
<td>Min # vertices</td>
<td>12</td>
<td>32</td>
<td>6</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>Average # vertices</td>
<td>19.77</td>
<td>74.49</td>
<td>429.61</td>
<td>508.50</td>
<td>391.40</td>
</tr>
<tr>
<td>Max # edges</td>
<td>1249</td>
<td>40120</td>
<td>4071</td>
<td>4783</td>
<td>5171</td>
</tr>
<tr>
<td>Min # edges</td>
<td>26</td>
<td>60</td>
<td>4</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>Average # edges</td>
<td>96.53</td>
<td>2457.78</td>
<td>497.75</td>
<td>594.87</td>
<td>456.89</td>
</tr>
<tr>
<td># graphs</td>
<td>1000</td>
<td>5000</td>
<td>2000</td>
<td>4999</td>
<td>11929</td>
</tr>
<tr>
<td># classes</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>Max class imbalance</td>
<td>1:1</td>
<td>1:3.4</td>
<td>1:1</td>
<td>1:1</td>
<td>1:5</td>
</tr>
</tbody>
</table>
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. ± 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).

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

<table>
<thead>
<tr>
<th>Size, average (# nodes, # edges)</th>
<th>REDDIT-B</th>
<th>REDDIT-5K</th>
<th>REDDIT-12K</th>
<th>COLLAB</th>
<th>IMDB-B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input shapes (for our approach)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphlet Shervashidze2009</td>
<td>551</td>
<td>5,046</td>
<td>12,208</td>
<td>3,238</td>
<td>275</td>
</tr>
<tr>
<td>WL Shervashidze2011</td>
<td>645</td>
<td>5,087</td>
<td>20,392</td>
<td>1,579</td>
<td>23</td>
</tr>
<tr>
<td>2D CNN (our approach)</td>
<td>6</td>
<td>16</td>
<td>52</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

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)

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

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

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

\[
\vec{h}_i' = \sigma \left( \frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j \right)
\]
<table>
<thead>
<tr>
<th>Method</th>
<th>Cora</th>
<th>Citeseer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>55.1%</td>
<td>46.5%</td>
</tr>
<tr>
<td>ManiReg (Belkin et al., 2006)</td>
<td>59.5%</td>
<td>60.1%</td>
</tr>
<tr>
<td>SemiEmb (Weston et al., 2012)</td>
<td>59.0%</td>
<td>59.6%</td>
</tr>
<tr>
<td>LP (Zhu et al., 2003)</td>
<td>68.0%</td>
<td>45.3%</td>
</tr>
<tr>
<td>DeepWalk (Perozzi et al., 2014)</td>
<td>67.2%</td>
<td>43.2%</td>
</tr>
<tr>
<td>ICA (Lu &amp; Getoor, 2003)</td>
<td>75.1%</td>
<td>69.1%</td>
</tr>
<tr>
<td>Planetoid (Yang et al., 2016)</td>
<td>75.7%</td>
<td>64.7%</td>
</tr>
<tr>
<td>Chebyshev (Defferrard et al., 2016)</td>
<td>81.2%</td>
<td>69.8%</td>
</tr>
<tr>
<td>GCN (Kipf &amp; Welling, 2017)</td>
<td>81.5%</td>
<td>70.3%</td>
</tr>
<tr>
<td><strong>GAT (ours)</strong></td>
<td><strong>83.3%</strong></td>
<td><strong>74.0%</strong></td>
</tr>
<tr>
<td>improvement w.r.t GCN</td>
<td>1.8%</td>
<td>3.7%</td>
</tr>
</tbody>
</table>
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 $G_k$ 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 $G_k$ is also randomly generated between $\{0, 1, 2\}$.
architecture

\[ h_{i}^{\ell+1} = f_{G-GCNN}^{\ell}( h_{i}^{\ell}, \{ h_{j}^{\ell} : j \to i \} ) = \text{ReLU} \left( U^{\ell} h_{i}^{\ell} + \sum_{j \to i} \eta_{ij} \odot V^{\ell} h_{j}^{\ell} \right) \]

\[ h_{i}^{\ell+1} = f^{\ell}( h_{i}^{\ell}, \{ h_{j}^{\ell} : j \to i \} ) + h_{i}^{\ell}. \]
performance

supervised learning
"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.

1: **Input:** $K$ subgraphs $\{S_k\}_{k=1}^K$, cut $S_0$, outer propagation step limit $T$, intra-subgraph and inter-subgraph propagation step limits $T_S$ and $T_C$.
2: **for** $t = 1, \ldots, T$ **do**
3:   **for all** $k \in \{1, \ldots, K\}$ **do in parallel**
4:     Call **SYNC**PROP within subgraph $S_k$ for $T_S$ steps.
5:     Call **SYNC**PROP within cut $S_0$ for $T_C$ steps.
6: **function** **SYNC**PROP
7:     Compute & send messages as in Eq. (1)
8:     Aggregate messages as in Eq. (2)
9:     Update states as in Eq. (3)
## Performance

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

<table>
<thead>
<tr>
<th>Method</th>
<th></th>
<th>Citeseer</th>
<th>Cora</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN†</td>
<td>(Kipf &amp; Welling, 2017)</td>
<td><strong>68.7 ± 2.0</strong></td>
<td><strong>80.4 ± 2.8</strong></td>
<td><strong>77.5 ± 2.1</strong></td>
</tr>
<tr>
<td>GGNN*†</td>
<td>(Li et al., 2016)</td>
<td>66.3 ± 2.0</td>
<td>78.9 ± 2.6</td>
<td>74.7 ± 2.8</td>
</tr>
<tr>
<td>GPNN</td>
<td></td>
<td>68.6 ± 1.7</td>
<td>79.9 ± 2.4</td>
<td>76.1 ± 2.0</td>
</tr>
</tbody>
</table>
"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

![GraphGAN examples](image)

(a) Original graph  
(b) Sibling graph  
(c) Degree distribution comparison
"SPECREDNET: 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 $P = D^{-1}A$
    - $H^{(t)} = \text{ReLU}(\tilde{H}^{(t)}W^{(t)})$
  - SoftMax($H^{(T)}W^{(T+1)}$)

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

- 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 $P = D^{-1}A$
    - $H^{(t)} = \text{ReLU}(\tilde{H}^{(t)}W^{(t)} + F\tilde{W}^{(t)})$
  - SoftMax($H^{(T)}W^{(T+1)} + F\tilde{W}^{(T+1)}$)

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

supervised learning