Deep Learning methods for Graphs and Sets

G. Nikolentzos and M. Vazirgiannis

LIX, École Polytechnique

ALTEGRAD 2023-2024

Outline

Outline

Message Passing Neural Networks for Learning Node Representations

Consist of a series of message passing layers usually followed by one or more fully-connected layers

The message passing phase runs for T time steps and updates the representation of each vertex h_v^t based on its previous representation and the representations of its neighbors:

$$\mathsf{m}_v^{(t+1)} = \mathrm{AGGREGATE}\left(\left\{\mathsf{h}_u^{(t)} \middle| u \in \mathcal{N}(v)\right\}\right)$$

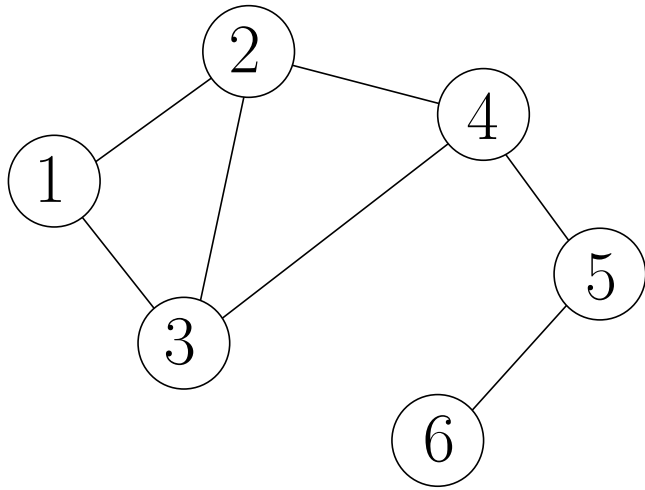
$$\mathsf{h}_v^{(t+1)} = \mathrm{COMBINE}\left(\mathsf{h}_v^{(t)}, \mathsf{m}_v^{(t+1)}\right)$$

where $\mathcal{N}(v)$ is the set of neighbors of v, and $\mathrm{AGGREGATE}$ and $\mathrm{COMBINE}$ are message functions and vertex update functions respectively

* a node's neighbors have no natural ordering
$\hookrightarrow$ the $\mathrm{AGGREGATE}$ function operates over an unordered set of vectors $\rightarrow$ must be invariant to permutations of the neighbors

Example of Message Passing Layer

$$
\begin{aligned}
h_1^{(t+1)} &= f(W_0^{(t)} h_1^{(t)} + W_1^{(t)} h_2^{(t)} + W_1^{(t)} h_3^{(t)}) \\
h_2^{(t+1)} &= f(W_0^{(t)} h_2^{(t)} + W_1^{(t)} h_1^{(t)} + W_1^{(t)} h_3^{(t)} + W_1^{(t)} h_4^{(t)}) \\
h_3^{(t+1)} &= f(W_0^{(t)} h_3^{(t)} + W_1^{(t)} h_1^{(t)} + W_1^{(t)} h_2^{(t)} + W_1^{(t)} h_4^{(t)}) \\
h_4^{(t+1)} &= f(W_0^{(t)} h_4^{(t)} + W_1^{(t)} h_2^{(t)} + W_1^{(t)} h_3^{(t)} + W_1^{(t)} h_5^{(t)}) \\
h_5^{(t+1)} &= f(W_0^{(t)} h_5^{(t)} + W_1^{(t)} h_4^{(t)} + W_1^{(t)} h_6^{(t)}) \\
h_6^{(t+1)} &= f(W_0^{(t)} h_6^{(t)} + W_1^{(t)} h_5^{(t)})
\end{aligned}
$$



Remark: Biases are omitted for clarity

Graph Convolutional Network (GCN)

Each message passing layer of the GCN model is defined as:

$$\mathrm{h}_v^{(t+1)} = \mathrm{ReLU}\left(\mathrm{W}^{(t)} \frac{1}{1+d(v)} \mathrm{h}_v^{(t)} + \sum_{u \in \mathcal{N}(v)} \mathrm{W}^{(t)} \frac{1}{\sqrt{(1+d(v))(1+d(u))}} \mathrm{h}_u^{(t)}\right)$$

where $d(v)$ is the degree of node v

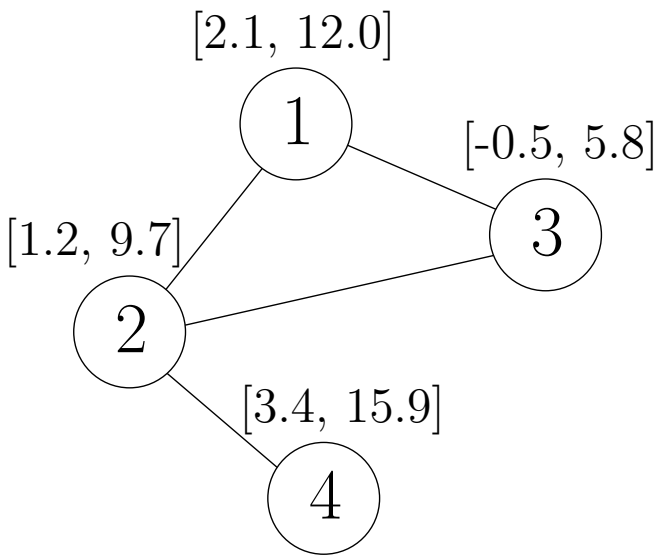
In matrix form, the above is equivalent to:

$$\mathrm{H}^{(t+1)} = \mathrm{ReLU}\left(\hat{\mathrm{A}}\, \mathrm{H}^{(t)}\, \mathrm{W}^{(t)}\right)$$

where $\hat{\mathrm{A}} = \tilde{\mathrm{D}}^{-\frac{1}{2}}\, \tilde{\mathrm{A}}\, \tilde{\mathrm{D}}^{-\frac{1}{2}}$, $\tilde{\mathrm{A}} = \mathrm{A} + \mathrm{I}$ and $\tilde{\mathrm{D}}$ is a diagonal matrix such that $\tilde{\mathrm{D}}_{ii} = \sum_{j=1}^{n} \tilde{\mathrm{A}}_{ij}$

[Kipf and Welling, ICLR'17]

Example of Message Passing Layer of GCN (1/2)



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$X = \begin{bmatrix} 2.1 & 12.0 \\ 1.2 & 9.7 \\ -0.5 & 5.8 \\ 3.4 & 15.9 \end{bmatrix}$$

We compute matrices $\tilde{A}$ and $\tilde{D}$:

$$\tilde{A} = A + I = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\tilde{D} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Example of Message Passing Layer of GCN (2/2)

And then matrix $\hat{A}$:

$$\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} = \begin{bmatrix} 0.333 & 0.288 & 0.333 & 0 \\ 0.288 & 0.25 & 0.288 & 0.353 \\ 0.333 & 0.288 & 0.333 & 0 \\ 0 & 0.353 & 0 & 0.5 \end{bmatrix}$$

The parameters of the message passing layer are as follows:

$$W = \begin{bmatrix} 1.064 & 0.211 & -0.557 \\ -1.282 & 0.614 & 0.996 \end{bmatrix} \qquad b = \begin{bmatrix} -1.177 & -0.540 & 1.331 \end{bmatrix}$$

The representations of the first message passing layer are computed as follows:

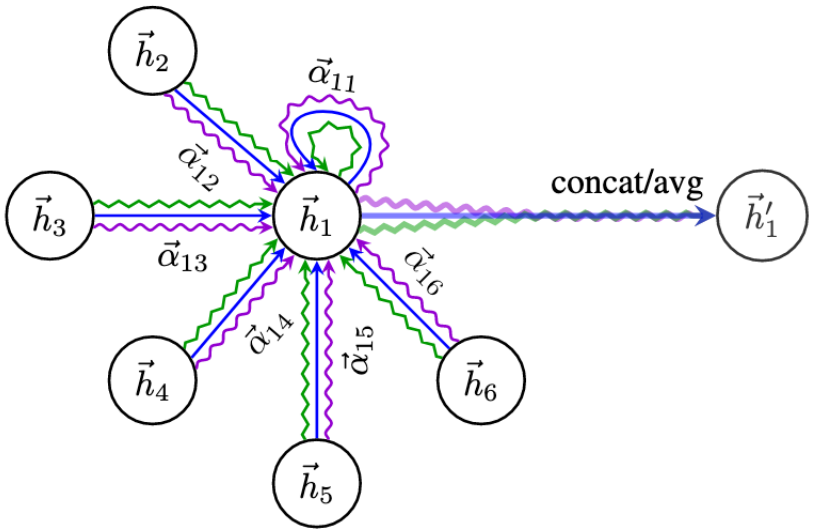
$$H = \mathrm{ReLU}\Big(\hat{A}(XW + b)\Big) = \begin{bmatrix} 0 & 5.024 & 9.466 \\ 0 & 7.859 & 13.588 \\ 0 & 5.024 & 9.466 \\ 0 & 6.971 & 11.281 \end{bmatrix}$$

Graph Attention Network (GAT)

- **Idea**: Messages from some neighbors may be more important than messages from others
- GAT applies self-attention on the nodes
- For nodes $v_j \in \mathcal{N}(v_i)$, computes attention coefficients that indicate the importance of node v_j's features to node v_i:

$$\alpha_{ij}^{(t)} = \frac{\exp\Big(\text{LeakyReLU}\big(\mathsf{a}^\top [\mathsf{W}^{(t)}\mathsf{h}_i^{(t)} || \mathsf{W}^{(t)}\mathsf{h}_j^{(t)}]\big)\Big)}{\sum_{k\in \mathsf{N}_i} \exp\Big(\text{LeakyReLU}\big(\mathsf{a}^\top [\mathsf{W}^{(t)}\mathsf{h}_i^{(t)} || \mathsf{W}^{(t)}\mathsf{h}_k^{(t)}]\big)\Big)}$$

where $[\cdot||\cdot]$ denotes concatenation of two vectors and a is a trainable vector



Graph Attention Network (GAT)

Then the representations of the nodes are updated as follows:

$$\mathsf{h}_i^{(t+1)} = \sigma\Big(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{(t)} \mathsf{W}^{(t)} \mathsf{h}_j^{(t)}\Big)$$

In matrix form, the above is equivalent to:

$$\mathsf{H}^{(t+1)} = \sigma\Big(\big(\mathsf{A} \odot \mathsf{T}^{(t)}\big)\mathsf{H}^{(t)}\mathsf{W}^{(t)}\Big)$$

where $\odot$ denotes elementwise product and is matrix such that $\mathsf{T}_{ij}^{(t)} = \alpha_{ij}^{(t)}$

More than one attention mechanisms can be employed by concatenating/averaging their respective node representations, e.g., for averaging:

$$\mathsf{h}_i^{(t+1)} = \sigma\Big(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} [\alpha_k^{(t)}]_{ij} \mathsf{W}_k^{(t)} \mathsf{h}_j^{(t)}\Big)$$

where $[\alpha_k^{(t)}]_{ij}$ are the attention coefficients computed by the k^{th} attention mechanism, and $\mathsf{W}_k^{(t)}$ is the corresponding weight matrix

[Veličković et al., ICLR'18]

GraphSAGE

The GraphSAGE model can deal with very large graphs
$\hookrightarrow$ the model does not take into account all neighbors of a node, but uniformly samples a fixed-size set of neighbors

Let $\mathcal{N}^k(v)$ be a uniformly drawn subset (of size k) from the set $\mathcal{N}(v)$
The message passing scheme of GraphSAGE is defined as follows:

$$\mathsf{m}_v^{(t)} = \mathrm{AGGREGATE}^{(t)}\left(\left\{\mathsf{h}_u^{(t)} \middle| u \in \mathcal{N}^k(v)\right\}\right)$$

$$\mathsf{h}_v^{(t+1)} = \sigma\left(\mathsf{W}^{(t)}\left[\mathsf{h}_v^{(t)} || \mathsf{m}_v^{(t)}\right]\right)$$

$$\mathsf{h}_v^{(t+1)} = \frac{\mathsf{h}_v^{(t+1)}}{\left|\left|\mathsf{h}_v^{(t+1)}\right|\right|_2}$$

The model draws different uniform samples at each iteration

[Hamilton et al., NIPS'17]

GraphSAGE

The model uses one of the following trainable aggregation functions:

1. **Mean aggregator**: the mean operator computes the elementwise mean of the representations of the neighbors and the node itself (the concatenation step, i.e., second Equation of previous slide is skipped):

$$\mathrm{h}_v^{(t+1)} = \sigma\left(\mathrm{W}^{(t)} \frac{\mathrm{h}_v^{(t)} + \sum_{u \in \mathcal{N}^k(v)} \mathrm{h}_u^{(t)}}{d(v)+1}\right)$$

   where $d(v)$ is the degree of node v

2. **LSTM aggregator**: the representations of the neighbors are passed on to an LSTM architecture
   ⚠ LSTMs are not permutation invariant

3. **Pooling aggregator**: an elementwise max-pooling operation is applied to aggregate information across the neighbor set:

$$\mathrm{AGGREGATE}_{\mathrm{pool}}^{(t)} = \max\left(\left\{\sigma\left(\mathrm{W}_{\mathrm{pool}}^{(t)} \mathrm{h}_u^{(t)}\right) \middle| u \in \mathcal{N}^k(v)\right\}\right)$$

   where max denotes the elementwise max operator

Skip Connections

Idea: Instead of using only the final node representations $h_v^{(T)}$ (i.e., obtained after T message passing steps), can also use the representations of the earlier message passing layers $h_v^{(1)}, \ldots, h_v^{(T-1)}$

Multi-hop information

- As one iterates, vertex representations capture more and more global information
- However, retaining more local, intermediary information might be useful too.
- Thus, we concatenate the representations produced at the different steps, finally obtaining $h_v = [h_v^{(1)} || h_v^{(2)} || \ldots || h_v^{(T)}]$

[Xu et al., ICML'18]

Outline

Graph Autoencoders (GAE)

One of the main problems in representation learning for graphs is the following: How can we learn node embedding representations in an unsupervised fashion?

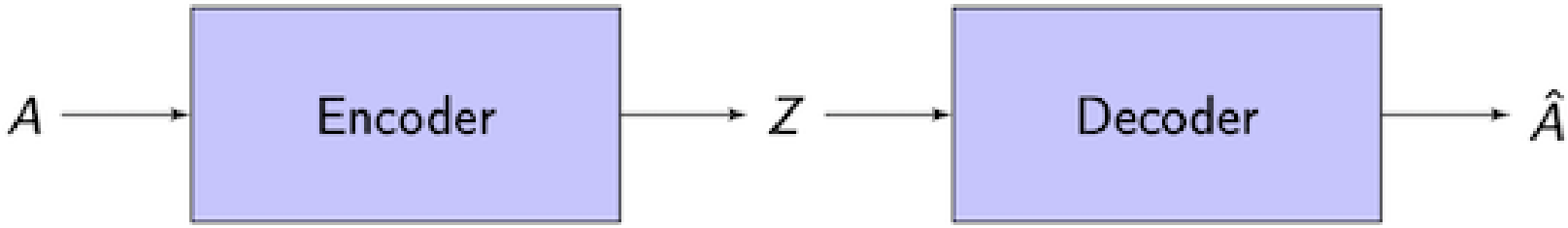
- DeepWalk
- node2vec

⋮

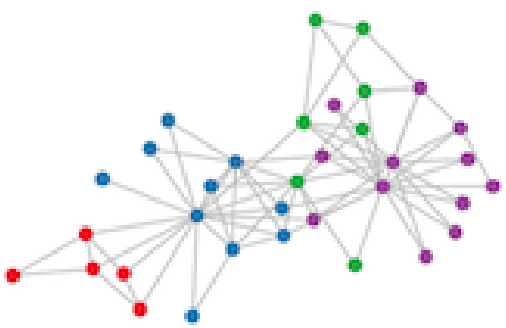
In the last few years: several attempts to generalize autoencoders to graphs:

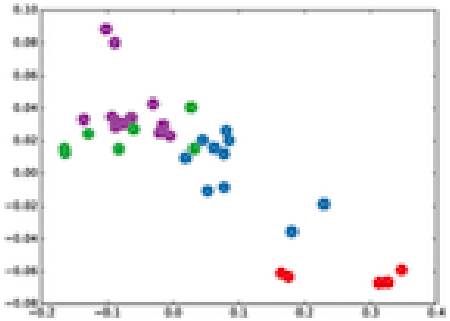
- input: $n \times n$ adjacency matrix $\mathbf{A}$ and (potentially) an $n \times d$ node features matrix $\mathbf{X}$, stacking-up d-dimensional vectors associated to each node
- objective: derive an $n \times d$ latent representation matrix $\mathbf{Z}$ (*encoding step*) from which we can reconstruct (*decoding step*) $\mathbf{A}$

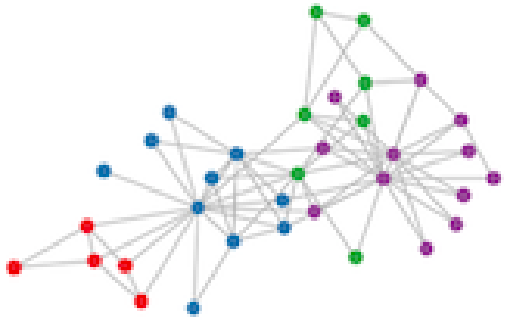
Graph Autoencoders (GAE)



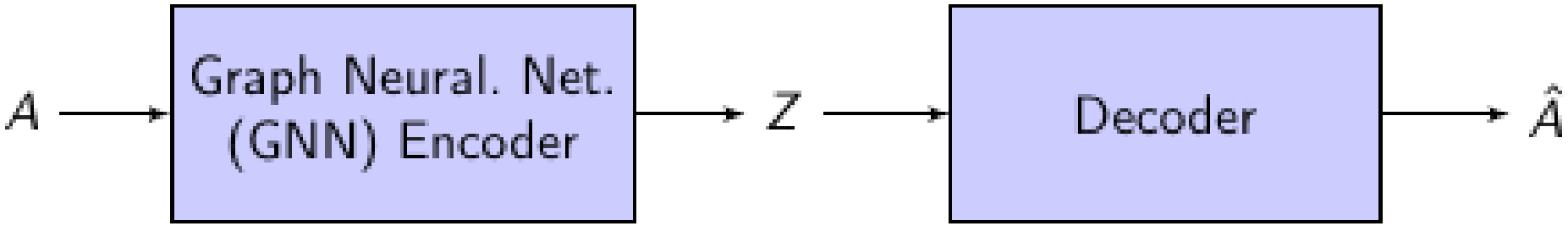
$$A = \begin{pmatrix} 0 & 1 & 0 & \dots & 1 \\ 1 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 0.523 & -1.012 \\ 2.127 & 0.316 \\ 0.912 & 0.127 \\ \dots & \dots \\ -1.210 & 0.026 \end{pmatrix} \quad \hat{A} = \begin{pmatrix} 0 & 1 & 0 & \dots & 1 \\ 1 & 0 & 1 & \dots & 1 \\ 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 0 & \dots & 0 \end{pmatrix}$$







Graph Autoencoders (GAE)



Encoder: usually a **Graph Neural Network**, e.g.:

- Graph Convolutional Network (GCN)
- Graph Attention Network (GAT)
- GraphSAGE

⋮

Most graph autoencoders rely on **multi-layer GCN** encoders

Graph Autoencoders (GAE)

Graph AE:

1. **encoder:** $\boxed{Z = \text{GNN}(A, X)}$
2. **decoder:** $\boxed{\hat{A} = \sigma(ZZ^\top)}$ i.e., for all node pairs (i, j), we have $\hat{A}_{ij} = \sigma(z_i^\top z_j)$

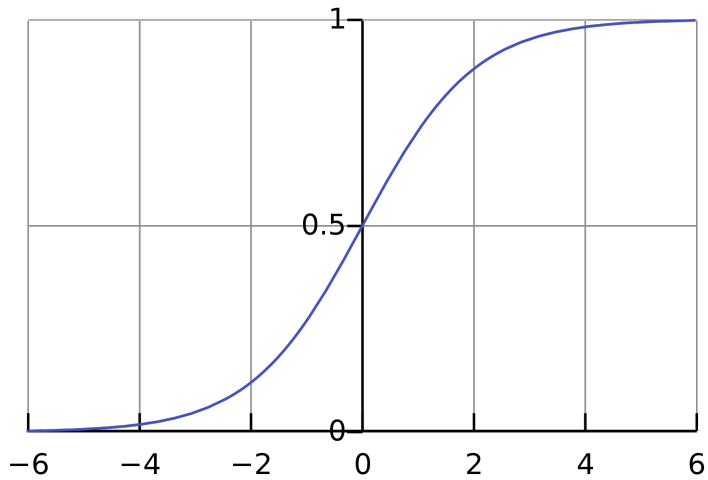


Figure: Sigmoid activation: $\sigma(x) = \frac{1}{1+e^{-x}}$

Reconstruction Loss[1]**:** capturing the similarity between A and $\hat{A}$

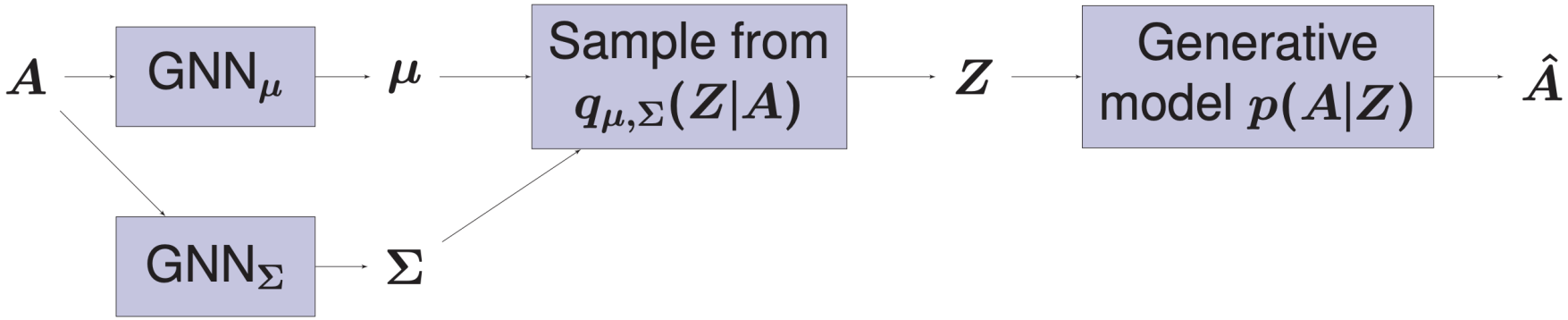
- e.g., **cross-entropy loss**: $-\sum_{i=1}^{n}\sum_{j=1}^{n}(A_{ij}\log(\hat{A}_{ij}) + (1 - A_{ij})\log(1 - \hat{A}_{ij}))$
- or **MSE** loss: $\sum_{i=1}^{n}\sum_{j=1}^{n}(A_{ij} - \hat{A}_{ij})^2$

[1] in losses, we usually reweight positive terms or use negative sampling, if G is sparse

Graph Variational Autoencoders (GVAE)

Also, **Graph VAE**

- extend **Variational Autoencoders (VAE)** to graph structures



Maximizing a lower bound of the model's likelihood (**ELBO**):

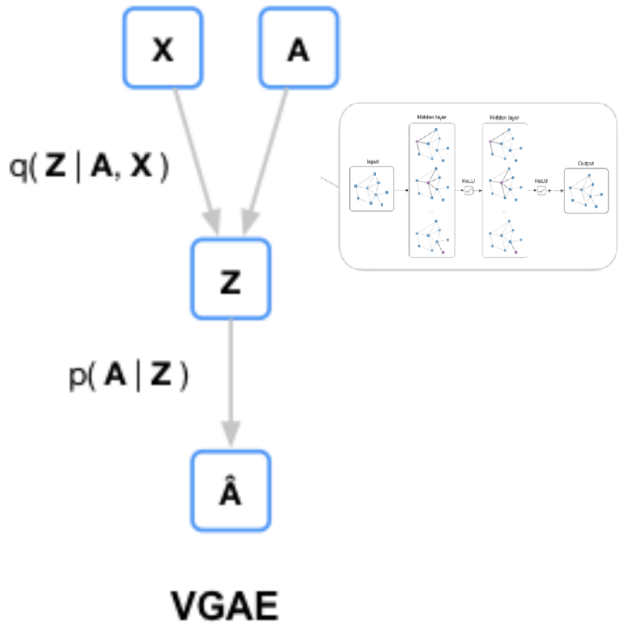
$$\mathcal{L} = \mathbb{E}_{q(\text{Z}|\text{A})}\Big[\log p(\text{A}|\text{Z})\Big] - \mathcal{D}_{KL}(q(\text{Z}|\text{A})||p(\text{Z}))$$

[Kipf and Welling, Bayesian Deep Learning Workshop'16]

Graph Variational Autoencoders (GVAE)

Encoder: $q(\mathrm{Z}|\mathrm{X},\mathrm{A}) = \prod_{i=1}^{n} q(\mathrm{z}_i|\mathrm{X},\mathrm{A})$ where $q(\mathrm{z}_i|\mathrm{X},\mathrm{A}) = \mathcal{N}(\mathrm{z}_i|\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2))$

Gaussian parameters learned by 2 GNNs:
$\boldsymbol{\mu} = \mathrm{GNN}_\mu(\mathrm{X},\mathrm{A})$ and $\log \boldsymbol{\sigma} = \mathrm{GNN}_\sigma(\mathrm{X},\mathrm{A})$



Decoder: $p(\mathrm{A}|\mathrm{Z}) = \prod_{i=1}^{n}\prod_{j=1}^{n} p(\mathrm{A}_{ij}|\mathrm{z}_i,\mathrm{z}_j)$
where $p(\mathrm{A}_{ij}=1|\mathrm{z}_i,\mathrm{z}_j) = \sigma(\mathrm{z}_i^\top \mathrm{z}_j)$

Maximizing ELBO:[2] $\boxed{\mathcal{L} = \mathbb{E}_{q(\mathrm{Z}|\mathrm{X},\mathrm{A})}\Big[\log p(\mathrm{A}|\mathrm{Z})\Big] - \mathcal{D}_{KL}(q(\mathrm{Z}|\mathrm{X},\mathrm{A})||p(\mathrm{Z}))}$

Performing full-batch gradient descent, using the *re-parameterization trick*, and choosing a Gaussian prior $p(\mathrm{Z}) = \prod_i p(\mathrm{z}_i) = \prod_i \mathcal{N}(\mathrm{z}_i|0,\mathrm{I})$.

[2] Evidence Lower Bound

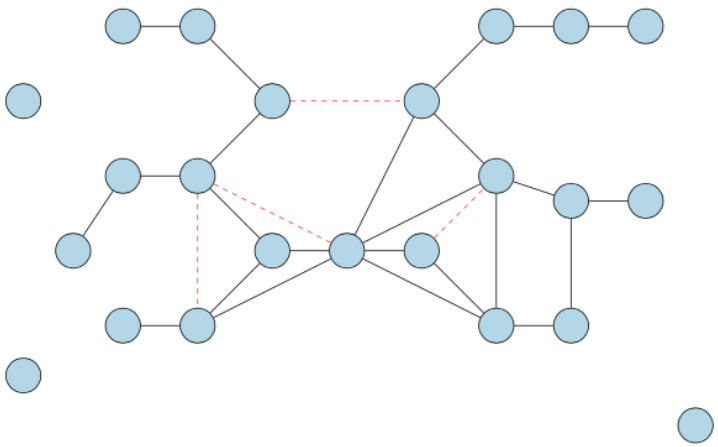
Applications of Graph Autoencoders

The embedding spaces learned via Graph AE and VAE led to many promising applications during the past few years:

- link prediction
- node clustering
- recommendation

$\vdots$

Recall: Link Prediction

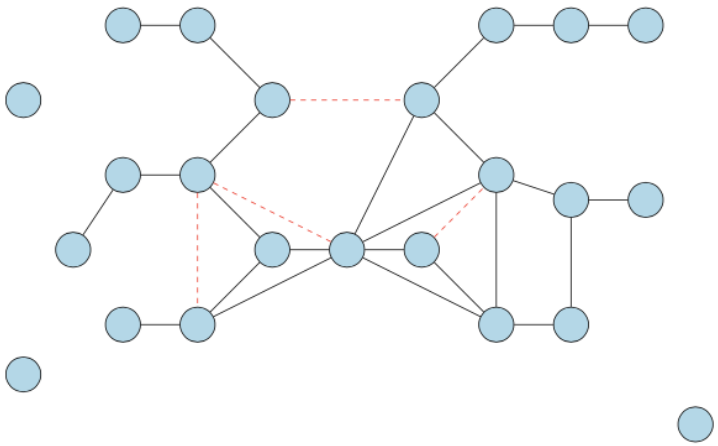


Test set:

- missing edges
- unconnected pairs of nodes

Pair of nodes	**Are nodes connected in ground-truth G?**
(v_1, v_2)	1
(v_3, v_4)	1
(v_5, v_6)	1
...	...
(v_7, v_8)	0
(v_9, v_{10})	0
(v_{11}, v_{12})	0

Recall: Link Prediction



Test set:

- missing edges
- unconnected pairs of nodes

Pair of nodes	**Are nodes connected in ground-truth G?**
(v_1, v_2)	1
(v_3, v_4)	1
(v_5, v_6)	1
...	...
(v_7, v_8)	0
(v_9, v_{10})	0
(v_{11}, v_{12})	0

$\longrightarrow$ **binary classification task**, identify missing edges from incomplete train graph

Link Prediction with Graph Autoencoders

Method	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01	84.2 ± 0.02	87.8 ± 0.01
DW [6]	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01	84.4 ± 0.00	84.1 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01	78.7 ± 0.02	84.1 ± 0.02	82.2 ± 0.01	87.4 ± 0.00
VGAE*	84.0 ± 0.02	87.7 ± 0.01	78.9 ± 0.03	84.1 ± 0.02	82.7 ± 0.01	87.5 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	$\mathbf{96.4} \pm 0.00$	$\mathbf{96.5} \pm 0.00$
VGAE	$\mathbf{91.4} \pm 0.01$	$\mathbf{92.6} \pm 0.01$	$\mathbf{90.8} \pm 0.02$	$\mathbf{92.0} \pm 0.02$	94.4 ± 0.02	94.7 ± 0.02

[Kipf and Welling, Bayesian Deep Learning Workshop'16]

Node Embedding - Cora Graph

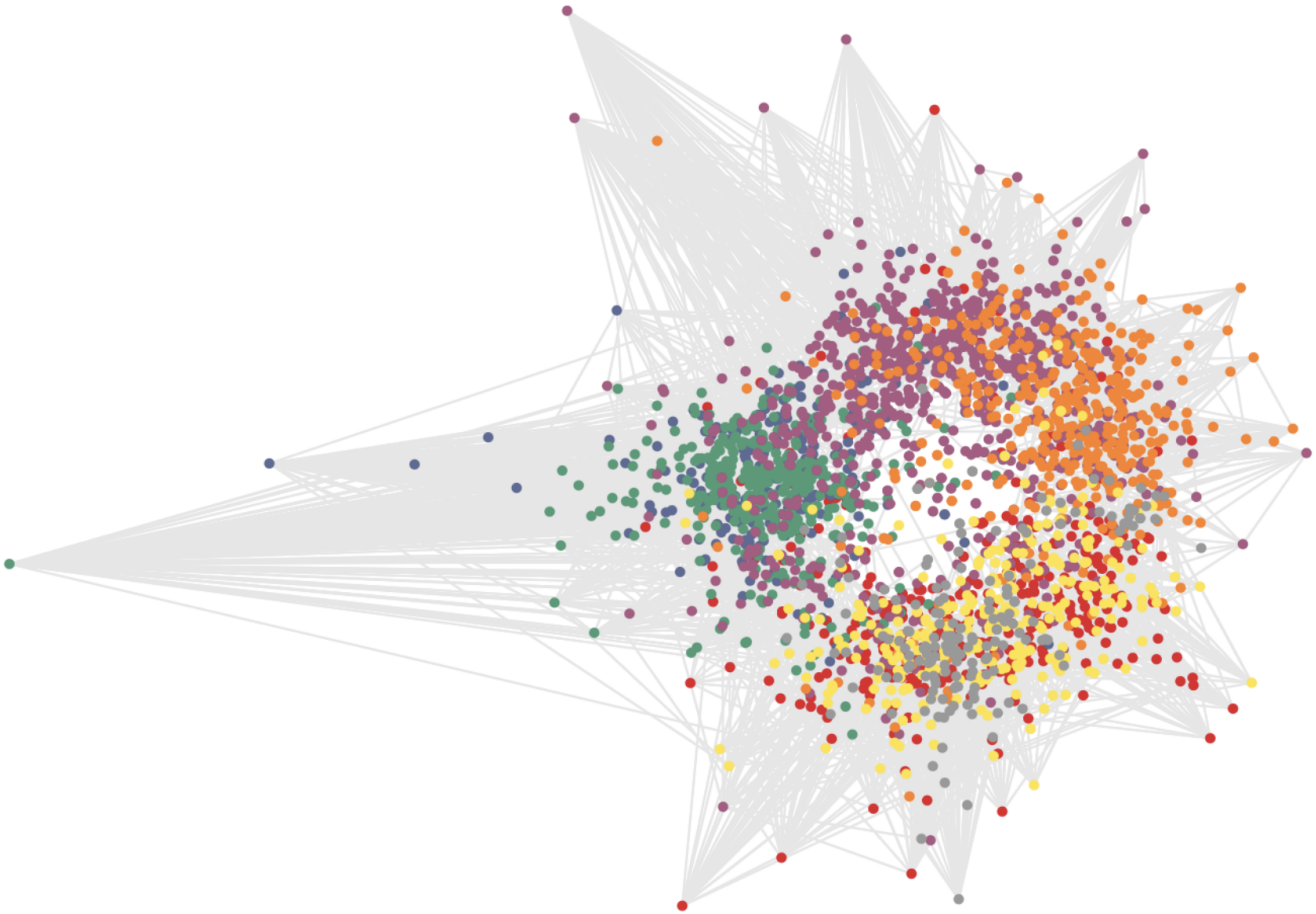


Figure: Projection of latent space representations, from Graph VAE model trained on Cora citation network. Colors denote document classes i.e. node labels (not provided during training)

Graph Autoencoders for Recommendation

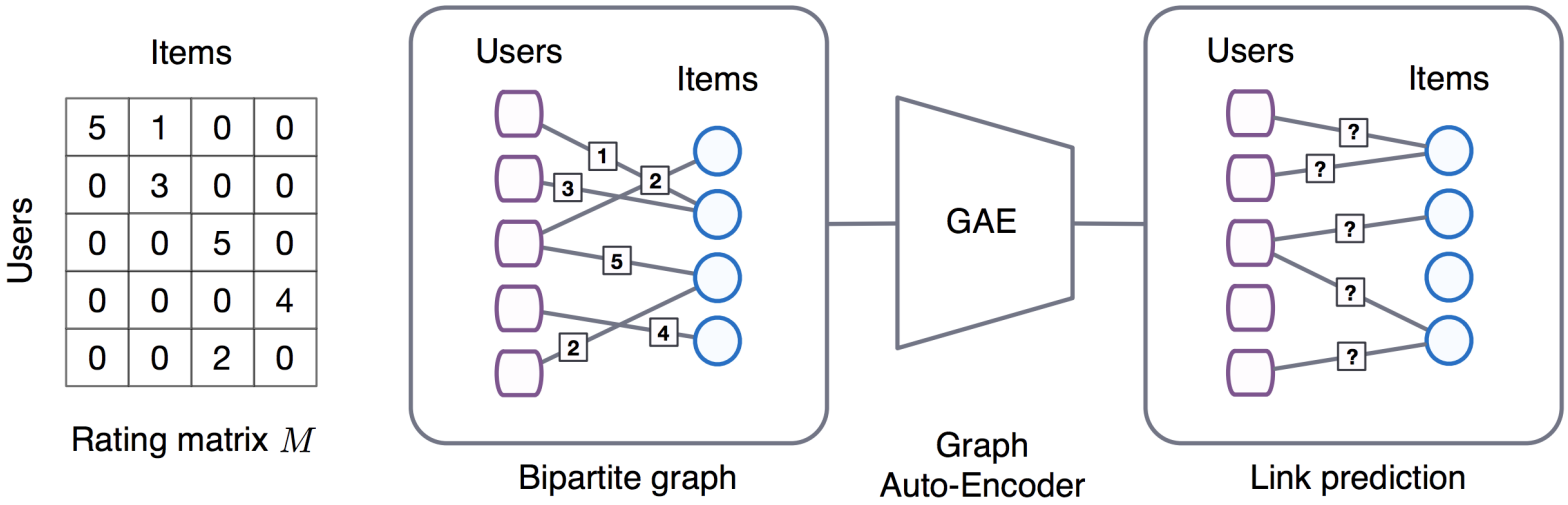
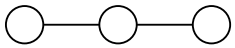


Figure: Using GAE for Matrix Completion and Recommendation

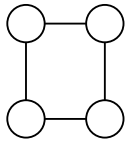
[van den Berg et al., KDD'18 Deep Learning Day]

Outline

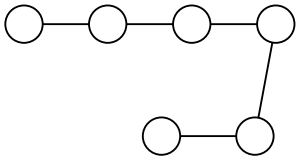
Graph Classification



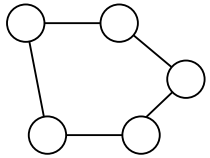
class -1



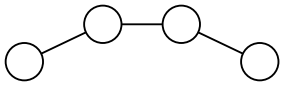
class 1



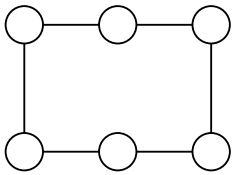
class -1



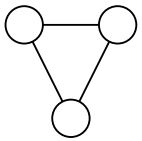
???



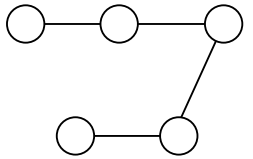
class -1



class 1



class 1



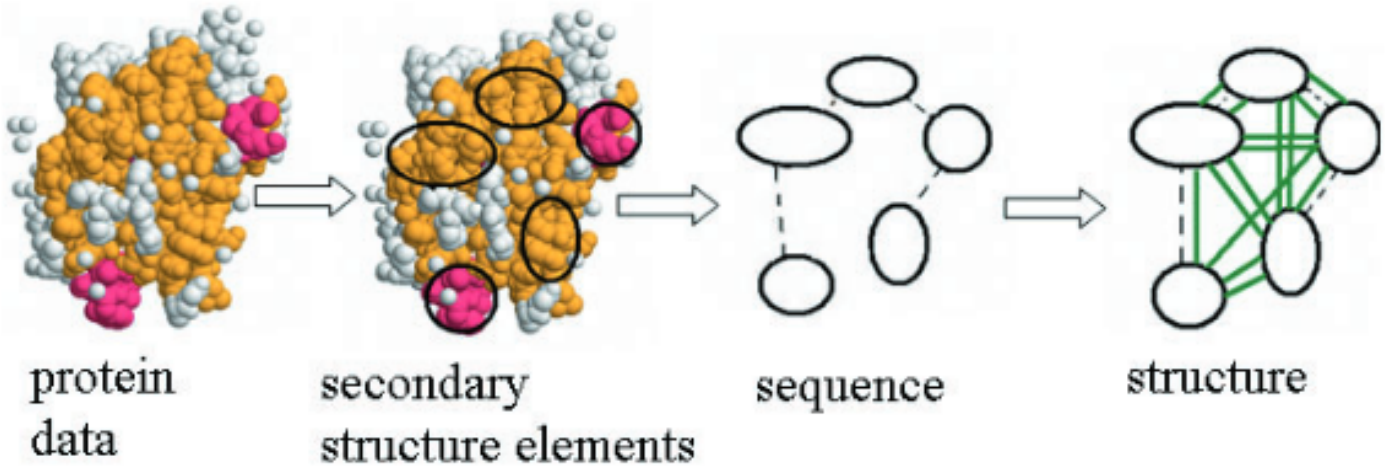
???

- Input data $G \in \mathcal{G}$
- Output $y \in \{-1, 1\}$
- Training set $\mathcal{S} = \{(G_1, y_1), \ldots, (G_n, y_n)\}$
- Goal: estimate a function $f : \mathcal{G} \rightarrow \in \{-1, 1\}$ to predict y from $f(G)$

Motivation - Protein Function Prediction

For each protein, create a graph that contains information about its

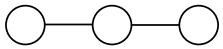
- structure
- sequence
- chemical properties



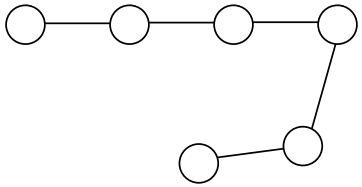
Perform **graph classification** to predict the function of proteins

[Borgwardt et al., Bioinformatics 21]

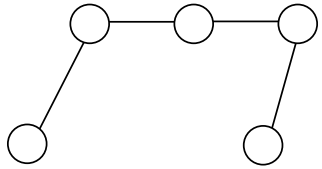
Graph Regression



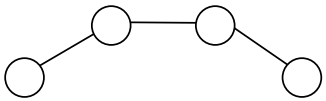
G_1
$y_1 = 3$



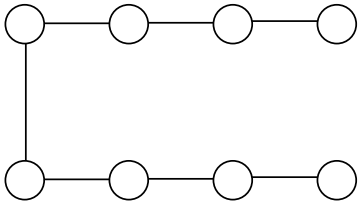
G_2
$y_2 = 6$



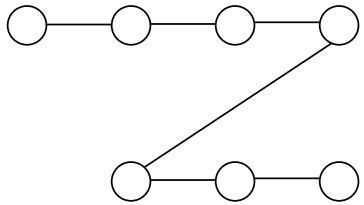
G_5
$y_5 = ???$



G_3
$y_3 = 4$



G_4
$y_4 = 8$

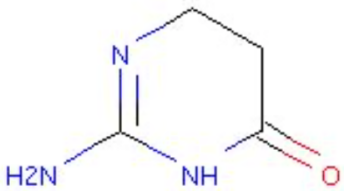


G_6
$y_6 = ???$

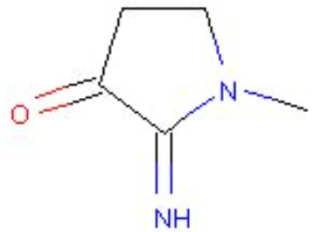
- Input data $G \in \mathcal{G}$
- Output $y \in \mathbb{R}$
- Training set $\mathcal{S} = \{(G_1, y_1), \ldots, (G_n, y_n)\}$
- Goal: estimate a function $f : \mathcal{G} \to \mathbb{R}$ to predict y from $f(G)$

Motivation - Molecular Property Prediction

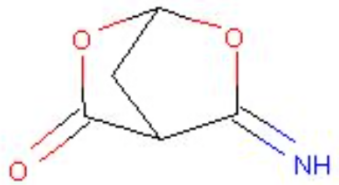
12 targets corresponding to molecular properties: ['mu', 'alpha', 'HOMO', 'LUMO', 'gap', 'R2', 'ZPVE', 'U0', 'U', 'H', 'G', 'Cv']



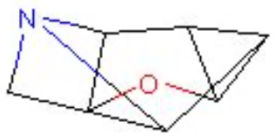
SMILES: NC1=NCCC(=O)N1
Targets: [2.54 64.1 -0.236 -2.79e-03 2.34e-01 900.7 0.12 -396.0 -396.0 -396.0 -396.0 26.9]



SMILES: CN1CCC(=O)C1=N
Targets: [4.218 68.69 -0.224 -0.056 0.168 914.65 0.131 -379.959 -379.951 -379.95 -379.992 27.934]

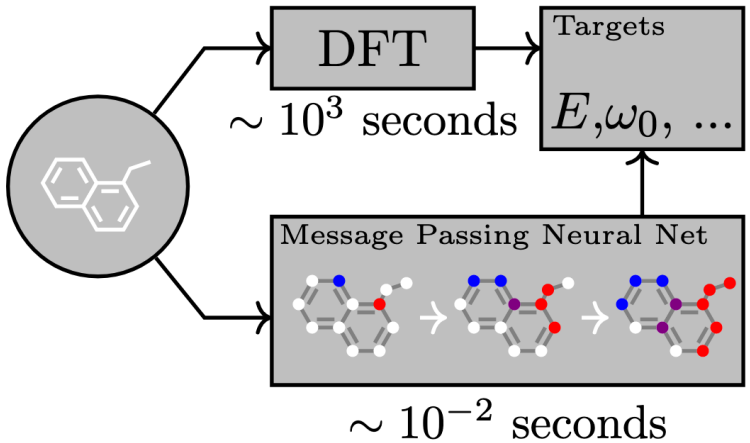


SMILES: N=C1OC2CC1C(=O)O2
Targets: [4.274 61.94 -0.282 -0.026 0.256 887.402 0.104 -473.876 -473.87 -473.869 -473.907 24.823]



SMILES: C1N2C3C4C5OC13C2C5
Targets: [? ? ? ? ? ? ? ? ? ? ? ?]

Perform **graph regression** to predict the values of the properties



[Gilmer et al., ICML'17]

Outline

Message Passing Neural Networks for Learning Graph Representations

Consist of a series of message passing layers followed by a readout function

Step 1: The message passing phase runs for T time steps and updates the representation of each vertex h_v^t based on its previous representation and the representations of its neighbors:

$$\mathsf{m}_v^{(t+1)} = \mathrm{AGGREGATE}\left(\left\{\mathsf{h}_u^{(t)} \middle| u \in \mathcal{N}(v)\right\}\right)$$

$$\mathsf{h}_v^{(t+1)} = \mathrm{COMBINE}\left(\mathsf{h}_v^{(t)}, \mathsf{m}_v^{(t+1)}\right)$$

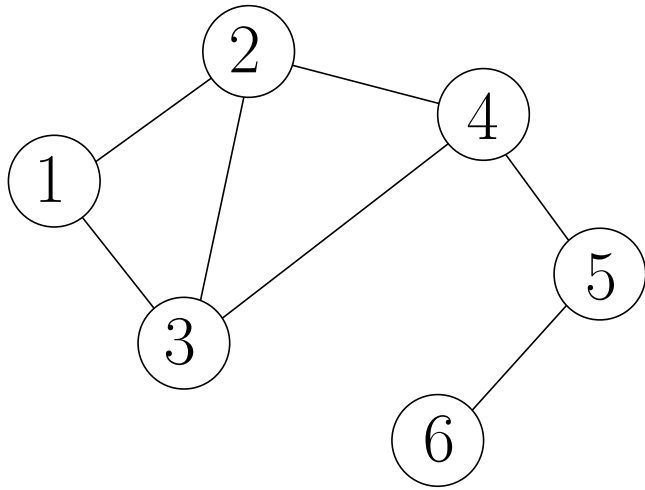
where $\mathcal{N}(v)$ is the set of neighbors of v, and $\mathrm{AGGREGATE}$ and $\mathrm{COMBINE}$ are message functions and vertex update functions respectively

Step 2: The readout step computes a feature vector for the whole graph using some readout function R:

$$\mathsf{h}_G = \mathrm{READOUT}\left(\left\{\mathsf{h}_v^{(T)} \middle| v \in G\right\}\right)$$

Example of Message Passing Layer

$$h_1^{(t+1)} = f(W_0^{(t)} h_1^{(t)} + W_1^{(t)} h_2^{(t)} + W_1^{(t)} h_3^{(t)})$$
$$h_2^{(t+1)} = f(W_0^{(t)} h_2^{(t)} + W_1^{(t)} h_1^{(t)} + W_1^{(t)} h_3^{(t)} + W_1^{(t)} h_4^{(t)})$$
$$h_3^{(t+1)} = f(W_0^{(t)} h_3^{(t)} + W_1^{(t)} h_1^{(t)} + W_1^{(t)} h_2^{(t)} + W_1^{(t)} h_4^{(t)})$$
$$h_4^{(t+1)} = f(W_0^{(t)} h_4^{(t)} + W_1^{(t)} h_2^{(t)} + W_1^{(t)} h_3^{(t)} + W_1^{(t)} h_5^{(t)})$$
$$h_5^{(t+1)} = f(W_0^{(t)} h_5^{(t)} + W_1^{(t)} h_4^{(t)} + W_1^{(t)} h_6^{(t)})$$
$$h_6^{(t+1)} = f(W_0^{(t)} h_6^{(t)} + W_1^{(t)} h_5^{(t)})$$

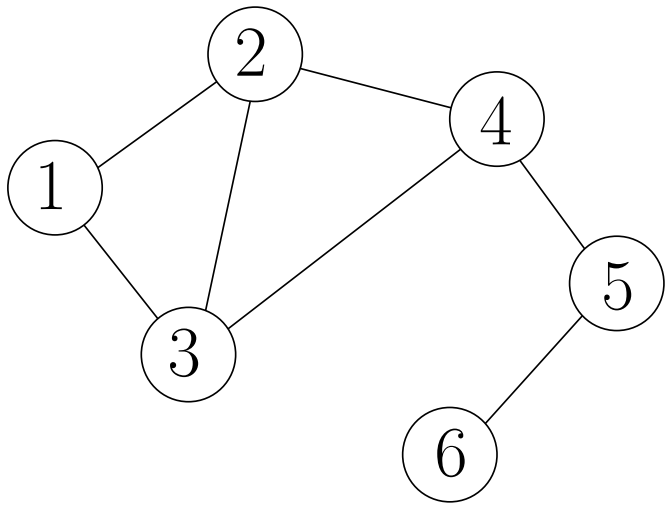


Remark: Biases are omitted for clarity

Readout Step Example

Output of message passing phase:

$$\left\{ \mathsf{h}_1^{(T)}, \mathsf{h}_2^{(T)}, \mathsf{h}_3^{(T)}, \mathsf{h}_4^{(T)}, \mathsf{h}_5^{(T)}, \mathsf{h}_6^{(T)} \right\}$$



Graph representation:

$$\mathsf{h}_G = \frac{1}{6}\left(\mathsf{h}_1^{(T)} + \mathsf{h}_2^{(T)} + \mathsf{h}_3^{(T)} + \mathsf{h}_4^{(T)} + \mathsf{h}_5^{(T)} + \mathsf{h}_6^{(T)} \right)$$

How Can we Build Message Passing Neural Networks for Learning Graph Representations?

1. Take a message passing neural network that can produce node representations
2. Add a readout function to the model. Simple and popular functions.
   - sum aggerator: computes the sum of the representations of the nodes of the graph

$$\mathbf{h}_G = \sum_{v \in V} \mathbf{h}_v^{(T)}$$

   - mean aggerator: computes the sum of the representations of the nodes of the graph

$$\mathbf{h}_G = \frac{1}{n} \sum_{v \in V} \mathbf{h}_v^{(T)}$$

   - max aggerator: an elementwise max-pooling operation is applied to the representations of the nodes of the graph

$$\mathbf{h}_G = \max\left(\left\{\mathbf{h}_v^{(T)}\right) \middle| v \in V\right\}\right)$$

   where max denotes the elementwise max operator

Example of Simple Readout Functions

Suppose we have a graph consisting of 3 nodes and we have that:
$\mathrm{h}_1^{(T)} = \begin{bmatrix} 1.2 & 1.4 & -1.0 \end{bmatrix}$ $\mathrm{h}_2^{(T)} = \begin{bmatrix} -2.4 & -0.6 & 1.3 \end{bmatrix}$
$\mathrm{h}_3^{(T)} = \begin{bmatrix} 1.5 & 1.3 & -0.9 \end{bmatrix}$

Then, we can produce graph representations as follows:

- sum aggerator:

$$\mathrm{h}_G = \mathrm{h}_1^{(T)} + \mathrm{h}_2^{(T)} + \mathrm{h}_3^{(T)} = \begin{bmatrix} 0.3 & 2.1 & -0.6 \end{bmatrix}$$

- mean aggerator:

$$\mathrm{h}_G = \frac{1}{3}\left(\mathrm{h}_1^{(T)} + \mathrm{h}_2^{(T)} + \mathrm{h}_3^{(T)}\right) = \begin{bmatrix} 0.1 & 0.7 & -0.2 \end{bmatrix}$$

- max aggerator:

$$\mathrm{h}_G = \max\left(\left\{\mathrm{h}_1^{(T)}, \mathrm{h}_2^{(T)}, \mathrm{h}_3^{(T)}\right\}\right) = \begin{bmatrix} 1.5 & 1.4 & 1.3 \end{bmatrix}$$

Convolutional Networks for Learning Molecular Fingerprints

Step 1: The network updates the states of the nodes as follows:

$$\mathrm{m}_v^{(t+1)} = \mathrm{h}_v^{(t)} + \sum_{u \in \mathcal{N}(v)} \mathrm{h}_u^{(t)}$$

$$\mathrm{h}_v^{(t+1)} = \sigma\left(\mathrm{E}_{d(v)}^{(t)} \mathrm{m}_v^{(t+1)}\right)$$

where $d(v)$ is degree of vertex v and $\mathrm{E}_{d(v)}^{(t)}$ a learned matrix for each time step t and vertex degree $d(v)$

Step 2: The network computes the graph representation as:

$$\mathrm{h}_G = \sum_{t=0}^{T} \sum_{v \in V} \mathrm{softmax}(\mathrm{W}^{(t)} \mathrm{h}_v^{(t)})$$

The output h_G is then fed to a fully-connected neural network

[Duvenaud et al, NIPS'15]

Deep Graph Convolutional Neural Network (DGCNN)

Step 1: Aggregates node information in local neighborhoods to extract local substructure information:

$$\mathrm{H}^{(t+1)} = f\left(\tilde{\mathrm{D}}^{-1}\tilde{\mathrm{A}}\mathrm{H}^{(t)}\mathrm{W}^{(t)}\right)$$

where $\tilde{\mathrm{A}} = \mathrm{A} + \mathrm{I}$, D is a diagonal matrix such that $\tilde{\mathrm{D}}_{ii} = \sum_j \tilde{\mathrm{A}}_{ij}$, and f is a nonlinear activation function

After T iterations, the model concatenates the outputs $\mathrm{H}^{(t)}$, for $t = 1, \ldots, T$ horizontally to form a concatenated output:

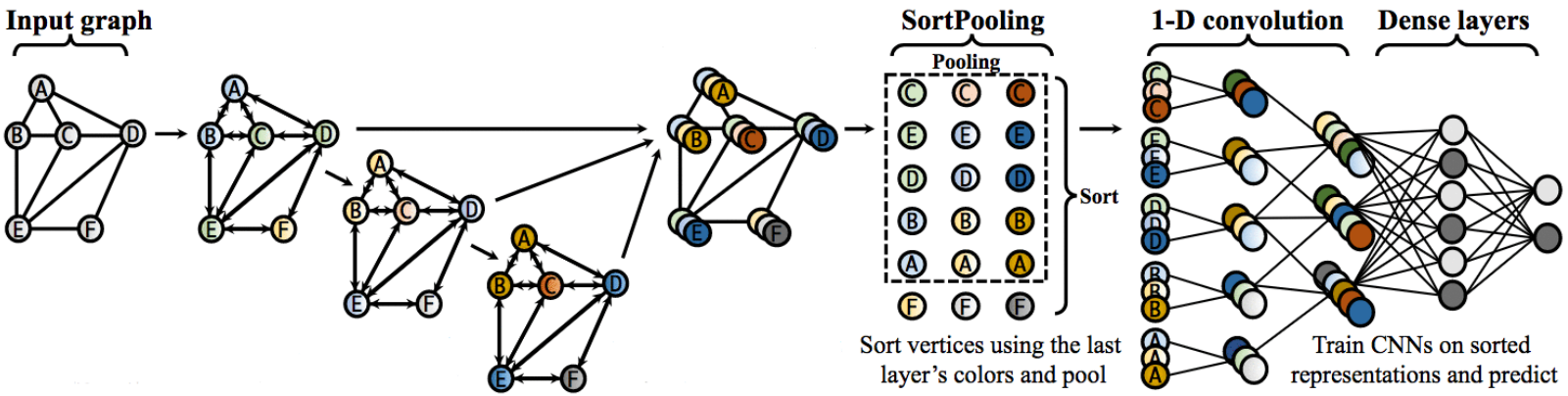
$$\mathrm{H} = \left[\mathrm{H}^{(1)}||\mathrm{H}^{(2)}||\ldots||\mathrm{H}^{(T)}\right]$$

[Zhang et al, AAAI'18]

Deep Graph Convolutional Neural Network (DGCNN)

Step 2: Employs the so-called SortPooling layer:

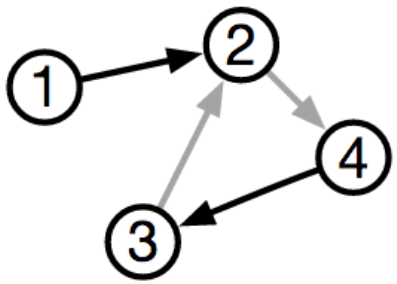
- Sorts the output H of previous step row-wise:
  - vertices are sorted in a descending order based on the last component of H
  - vertices that have the same value in the last component are compared based on the second to last component and so on
- Unifies the sizes of the outputs to handle graphs with different numbers of vertices:
  - Truncates/extends the output tensor in the first dimension from n to k
- Output is then passed to traditional CNN

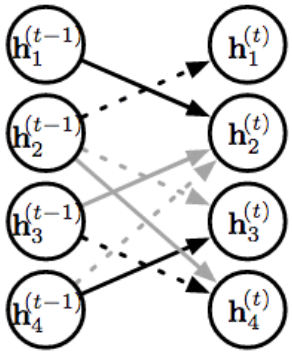


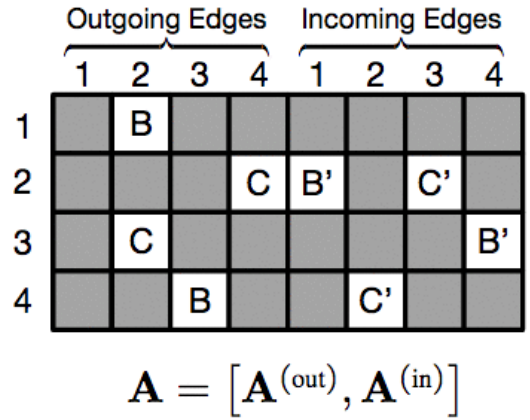
Gated Graph Sequence Neural Network (GG-NN)

Step 1: The model employs the following message passing scheme

- the network transfers information between each vertex and its neighbors: $M^{(t)} = A^{(\text{out})}H^{(t)} + A^{(\text{in})}H^{(t)}$ (assumes directed graphs)







- and then the following GRU-like updates take place:

$$
\begin{aligned}
Z^{(t)} &= \sigma(M^{(t)}W^z + H^{(t)}U^z) \\
R^{(t)} &= \sigma(M^{(t)}W^r + H^{(t)}U^r) \\
\tilde{H}^{(t+1)} &= \tanh\big(M^{(t)}W + (R^t \odot H^{(t)})U\big) \\
H^{(t+1)} &= (1 - Z^{(t)}) \odot H^{(t)} + Z^{(t)} \odot \tilde{H}^{(t+1)}
\end{aligned}
$$

[Li et al, ICLR'16]

Step 2: Generates a a graph level representation vector as follows:

$$h_G = \tanh\left(\sum_{v \in V} \sigma\left(W^i\left([h_v^{(T)}||x_v]\right)\right) \odot \tanh\left(W^j\left([h_v^{(T)}||x_v]\right)\right)\right)$$

where $\sigma\left(W^i\left([h_v^{(T)}||x_v]\right)\right)$ is a soft attention mechanism that decides which nodes are relevant to the current graph-level task, and x_v is the initial feature of node v

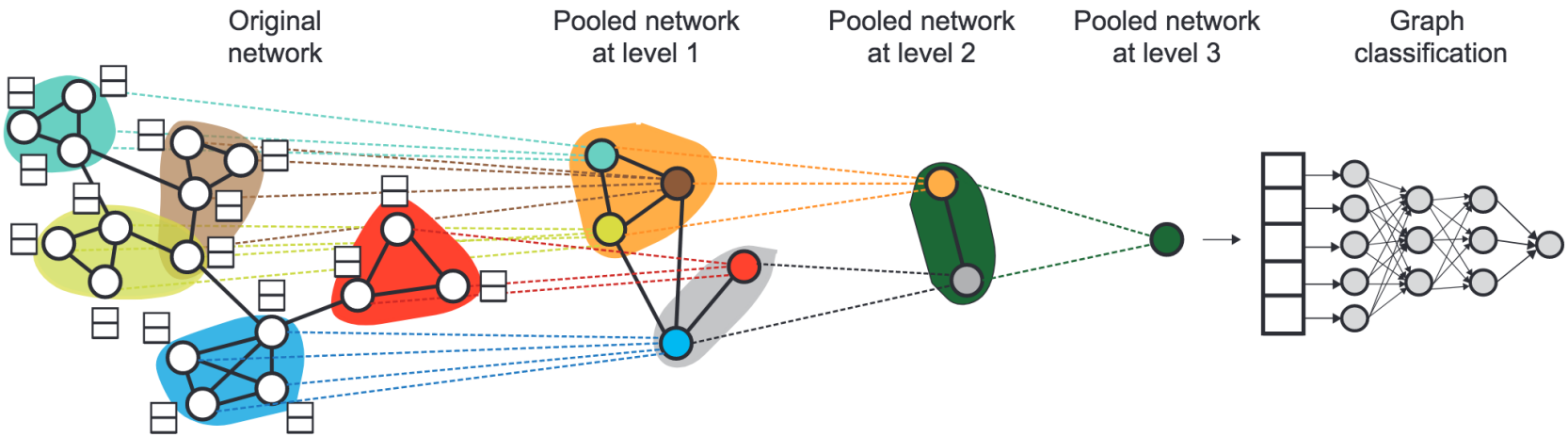
Differentiable Graph Pooling (DiffPool)

Idea: Simple readout functions too flat
↪ Aggregate information in a hierarchical way to capture the entire graph

The DiffPool model

- learns hierarchical pooling analogous to CNNs
- sets of nodes are pooled hierarchically
- soft assignment of nodes to next-level nodes

A different GNN is learned at every level of abstraction



[Ying et al, NIPS'18]

Differentiable Graph Pooling (DiffPool)

A matrix $S^{(t)} \in \mathbb{R}^{n_t \times n_{t+1}}$ is associated with each DiffPool layer

- corresponds to the learned cluster assignment matrix at layer t
- each row corresponds to one of the n_t nodes (or clusters) at layer t and each column to one of the n_{t+1} clusters at the next layer $t+1$
- it provides a soft assignment of each node at layer l to a cluster in the next coarsened layer $t+1$

Each DiffPool layer coarsens the input graph:

$$X^{(t+1)} = S^{(t)^\top} Z^{(t)}$$

$$A^{(t+1)} = S^{(t)^\top} A^{(t)} S^{(t)}$$

where $A^{(t+1)}$ is the coarsened adjacency matrix, and $X^{(t+1)}$ is a matrix of embeddings for each node/cluster

Differentiable Graph Pooling (DiffPool)

- DiffPool generates the assignment and embedding matrices using two separate message passing neural networks
- Both are applied to the input cluster node features $\mathsf{X}^{(t)}$ and coarsened adjacency matrix $\mathsf{A}^{(t)}$

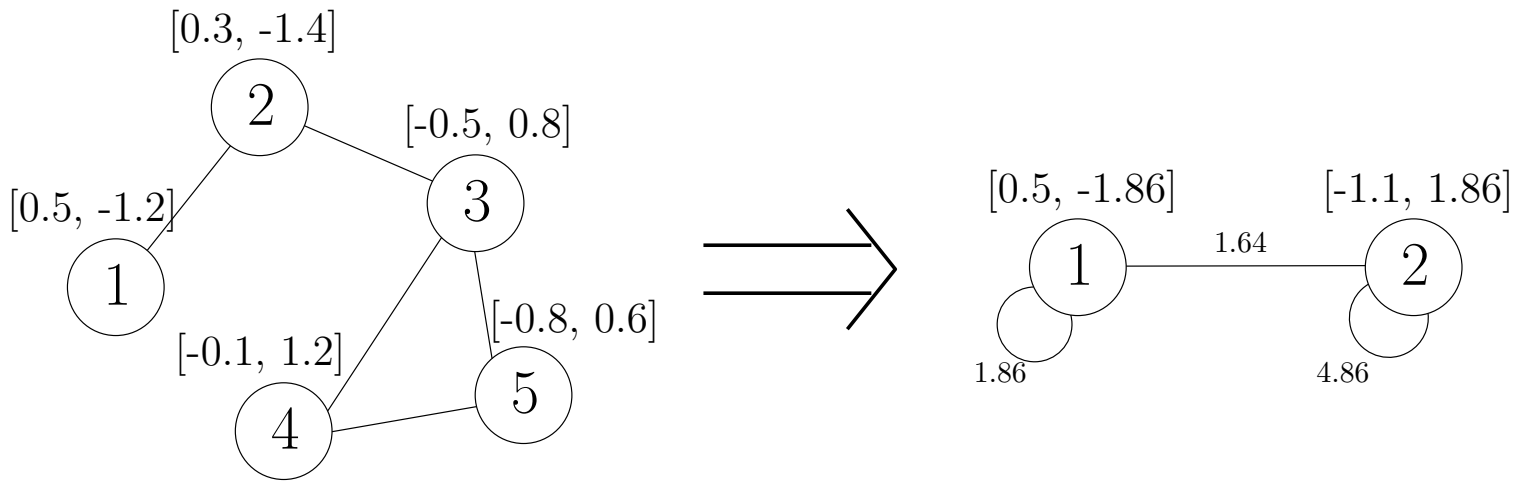
$$\mathsf{Z}^{(t)} = \mathrm{GNN}^{(t)}_{\mathrm{embed}}(\mathsf{A}^{(t)}, \mathsf{X}^{(t)})$$
$$\mathsf{S}^{(t)} = \mathrm{softmax}\big(\mathrm{GNN}^{(t)}_{\mathrm{pool}}(\mathsf{A}^{(t)}, \mathsf{X}^{(t)})\big)$$

  where the softmax function is applied in a row-wise fashion
- $\mathrm{GNN}^{(t)}_{\mathrm{embed}}$ generates new representations for the input nodes
- $\mathrm{GNN}^{(t)}_{\mathrm{pool}}$ generates a probabilistic assignment of the input nodes to n_{t+1} clusters

Example of Coarsening Procedure of DiffPool

$$A^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad Z^{(1)} = \begin{bmatrix} 0.5 & -1.2 \\ 0.3 & -1.4 \\ -0.5 & 0.8 \\ -0.1 & 1.2 \\ -0.8 & 0.6 \end{bmatrix} \quad S^{(1)} = \begin{bmatrix} 0.9 & 0.1 \\ 0.8 & 0.2 \\ 0.2 & 0.8 \\ 0.1 & 0.9 \\ 0.1 & 0.9 \end{bmatrix}$$

$$X^{(2)} = S^{(1)^\top} Z^{(1)} = \begin{bmatrix} 0.5 & -1.86 \\ -1.1 & 1.86 \end{bmatrix} \quad A^{(2)} = S^{(1)^\top} A^{(1)} S^{(1)} = \begin{bmatrix} 1.86 & 1.64 \\ 1.64 & 4.86 \end{bmatrix}$$



Outline

How Powerful Are Message Passing Graph Neural Networks?

- The expressive power of standard message passing graph neural networks has recently started being investigated
- Xu et al. and Morris et al. showed independently that they are **at most** as powerful as the Weisfeiler-Lehman (WL) test of isomorphism in terms of distinguishing between non-isomorphic graphs

Lemma (Xu et al., ICLR'19)

Let G_1 and G_2 be any two non-isomorphic graphs. If a standard graph neural network $A : \mathcal{G} \to \mathbb{R}^d$ maps G_1 and G_2 to different embeddings, the Weisfeiler-Lehman graph isomorphism test also decides G_1 and G_2 are not isomorphic.

- Several well-established models are less powerful than the WL test (e.g., GCN, GraphSAGE, GAT)
- However, compared to Weisfeiler-Lehman kernel:
  - more flexible $\to$ can adapt to the learning task
  - can handle continuous node features

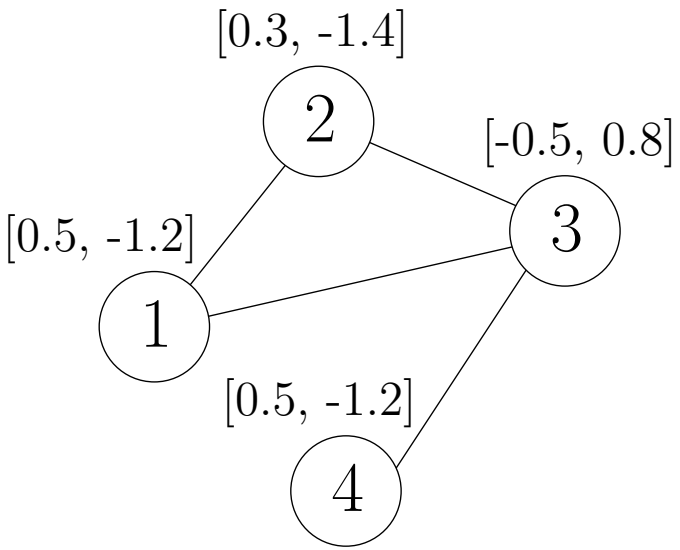
[Xu et al., ICLR'19; Morris et al., AAAI'19]

How Powerful Are Message Passing Graph Neural Networks?

Multiset: a multiset is a generalized concept of a set that allows multiple instances for its elements

When node features are from a countable universe:

- Features of all nodes can be thought of as a multiset
- Neighborhood of each node can be thought of as a multiset
- Node representations at deeper layers are also from a countable universe



The representations of the neighbors of node 3 correspond to a multiset, e.g., $\{[0.5, -1.2], [0.3, -1.4], [0.5, -1.2]\}$ is a multiset

The representations of all the nodes of the graph also correspond to a multiset, e.g., $\{[0.5, -1.2], [0.3, -1.4], [-0.5, 0.8], [0.5, -1.2]\}$ is a multiset

How Powerful Are Message Passing Graph Neural Networks?

The $\mathrm{AGGREGATE}$, $\mathrm{COMBINE}$ and $\mathrm{READOUT}$ functions of a message passing model are injective $\Rightarrow$ The model is as powerful as the WL test

The $\mathrm{AGGREGATE}$ and $\mathrm{READOUT}$ functions operate on multisets of node representations

Question: Are commonly-employed $\mathrm{AGGREGATE}$ and $\mathrm{READOUT}$ functions injective or not?

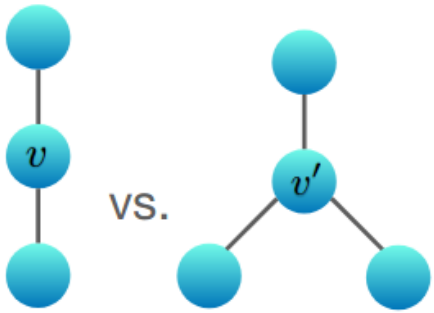
How Powerful Are Message Passing Graph Neural Networks?

The AGGREGATE, COMBINE and READOUT functions of a message passing model are injective $\Rightarrow$ The model is as powerful as the WL test

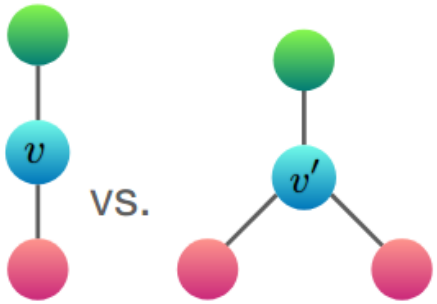
The AGGREGATE and READOUT functions operate on multisets of node representations

Question: Are commonly-employed AGGREGATE and READOUT functions injective or not?

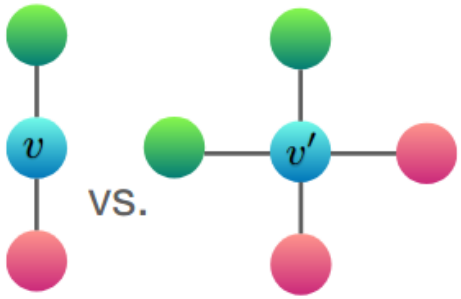
Turns out that mean and max functions are **not** injective!



(a) Mean and Max both fail



(b) Max fails



(c) Mean and Max both fail

On the other hand, sum aggregators can represent injective, in fact, universal functions over multisets

Graph Isomorphism Network (GIN)

GIN is a message passing neural network that

- models injective multiset functions for the neighborhood and node aggregation
- has the same power as the Weisfeiler-Lehman test

Step 1: GIN updates node representations as follows:

$$\mathrm{h}_v^{(t+1)} = \mathrm{MLP}^{(t)}\left((1+\epsilon^{(t)})\mathrm{h}_v^{(t)} + \sum_{u \in \mathcal{N}(v)} \mathrm{h}_u^{(t)}\right)$$

where ϵ is an irrational number

Step 2: Utilizes the following graph-level readout function which uses information from all iterations of the model:

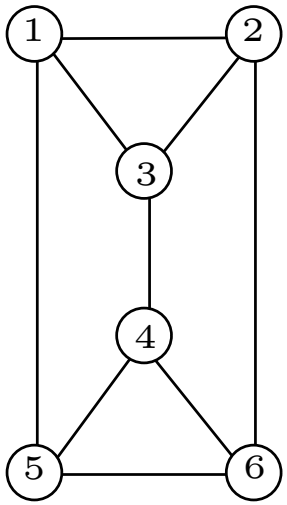
$$\mathrm{h}_G = \left[\sum_{v \in G} \mathrm{h}_v^{(0)} || \ldots || \sum_{v \in G} \mathrm{h}_v^{(T)}\right]$$

[Xu et al., ICLR'19]

Further Limitations of Standard GNNs

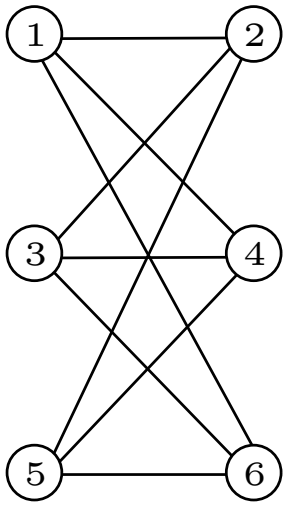
All nodes are annotated with the same feature $h_i^0 = 1$ for all $i \in \{1, \ldots, 6\}$

$$h_1^1 = f(W_0^0 h_1^0 + W_1^0 h_2^0 + W_1^0 h_3^0 + W_1^0 h_5^0) = f(W_0^0 + 3W_1^0)$$
$$h_2^1 = f(W_0^0 h_2^0 + W_1^0 h_1^0 + W_1^0 h_3^0 + W_1^0 h_6^0) = f(W_0^0 + 3W_1^0)$$
$$h_3^1 = f(W_0^0 h_3^0 + W_1^0 h_1^0 + W_1^0 h_2^0 + W_1^0 h_4^0) = f(W_0^0 + 3W_1^0)$$
$$h_4^1 = f(W_0^0 h_4^0 + W_1^0 h_3^0 + W_1^0 h_5^0 + W_1^0 h_6^0) = f(W_0^0 + 3W_1^0)$$
$$h_5^1 = f(W_0^0 h_5^0 + W_1^0 h_1^0 + W_1^0 h_4^0 + W_1^0 h_6^0) = f(W_0^0 + 3W_1^0)$$
$$h_6^1 = f(W_0^0 h_6^0 + W_1^0 h_2^0 + W_1^0 h_4^0 + W_1^0 h_5^0) = f(W_0^0 + 3W_1^0)$$



G_1

$$h_1^1 = f(W_0^0 h_1^0 + W_1^0 h_2^0 + W_1^0 h_4^0 + W_1^0 h_6^0) = f(W_0^0 + 3W_1^0)$$
$$h_2^1 = f(W_0^0 h_2^0 + W_1^0 h_1^0 + W_1^0 h_3^0 + W_1^0 h_5^0) = f(W_0^0 + 3W_1^0)$$
$$h_3^1 = f(W_0^0 h_3^0 + W_1^0 h_2^0 + W_1^0 h_4^0 + W_1^0 h_6^0) = f(W_0^0 + 3W_1^0)$$
$$h_4^1 = f(W_0^0 h_4^0 + W_1^0 h_1^0 + W_1^0 h_3^0 + W_1^0 h_5^0) = f(W_0^0 + 3W_1^0)$$
$$h_5^1 = f(W_0^0 h_5^0 + W_1^0 h_2^0 + W_1^0 h_4^0 + W_1^0 h_6^0) = f(W_0^0 + 3W_1^0)$$
$$h_6^1 = f(W_0^0 h_6^0 + W_1^0 h_1^0 + W_1^0 h_3^0 + W_1^0 h_5^0) = f(W_0^0 + 3W_1^0)$$

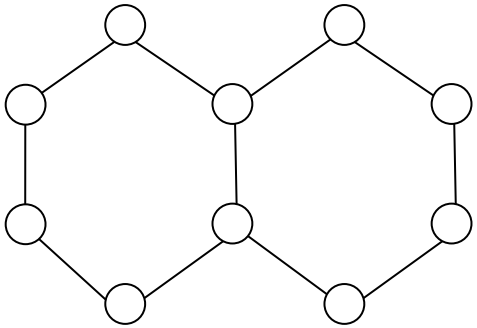


G_2

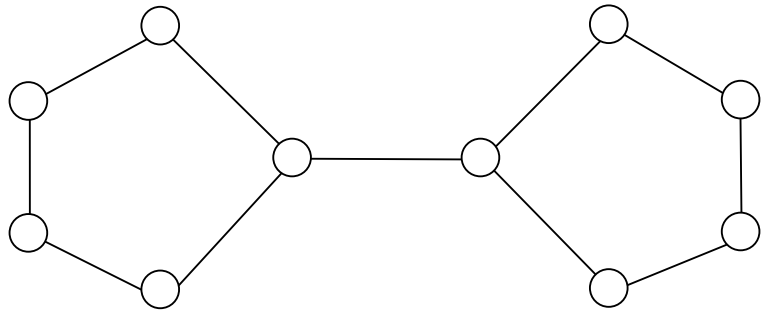
Why Do We Care About the Expressive Power?

- Non-isomorphic that are not distinguished by a model are mapped to the same feature vector!!
- Therefore, there are cases where the model cannot assign different labels to different graphs
- Consider, for example, the following two chemical compounds

Decalin



Bicyclopentyl



- The above two compounds cannot be distinguished by the WL algorithm ☹ GNNs that are not more powerful than WL cannot embed the two compounds into different representations

More Powerful Graph Neural Networks

Question: Are there any models that can distinguish more pairs of non-isomorphic graphs than the WL algorithm?

More Powerful Graph Neural Networks

Question: Are there any models that can distinguish more pairs of non-isomorphic graphs than the WL algorithm?

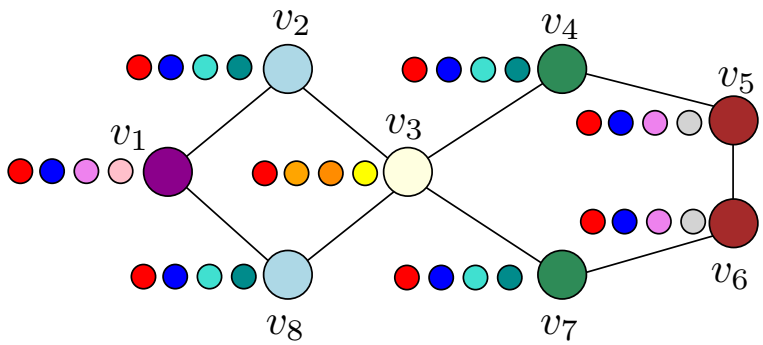
A very active field of research!

There are models that:

- are inspired from high-dimensional variants of the WL algorithm $\rightarrow$ a generalization of the WL which colors tuples from V^k instead of nodes
  - k-GNN **[Morris et al., AAAI'19]**
- extract and process subgraphs
  - k-hop **[Nikolentzos et al., Neural Networks 130]**
  - node-deleted subgraphs **[Cotta et al., NeurIPS'21]**
  - path based subgraphs **[Gaspard et al., ICML'23]**
- consider all possible permutations of nodes
  - RelationalPooling **[Murphy et al., ICML'19]**
  - CLIP **[Dasoulas et al., IJCAI'20]**
- utilize invariant and equivariant linear layers
  - k-order graph networks **[Maron et al., ICLR'19]**

Higher Order Weisfeiler and Leman (WL) Algorithm (1/4)

- The standard Weisfeiler-Leman algorithm (WL) refines colors at the level of single nodes



- We can generalize WL to k-tuples of nodes, thus improving its expressive power
  - higher-order variants can distinguish no less pairs of non-isomorphic graphs than WL **[Cai et al., Combinatorica 12(4); Grohe and Otto, The Journal of Symbolic Logic 80(3)]**
  - for example, for $k = 3$, the k-WL algorithm would refine the colors of all 3-tuples of nodes such as $\{v_1, v_3, v_4\}$ in the above graph

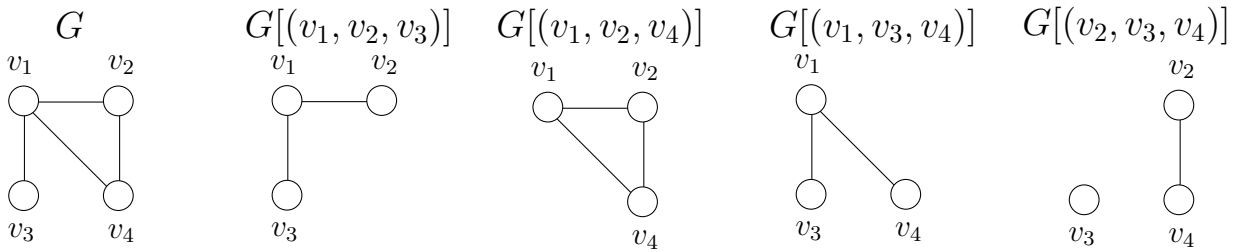
Higher Order Weisfeiler and Leman (WL) Algorithm (2/4)

There are two different variants of high order WL algorithms: k-WL and k-FWL (Folklore WL)

- they both construct a coloring of all k-tuples (potentially all permutations) of nodes, i.e., $c : V^k \to \Sigma$ where Σ is a set of colors
- testing isomorphism of two graphs G, G' is then performed by comparing the histograms of colors produced by those algorithms
- Let c_i denote the coloring in the i-th iteration of the algorithm
- Let v denote a k-tuple of nodes, i.e., $\mathrm{v} = (v_1, \ldots, v_k) \in V^k$
- Then, $c_i(\mathrm{v})$ is the color of k-tuple v in the i-th iteration of the algorithm
- In both algorithms, the initial coloring c_0 is defined using the isomorphism type of each k-tuple

  Example: we have $\binom{4}{3} = 4$ 3-tuples of nodes when $n = 4$

$c_0\big((v_1, v_2, v_3)\big) =$ ●
$c_0\big((v_1, v_2, v_4)\big) =$ ●
$c_0\big((v_1, v_3, v_4)\big) =$ ●
$c_0\big((v_2, v_3, v_4)\big) =$ ●



Higher Order Weisfeiler and Leman (WL) Algorithm (3/4)

- In the standard WL algorithm, the neighborhood of each node consists of the nodes that are adjacent to that node
- **Question:** How exactly is the neighborhood of a k-tuple defined?

Higher Order Weisfeiler and Leman (WL) Algorithm (3/4)

- In the standard WL algorithm, the neighborhood of each node consists of the nodes that are adjacent to that node
- **Question:** How exactly is the neighborhood of a k-tuple defined?
- Given a k-tuple $\mathrm{v} = (v_1, \ldots, v_k)$, the definition of a neighborhood differs in the WL and FWL variants:

$$\text{WL} \ : \mathcal{N}_j(\mathrm{v}) = \Big\{(v_1, \ldots, v_{j-1}, w, v_{j+1}, \ldots, v_k) \mid w \in V\Big\}, j \in [k]$$

$$\text{FWL} \ : \mathcal{N}^F_w(\mathrm{v}) = \Big((w, v_2, \ldots, v_k), (v_1, w, \ldots, v_k), \ldots, (v_1, \ldots, v_{k-1}, w)\Big), \ w \in V$$

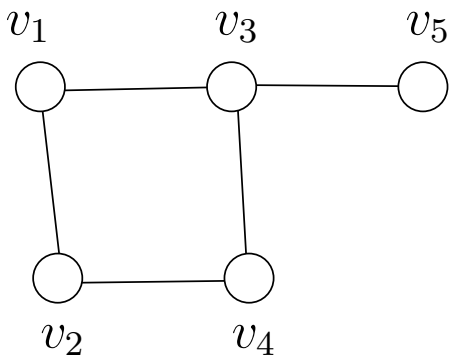
- $\mathcal{N}_j(\mathrm{v})$ is a set of n k-tuples, while $\mathcal{N}^F_w(\mathrm{v})$ is an ordered set of k k-tuples

Example:
Let $\mathrm{v} = (v_1, v_2, v_3)$
Then, the neighborhood of v is defined as follows:

$$\mathcal{N}_2(\mathrm{v}) = \Big\{(v_1, v_1, v_3), (v_1, v_2, v_3), (v_1, v_3, v_3), (v_1, v_4, v_3), (v_1, v_5, v_3)\Big\}$$

or

$$\mathcal{N}^F_{v_5}(\mathrm{v}) = \Big((v_5, v_2, v_3), (v_1, v_5, v_3), (v_1, v_2, v_5)\Big)$$



Higher Order Weisfeiler and Leman (WL) Algorithm (4/4)

- The coloring update rules for the two variants are:

$$\text{WL} \ : c_i(\mathrm{v}) = \text{hash}\left(c_{i-1}(\mathrm{v}), \left(\left\{\!\left\{ c_{i-1}(\mathrm{w}) \mid \mathrm{w} \in \mathcal{N}_j(\mathrm{v}) \right\}\!\right\} \,\middle|\, j \in [k] \right) \right)$$

$$\text{FWL} \ : c_i^F(\mathrm{v}) = \text{hash}\left(c_{i-1}(\mathrm{v}), \left\{\!\left\{ (c_{i-1}(\mathrm{w}) \mid \mathrm{w} \in \mathcal{N}_w^F(\mathrm{v})) \,\middle|\, w \in V \right\}\!\right\} \right)$$

  where hash is a bijective map from the collection of all possible tuples to Σ

- When $k = 1$ both WL and FWL, degenerate to:

$$c_i(v) = \text{hash}\left(c_{i-1}(v), \left\{\!\left\{ c_{i-1}(u) \mid u \in V \right\}\!\right\} \right)$$

  which will not refine any initial color

- Several known results of WL and FWL algorithms are known:
  - 1-WL and 2-WL have equivalent discrimination power
  - k-FWL is equivalent to $(k+1)$-WL for $k \geq 2$
  - For each $k \geq 2$, there is a pair of non-isomorphic graphs distinguishable by $(k+1)$-WL but not by k-WL

k-GNN (1/2)

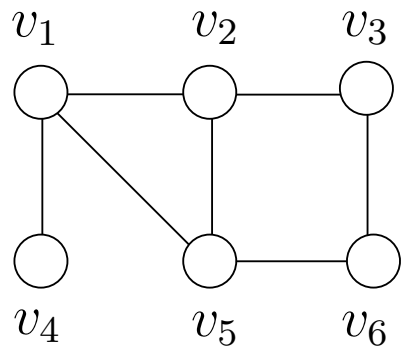
Idea: The k-GNN model relies on a message-passing scheme between subgraphs of cardinality k rather than individual nodes

- It is based on k-WL
  ☹ Due to scalability issues, a set-based variant of k-WL is employed
- Let $A = \{v_1, \ldots, v_k\} \in V^k$ be a set of k nodes
- The neighborhood of A is defined as

$$\mathcal{N}(A) = \left\{ B \in V^k \,\middle|\, |A \cap B| = k - 1 \right\}$$

Example:
Let $A = \{v_2, v_3, v_5\}$
Then, the neighborhood of A is defined as follows:

$$\mathcal{N}(A) = \Big\{ \{v_1, v_2, v_3\}, \{v_2, v_3, v_4\}, \{v_2, v_3, v_6\}, \{v_1, v_2, v_5\}, \{v_2, v_4, v_5\}, \{v_2, v_5, v_6\}, \{v_1, v_3, v_5\}, \{v_3, v_4, v_5\}, \{v_3, v_5, v_6\} \Big\}$$



k-GNN (2/2)

- Set of neighbors can be split into two subsets:
  - the local neighborhood $\mathcal{N}_L(A)$ consists of all $B \in \mathcal{N}(A)$ such that $(v, w) \in E$ for the unique $v \in A \setminus B$ and the unique $w \in B \setminus A$
  - the global neighborhood $\mathcal{N}_G(A)$ then is defined as $\mathcal{N}(A) \setminus \mathcal{N}_L(A)$
  - In the previous example,
    $\mathcal{N}_L(A) = \{\{v_1, v_2, v_3\}, \{v_2, v_3, v_6\}, \{v_2, v_5, v_6\}, \{v_1, v_3, v_5\}\}$
- Initially, the model colors each element $A \in V^k$ with its isomorphism type
- Each message passing layer of the k-GNN model is defined as:

$$\mathrm{h}_A^{(t+1)} = \sigma\left(\mathrm{W}_1^{(t)}\,\mathrm{h}_A^{(t)} + \sum_{B \in \mathcal{N}_L(A) \cup \mathcal{N}_G(A)} \mathrm{W}_2^{(t)}\,\mathrm{h}_B^{(t)}\right)$$

- To scale k-GNNs to larger datasets, the global neighborhood can be omitted:

$$\mathrm{h}_A^{(t+1)} = \sigma\left(\mathrm{W}_1^{(t)}\,\mathrm{h}_A^{(t)} + \sum_{B \in \mathcal{N}_L(A)} \mathrm{W}_2^{(t)}\,\mathrm{h}_B^{(t)}\right)$$

[Morris et al., AAAI'19]

Local δ-k-GNN

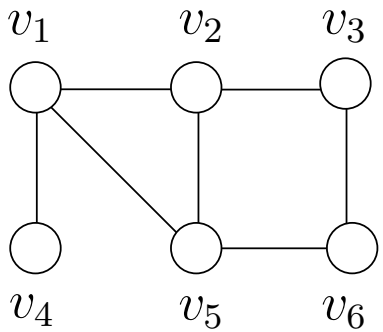
Idea: Another model that relies on a message-passing scheme between subgraphs of cardinality k to achieve higher expressive power

- Based on δ-k-LWL, a local variant of k-WL
- Given a k-tuple $\mathrm{v} = (v_1, \ldots, v_k)$, the neighborhood of v is an ordered set of k sets defined as follows:

$$\mathcal{N}(\mathrm{v}) = \left(\{(w, v_2, \ldots, v_k) \mid w \in \mathcal{N}(v_1)\}, \ldots, \{(v_1, v_2, \ldots, w) \mid w \in \mathcal{N}(v_k)\} \right)$$

Example:
Let $\mathrm{v} = \{v_2, v_3, v_5\}$
Then, the neighborhood of v is defined as follows:

$$\mathcal{N}(\mathrm{v}) = \Big(\{(v_1, v_3, v_5), (v_3, v_3, v_5), (v_5, v_3, v_5)\}, \{(v_2, v_2, v_5), (v_2, v_6, v_5)\}, \{(v_2, v_3, v_1), (v_2, v_3, v_2), (v_2, v_3, v_6)\} \Big)$$



[Morris et al., NeurIPS'20]

Local δ-k-GNN

Idea: Another model that relies on a message-passing scheme between subgraphs of cardinality k to achieve higher expressive power

- Based on δ-k-LWL, a local variant of k-WL
- Given a k-tuple $\mathrm{v} = (v_1, \ldots, v_k)$, the neighborhood of v is an ordered set of k sets defined as follows:

$$\mathcal{N}(\mathrm{v}) = \Big(\big\{ (w, v_2, \ldots, v_k) \mid w \in \mathcal{N}(v_1) \big\}, \ldots, \big\{ (v_1, v_2, \ldots, w) \mid w \in \mathcal{N}(v_k) \big\} \Big)$$

- The coloring update rule is:

$$c_i(\mathrm{v}) = \mathrm{hash}\Big(c_{i-1}(\mathrm{v}), \big\{\!\big\{ (c_{i-1}(\mathrm{w}) \mid \mathrm{w} \in \mathcal{N}(\mathrm{v})) \big\}\!\big\} \Big)$$

- Initially, the model colors each k-tuple with its isomorphism type
- Each message passing layer of the k-GNN model is defined as:

$$\mathrm{h}_{\mathrm{v}}^{(t+1)} = f\left(\mathrm{W}_1^{(t)} \, \mathrm{h}_{\mathrm{v}}^{(t)} + \sum_{\mathrm{w} \in \mathcal{N}(\mathrm{v})} \mathrm{W}_2^{(t)} \, \mathrm{h}_{\mathrm{w}}^{(t)} \right)$$

[Morris et al., NeurIPS'20]

Relational Pooling (1/2)

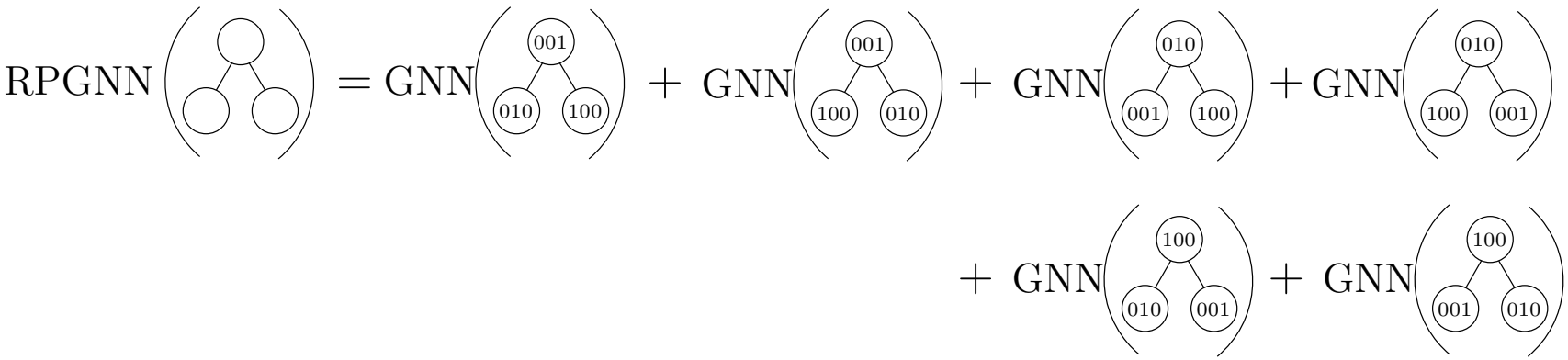
Idea: increase a model's expressive power by considering all possible permutations of nodes!

- Given graph G consisting of n nodes, let $\mathsf{A} \in \mathbb{R}^{n \times n}$, and $\mathsf{X} \in \mathbb{R}^{n \times d}$ denote the adjacency matrix and matrix of node features of G, respectively
- Then, a representation for the entire graph is produced as follows:

$$\mathsf{h}_G = \frac{1}{n!} \sum_{\mathsf{P} \in \Pi} f(\mathsf{P}\,\mathsf{A}\,\mathsf{P}^\top, \mathsf{P}\,\mathsf{X})$$

  where Π is the set of $n \times n$ permutation matrices

- Example of an RP model:
  (1) add unique IDs as node features
  (2) run any GNN
  (3) sum over all permutations of IDs



Relational Pooling (2/2)

- RP is very powerful
  ☺ RP is universal graph representation if f is expressive enough!

Relational Pooling (2/2)

- RP is very powerful
  ☺ RP is universal graph representation if f is expressive enough!
- ☹ But intractable!!
  since $n!$ is very large even for small values of n
  - for example, for $n = 10$, $n! = 3,628,800$
- Therefore, sum of all permutations needs to be approximated
- At each epoch, just sample one set of permutation-sensitive IDs

$$\overline{\text{RPGNN}}\left(\quad\right) = 0 + 0 + 0 + \text{GNN}\left(\begin{matrix}010\\100\quad 001\end{matrix}\right) + 0 + 0 + 0$$

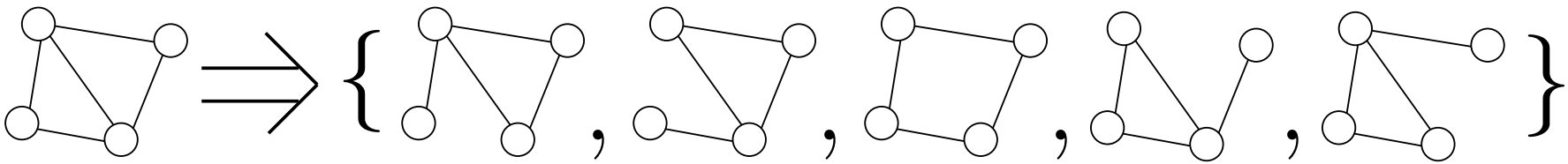
- Works well in practice!

[Murphy et al., ICML'19]

Subgraph GNNs

Idea: We can decompose a graph into a set of subgraphs and process those subgraphs

- Step 1: Extract subgraphs from a graph and represent the graph as a set of its subgraphs

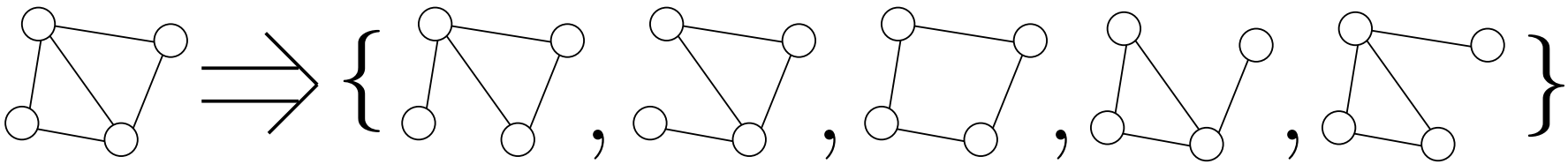


- Step 2: We can generate a representation for the graph by mapping the set of subgraphs into a vector

Subgraph GNNs

Idea: We can decompose a graph into a set of subgraphs and process those subgraphs

- Step 1: Extract subgraphs from a graph and represent the graph as a set of its subgraphs

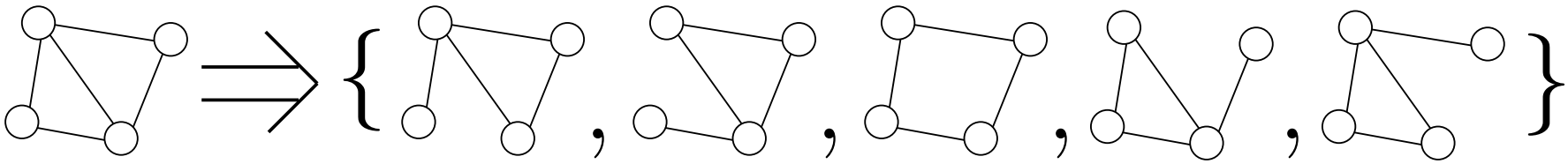


- Step 2: We can generate a representation for the graph by mapping the set of subgraphs into a vector
- However, two main challenges arise:
  1. How to extract subgraphs from a given graph?
     $\hookrightarrow$ different models propose different policies
  2. How to process sets of subgraphs?
     $\hookrightarrow$ each subgraph can be mapped into a vector and then, set of vectors mapped into a single representation

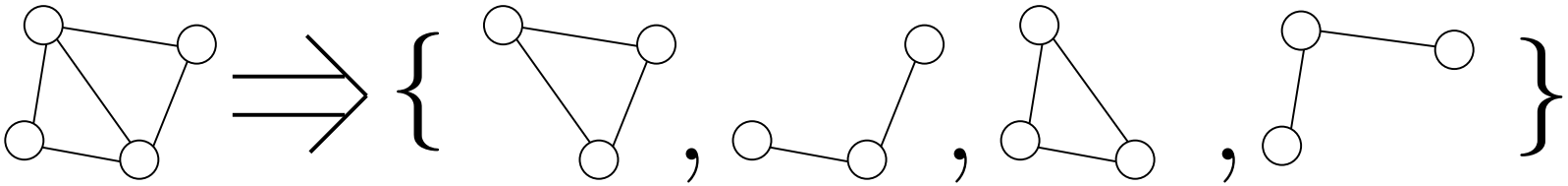
How to Extract Subgraphs from a Given Graph?

Different policies can be employed

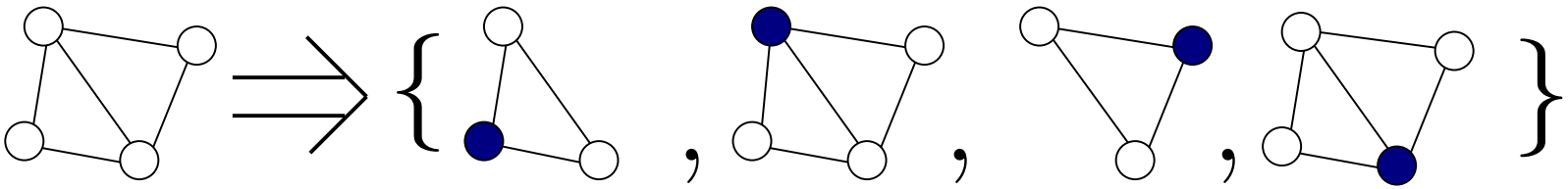
- Edge-deleted subgraphs



- Node-deleted subgraphs



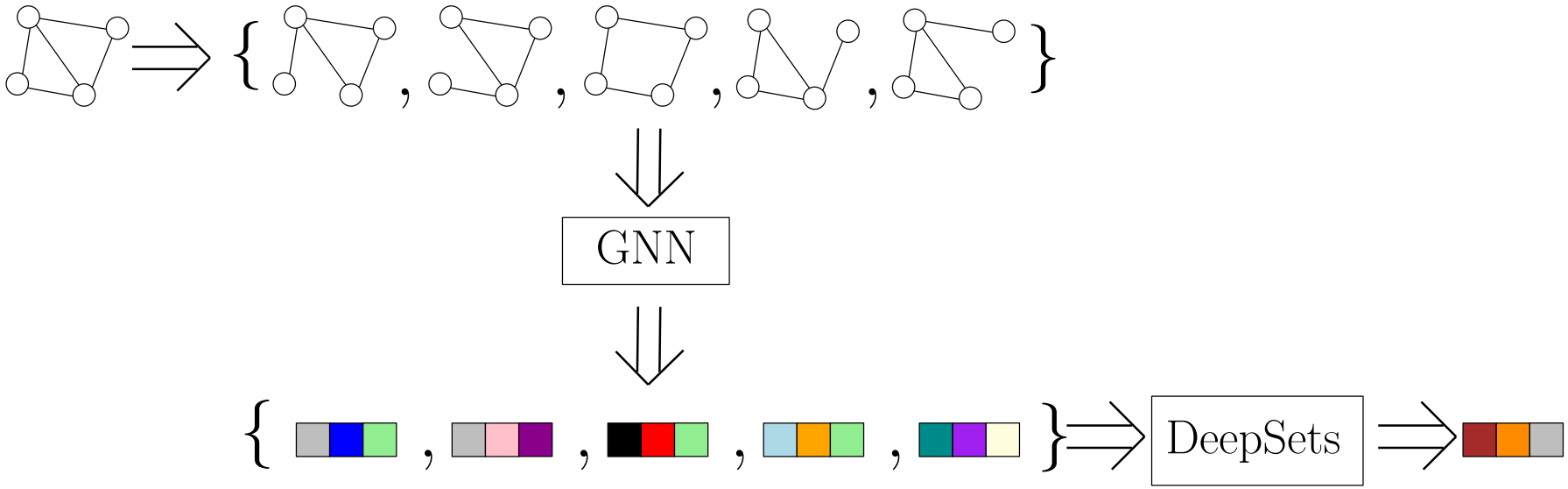
- Ego-networks (rooted)



[Bevilacqua et al., ICLR'22]

How to Process Sets of Subgraphs?

- Use some GNN model (e.g., GIN) to obtain a vector representation for each subgraph
- Then, use DeepSets to obtain a final representation for entire graph



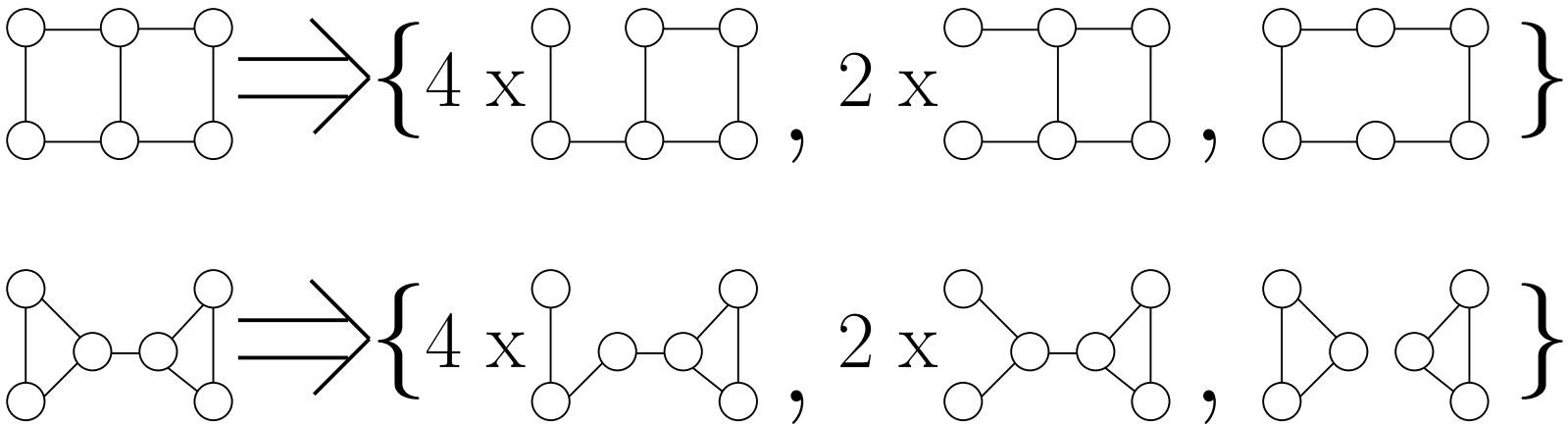
- Can such a model distinguish between more non-isomorphic pairs of graphs than the WL algorithm?

Subgraph GNNs can Improve Expressive Power (1/2)

- Consider the following two graphs:

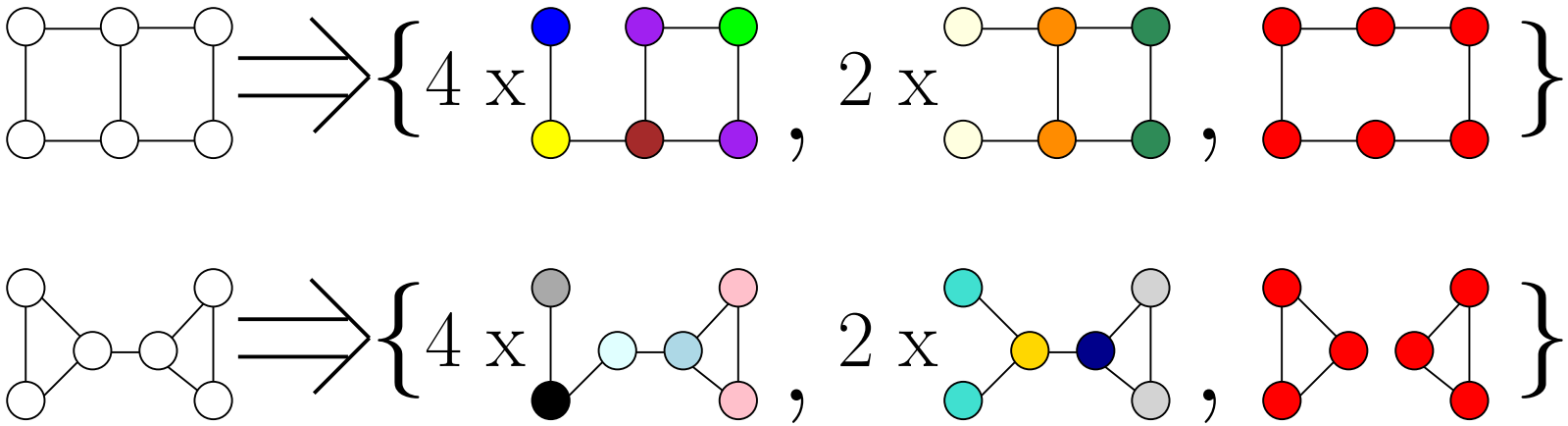
- WL cannot detect that the two graphs are not isomorphic

- The two graphs can be decomposed into edge deleted subgraphs as follows:



Subgraph GNNs can Improve Expressive Power (2/2)

- If we run the WL algorithm on the 6 types of subgraphs, we end up with the following colors:



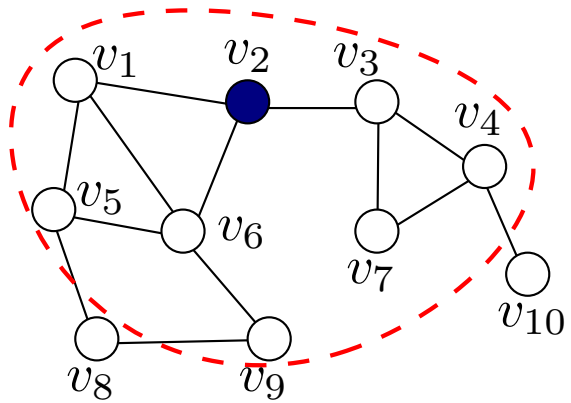
- Thus, we end up with two different multisets of vectors
- These can be fed to a model that can embed different multisets into different vectors (e.g., DeepSets)
- Such a model is more powerful than the WL algorithm

k-hop Graph Neural Networks (1/2)

A subgraph GNN model that decomposes each graph into a set of neighborhood subgraphs

Sprecifically, to update the representation of a node, the following steps are followed:

- Extract k-hop neighborhood of node $\mathcal{N}_k(v)$
- Let $R_d(v)$ denote the set of nodes at distance (hop count) exactly $d > 0$ from v
- Update representations of nodes in $\mathcal{N}_k(v)$ starting from the those in $R_k(v)$, then those in $R_{k-1}(v)$ and so on
- The model learns a new feature vector h_u for every $u \in \mathcal{N}_k(v)$
  $\hookrightarrow$ these representations are only learned for the purpose of updating the root node's representation



$\mathcal{N}_2(v_2) = \{v_1, v_3, v_4, v_5, v_6, v_7, v_9\}$
$R_1(v_2) = \{v_1, v_3, v_6\}$
$R_2(v_2) = \{v_4, v_5, v_7, v_9\}$

☺ The model is shown to be more powerful than the WL algorithm
$\hookrightarrow$ it can detect triangles, while WL cannot

[Nikolentzos et al., Neural Networks 130]

k-hop Graph Neural Networks (2/2)

At each hop level, two different types of node representation updates take place:

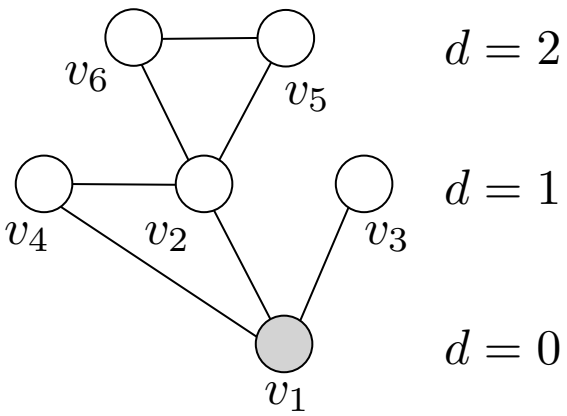
(1) an update based on neighbors located at the same hop level

(2) an update based on neighbors located at the immediately higher hop level (if any)

Update #1:

- Let $u \in R_d(v)$ be a node whose shortest path distance from v is equal to d
- Let $\mathcal{B} = \mathcal{N}_1(u) \cap R_d(v)$ denote the neighbors of u at the same level of the k-hop subgraph (e.g., for node v_6, $\mathcal{B} = \{v_5\}$)
- If $\mathcal{B}$ is not empty, the representation h_u of u is updated as follows:

$$\mathsf{h}'_u = \mathsf{MLP}^{\mathsf{i}}_d(\mathsf{h}_u) + \sum_{w \in \mathcal{B}} \mathsf{MLP}^{\mathsf{ii}}_d(\mathsf{h}_w)$$



- For instance, the representation h_{v_6} of node v_6 is updated as follows:

$$\mathsf{h}'_{v_6} = \mathsf{MLP}^{\mathsf{i}}_2(\mathsf{h}_{v_6}) + \mathsf{MLP}^{\mathsf{ii}}_2(\mathsf{h}_{v_5})$$

k-hop Graph Neural Networks (2/2)

At each hop level, two different types of node representation updates take place:

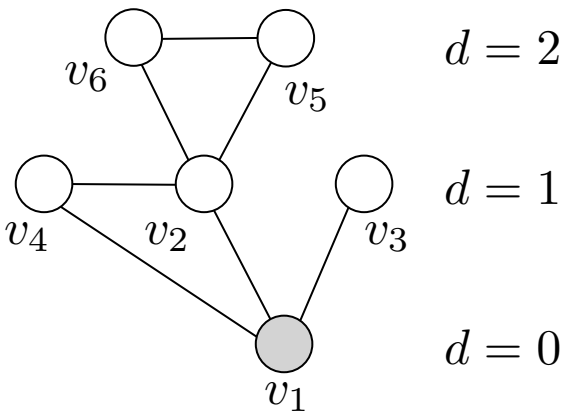
(1) an update based on neighbors located at the same hop level

(2) an update based on neighbors located at the immediately higher hop level (if any)

Update #2:

- Let $u \in R_d(v)$ be a node whose shortest path distance from v is equal to d.
- Let $\mathcal{D} = \mathcal{N}_1(u) \cap R_{d+1}(v)$ denote the neighbors of u that belong to level $d+1$ of the k-hop subgraph
- If $\mathcal{D}$ is not empty, the representation h_u of u is updated as follows:

$$\mathrm{h}'_u = \mathrm{MLP}^{\mathrm{iv}}_d(\mathrm{h}_u) + \sum_{w \in \mathcal{D}} \mathrm{MLP}^{\mathrm{iii}}_{d+1}(\mathrm{h}_w)$$



- For instance, the representation h_{v_2} of node v_2 is updated as follows:

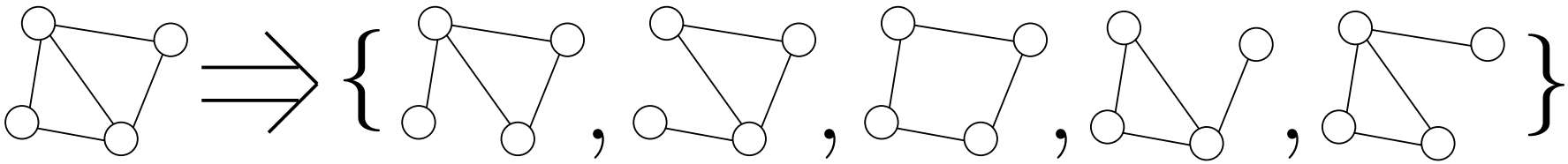
$$\mathrm{h}'_{v_2} = \mathrm{MLP}^{\mathrm{iv}}_1(\mathrm{h}_{v_2}) + \mathrm{MLP}^{\mathrm{iii}}_2(\mathrm{h}_{v_5}) + \mathrm{MLP}^{\mathrm{iii}}_2(\mathrm{h}_{v_6})$$

Subgraph Sampling

Problem: Full policies might generate too many subgraphs ☹

Example: Consider the edge-deleted subgraphs policy

- let m denote the number of edges of a graph
- m subgraphs need to be proceesed and each one of the m subgraphs has almost the same size as the input graph
- this is prohibitive for large values of m (e.g., if m is in the order of millions)
- in the case of the following graph (which is very small), we need to process 5 graphs instead of 1



Solution: Sample subsets of the entire set of subgraphs

- Allows a model to process large graphs
- Reduces training time
- Works well in practice

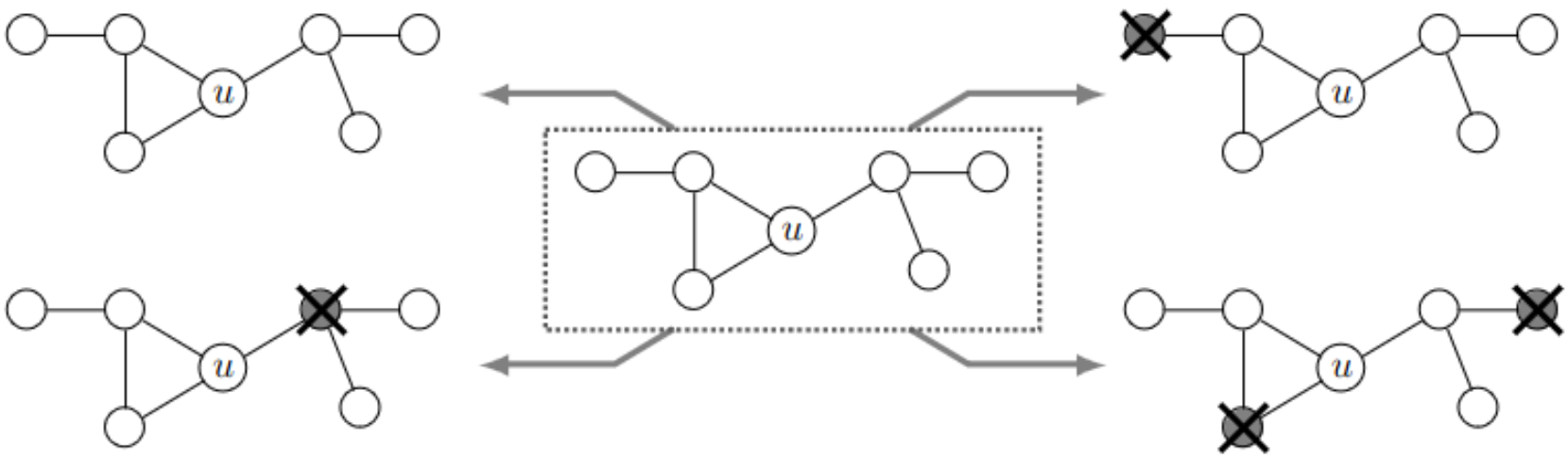
DropGNN (1/2)

DropGNNs deletes a node or a set of nodes from a node's neighborhood

Capitalizes on the idea of sampling: it executes multiple independent runs, with some of the nodes randomly and independently dropped in each run

In different runs, different number of nodes are dropped

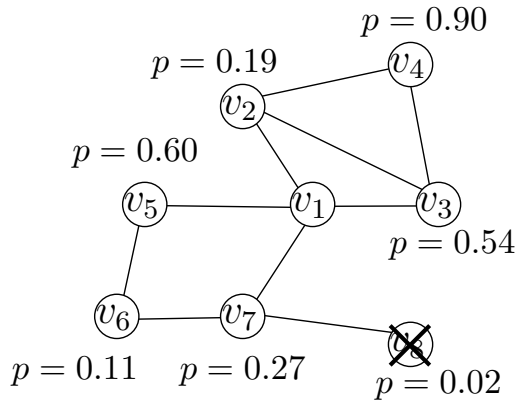
- a dropout combination is a subset of nodes dropping out
- a k-dropout is a dropout combination that has size k
- 4 examples of dropout combinations from node u's 2-hop neighborhood are given below:



Source: **[Papp et al., NeurIPS'21]**

DropGNN (2/2)

- In each run, every node is removed with probability p, independently from all other nodes
  $\hookrightarrow$ identical node embeddings are only produced if the same nodes are dropped
- Typically use a relatively small dropout probability p
  - in each run, only a few nodes are removed
  - as a result, 1-dropouts are frequent, while k-dropouts (for $k > 1$) are unlikely
  - for instance, if $p = 0.1$, only node v_8 of the graph on the right is dropped



- To reduce the effect of randomization on the final outcome, multiple independent runs need to be executed
  - number of runs needs to be large $\rightarrow$ this ensures that the set of observed dropout combinations is close to the actual probability distribution of dropouts
- The independent runs produce different final embeddings for each node!! They need to be merged into a single final embedding
  - the model first applies a transformation on each embedding
  - it then uses the sum operator

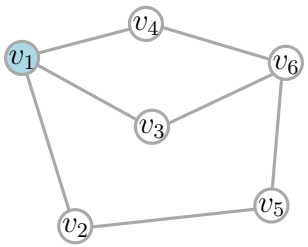
Paths Collections

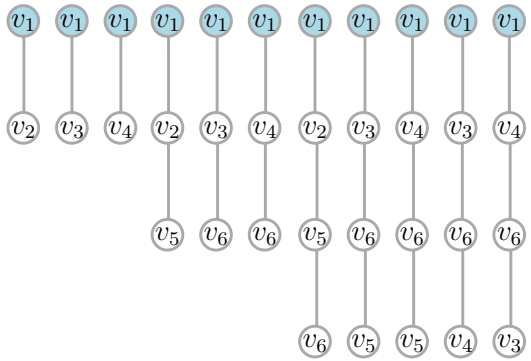
Idea: Explicitly representing paths (and nodes) instead of only nodes in GNNs should lead to more accurate and expressive models

Paths Collections

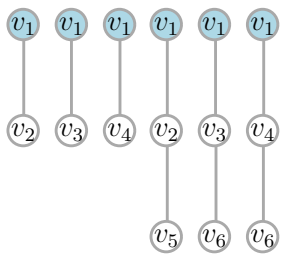
Idea: Explicitly representing paths (and nodes) instead of only nodes in GNNs should lead to more accurate and expressive models

We make use of three different collections of paths:

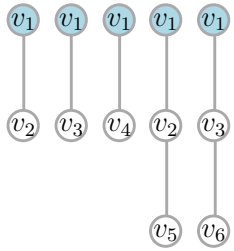




all paths ($\mathcal{AP}$)



all shortest paths ($\mathcal{SP}^+$)

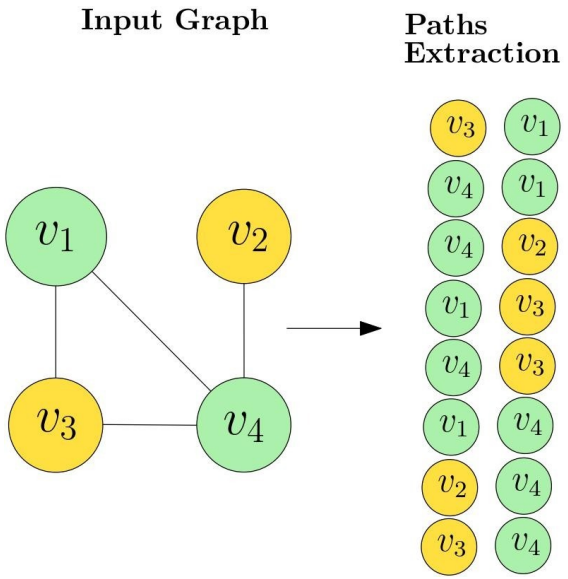


single shortest paths ($\mathcal{SP}$)

Path Neural Networks

1. Given a collection of paths in a graph,

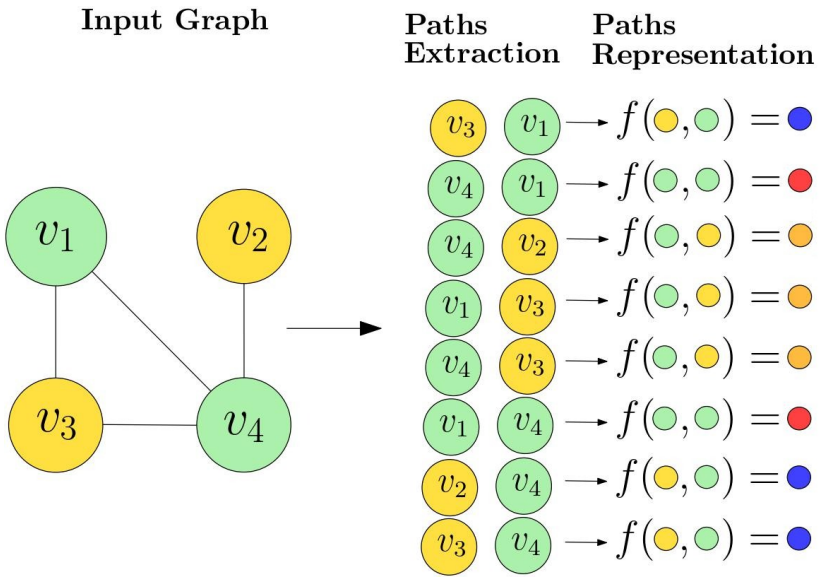
At layer ℓ of the model we process paths of length ℓ, e.g., $\ell = 2$



Path Neural Networks

1. Given a collection of paths in a graph, we apply an LSTM to learn path representations.

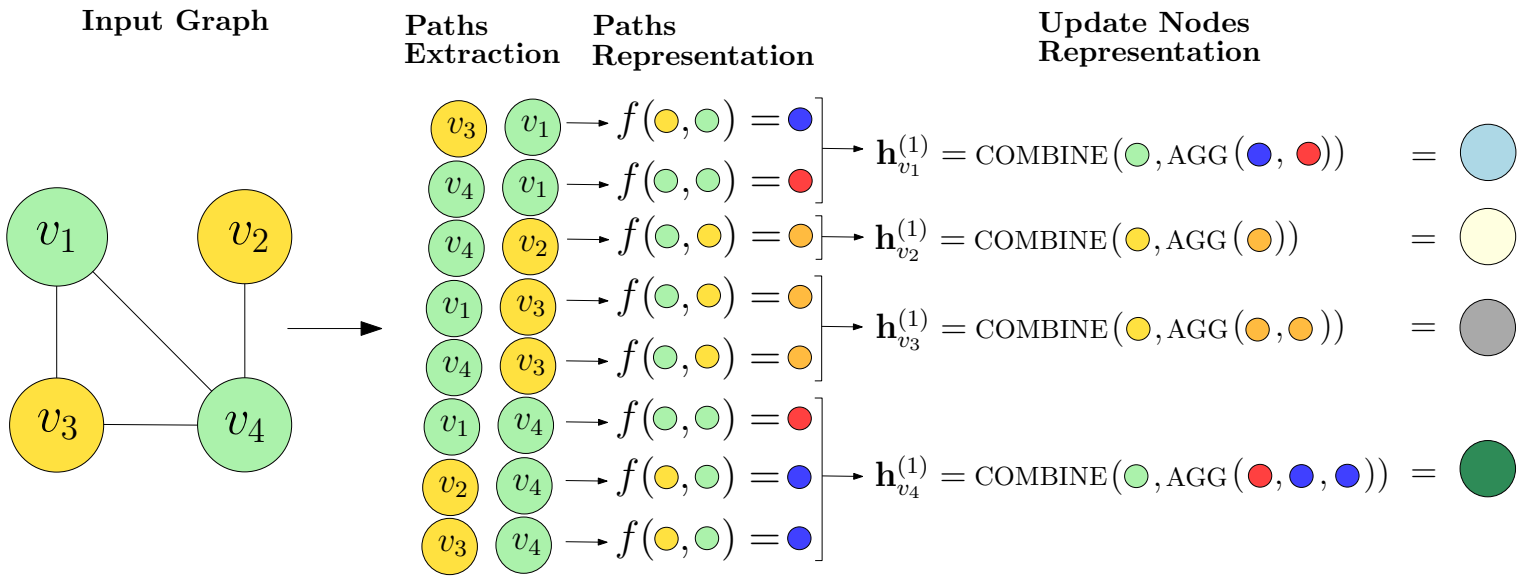
At layer ℓ of the model we process paths of length ℓ, e.g., $\ell = 2$



Path Neural Networks

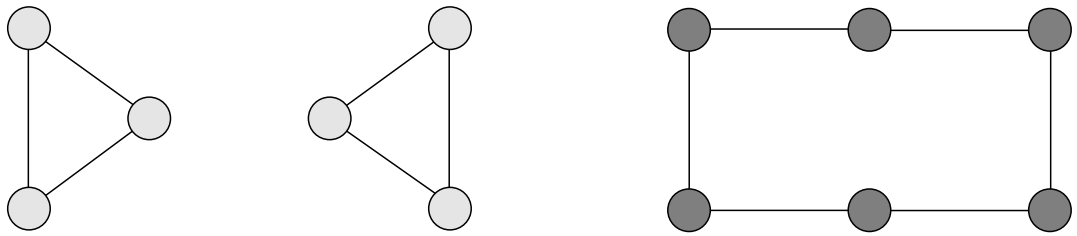
1. Given a collection of paths in a graph, we apply an LSTM to learn path representations.
2. We then aggregate the representations of all paths emanating from a node to form updated node representations.

At layer ℓ of the model we process paths of length ℓ, e.g., $\ell = 2$



Path Neural Networks – Theoretical Results

- PathNN-$\mathcal{AP}$ and PathNN-$\mathcal{SP}^+$ are strictly more powerful than the WL test
- For example, the following two graphs
  - cannot be distinguished by the WL algorithm
  - can be distinguished by PathNN-$\mathcal{AP}$ and PathNN-$\mathcal{SP}^+$ for $\ell > 2$



G_1 G_2

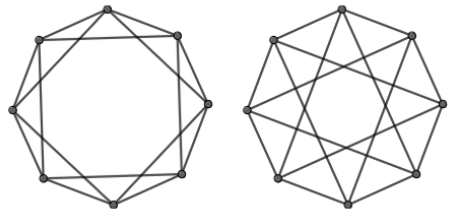
- Why?

Experimental Evaluation for Expressiveness

Experimented with two datasets that evaluate the models' expressive power

(i) CSL dataset

- contains 4-regular graphs with edges connected to form a cycle and containing skip-links between nodes
- graphs are not distinguishable by the 1-WL algorithm
- graph $G_{n,k}$ consists of n nodes such that nodes i and j are connected if and only if $|i-j| \equiv 1$ or $k (\mathrm{mod} n)$
- 150 graphs in total, each consisting of 41 nodes
- graphs are equally divided into 10 isomorphism classes based on the skip-link length of the graph



The $G_{8,2}$ and $G_{8,3}$ graphs.
Source: **[Chen et al., NeurIPS'19]**

(ii) EXP dataset

- a synthetic dataset that contains pairs of graphs that are not distinguishable by the 1-WL algorithm
- each graph encodes a propositional formula, and given a pair G, G' of non-isomorphic graphs, the two graphs have different SAT outcomes
  G encodes a satisfiable formula, while G' encodes an unsatisfiable formula

Experimental Results

Table: Test set classification accuracy of the different models on the CSL and EXP datasets. Best results are highlighted in **bold**. NA means not available.

Model	**CSL** ↑	**EXP-Class** ↑
GCN **[Kipf and Welling, ICLR'17]**	10.0 ± 0.0	50.0 ± 0.0
GraphSAGE **[Hamilton et al., NeurIPS'17]**	10.0 ± 0.0	50.0 ± 0.0
GAT **[Velickovic et al., ICLR'18]**	10.0 ± 0.0	50.0 ± 0.0
GIN **[Xu et al., ICLR'19]**	10.0 ± 0.0	50.0 ± 0.0
RP-GIN **[Murphy et al., ICML'19]**	37.6 ± 12.9	NA
RingGNN **[Chen et al., NeurIPS'19]**	10.0 ± 0.0	NA
PPGN **[Maron et al., NeurIPS'19]**	97.8 ± 10.9	**100.0** ± 0.0
Nested GNN **[Zhang and Li, NeurIPS'21]**	99.9 ± 0.26	NA
PathNN-$\mathcal{SP}$ **[Gaspard et al., ICML'23]**	90.0 ± 0.0	**100.0** ± 0.0
PathNN-$\mathcal{SP}^+$ **[Gaspard et al., ICML'23]**	**100.0** ± 0.0	**100.0** ± 0.0
PathNN-$\mathcal{AP}$ **[Gaspard et al., ICML'23]**	**100.0** ± 0.0	**100.0** ± 0.0

Outline

Transformer

Transformer has become a dominant architecture in many domains:

- natural language processing
- computer vision

The Transformer architecture consists of a composition of Transformer layers

- Each Transformer layer has two parts:
  (i) a self-attention module
  (ii) a position-wise feed-forward network (FFN)
- Let $\mathrm{H} = [\mathrm{h}_1^\top, \ldots, \mathrm{h}_n^\top]^\top \in \mathbb{R}^{n \times d}$ denote the input of self-attention module where d is the hidden dimension and $\mathrm{h}_i \in \mathbb{R}^{1 \times d}$ is the hidden representation at position i
- The input H is projected by three matrices $\mathrm{W}_Q \in \mathbb{R}^{d \times d_K}$, $\mathrm{W}_K \in \mathbb{R}^{d \times d_K}$ and $\mathrm{W}_V \in \mathbb{R}^{d \times d_V}$ to the corresponding representations Q, K, V:

$$\mathrm{Q} = \mathrm{H}\,\mathrm{W}_Q, \qquad \mathrm{K} = \mathrm{H}\,\mathrm{W}_K, \qquad \mathrm{V} = \mathrm{H}\,\mathrm{W}_V$$

- The self-attention is then calculated as

$$\mathrm{A} = \frac{\mathrm{Q}\,\mathrm{K}^\top}{\sqrt{d_K}}$$

$$\mathrm{Attn}(\mathrm{H}) = \mathrm{softmax}(\mathrm{A})\,\mathrm{V}$$

where A is a matrix capturing the similarity between queries and key

Graphormer (1/3)

- Suppose we trivially apply a Transformer to graph data
- ☹ For each node v_i, self-attention only calculates the semantic similarity between v_i and other nodes, without considering the structural information of the graph
- **Idea:** incorporate structural information of graphs into the model
  But what type of structural information?

Graphormer (1/3)

- Suppose we trivially apply a Transformer to graph data
- ☹ For each node v_i, self-attention only calculates the semantic similarity between v_i and other nodes, without considering the structural information of the graph
- **Idea:** incorporate structural information of graphs into the model
  But what type of structural information?
- In a graph, different nodes may have different importance, e.g., celebrities are considered to be more influential than the rest of the users in a social network
- Thus, use Centrality Encoding to capture the node importance in the graph
  $\hookrightarrow$ each node two real-valued embedding vectors according to its indegree and outdegree (which are added to the vector of initial node features):

$$h_v^{(0)} = x_v + z^-_{\deg^-(v)} + z^+_{\deg^+(v)}$$

  where $z^-_{\deg^-(v)} + z^+_{\deg^+(v)} \in \mathbb{R}^d$ are learnable embedding vectors specified by the indegree $\deg^-(v)$ and out-degree $\deg^-(v)$ of v, respectively (could be unified to $\deg(v)$ in case of undirected graphs)

Graphormer (2/3)

- Also need to capture the structural relation between nodes (they lie in a non-Euclidean space and are linked by edges)
- **Idea:** To model such structural information, for each node pair, a learnable embedding is assigned based on their spatial relation
- This is encoded as a bias term in the softmax attention. Thus, the (i, j)-element of the Query-Key product matrix A is computed as:

$$A_{ij} = \frac{(h_{v_i} W_Q)(h_{v_j} W_K)^\top}{\sqrt{d}} + b_{\phi(v_i, v_j)}$$

where $\phi : V \times V \to \mathbb{R}$ is the function that measures the shortest path distance between the two nodes (if the nodes are not connected, the output of ϕ is set to a special value)

Benefits of Transformer layer:

- it allows each node to attend to all other nodes in the graph (while in MPNNs the receptive field is restricted to the neighbors)
- the term $b_{\phi(v_i, v_j)}$ allows nodes to adaptively attend to all other nodes according to the graph structural information
- For example, if $\phi(v_i, v_j) \uparrow \Rightarrow b_{\phi(v_i, v_j)} \downarrow$, model will pay more attention to local neighborhood and less attention to distant nodes

Graphormer (3/3)

- A Graphormer layer is defined as follows:

$$h^{'(t)} = \mathrm{MHA}\big(\mathrm{LN}(h^{(t-1)})\big) + h^{(t-1)}$$
$$h^{(t)} = \mathrm{FFN}\big(\mathrm{LN}(h^{'(t)})\big) + h^{'(t)}$$

  where MHA, LN and FFN denote a multi-head self-attention layer, a normalization layer and a feed-forward block, respectively

- A special node called `[VNode]` is also added to the graph edges are added between `[VNode]` and all the nodes of the graph

- In graph-level tasks, the representation of the entire graph h_G is set equal to the representation of `[VNode]` in the final layer

Challenges:

- complexity of the self-attention module is quadratic
- this restricts Graphormer's application on large graphs

[Ying et al., NeurIPS'21]

Experimental Evaluation

Experiments on two molecular property prediction datasets:

(i) ZINC 12K dataset

- a graph regression dataset
- consists of $12,000$ molecules
- the task is to predict the constrained solubility of molecules, an important chemical property for designing generative GNNs for molecules

(ii) ogbg-molhiv

- a binary graph classification dataset from the Open Graph Benchmark (OGB)
- consists of $41,127$ molecules
- the task is to predict whether a molecule inhibits HIV virus replication or not

All experiments are conducted using available train/val/test splits

Graph Regression Results

Table: Mean absolute error ($\pm$ standard deviation) of the different methods on the ZINC12K dataset. K denotes the number of employed layers. Results are averaged over 10 random seeds. Best performance is highlighted in **bold**. Parameter budget is set to 500K parameters.

	K	**ZINC12K** $\downarrow$
GCN **[Kipf and Welling, ICLR'17]**	16	0.278 $\pm$ 0.003
GraphSAGE **[Hamilton et al., NeurIPS'17]**	16	0.398 $\pm$ 0.002
MoNet **[Monti et al., CVPR'17]**	16	0.292 $\pm$ 0.006
GAT **[Velickovic et al., ICLR'18]**	16	0.384 $\pm$ 0.007
GIN **[Xu et al., ICLR'19]**	5	0.387 $\pm$ 0.015
RingGNN **[Chen et al., NeurIPS'19]**	2	0.353 $\pm$ 0.019
PPGN **[Maron et al., NeurIPS'19]**	3	0.256 $\pm$ 0.054
GNNML3 **[Balcilar et al., ICML'21]**	NA	0.161 $\pm$ 0.006
Graphormer **[Ying et al., NeurIPS'21]**	NA	0.122 $\pm$ 0.006
CIN **[Bodnar et al., NeurIPS'21]**	NA	**0.079** $\pm$ 0.006
ESAN **[Bevilacqua et al., ICLR'22]**	NA	0.102 $\pm$ 0.003
KP-GIN **[Feng et al., NeurIPS'22]**	NA	0.093 $\pm$ 0.007
AgentNet **[Martinkus et al., ICLR'23]**	NA	0.258 $\pm$ 0.033
PathNN **[Gaspard et al., ICML'23]**	4	0.090 $\pm$ 0.004

Graph Classification Results

Table: ROC-AUC score ($\pm$ standard deviation) of the different methods on the ogbg-molhiv dataset. Results are averaged over 10 random seeds. Best performance is highlighted in **bold**.

	ogbg-molhiv $\uparrow$
GCN **[Kipf and Welling, ICLR'17]**	76.06 $\pm$ 0.97
GIN **[Xu et al., ICLR'19]**	75.58 $\pm$ 1.40
GSN **[Bouritsas et al., TPAMI 45(1)]**	77.99 $\pm$ 1.00
HIMP **[Fey et al., arXiv:2006.12179]**	78.80 $\pm$ 0.82
PNA **[Corso et al., NeurIPS'20]**	79.05 $\pm$ 1.32
DGN **[Beaini et al., ICML'21]**	79.70 $\pm$ 0.97
Graphormer **[Ying et al., NeurIPS'21]**	80.51 $\pm$ 0.53
CIN **[Bodnar et al., NeurIPS'21]**	**80.94** $\pm$ 0.57
ESAN **[Bevilacqua et al., ICLR'22]**	78.00 $\pm$ 1.42
E-SPN **[Abboud et al., LOG'23]**	77.10 $\pm$ 1.20
GRWNN **[Nikolentzos and Vazirgiannis, AISTATS'23]**	78.38 $\pm$ 0.99
AgentNet **[Martinkus et al., ICLR'23]**	78.33 $\pm$ 0.69
PathNN **[Gaspard et al., ICML'23]**	79.17 $\pm$ 1.09

Random Walk Graph Neural Network (RWNN)

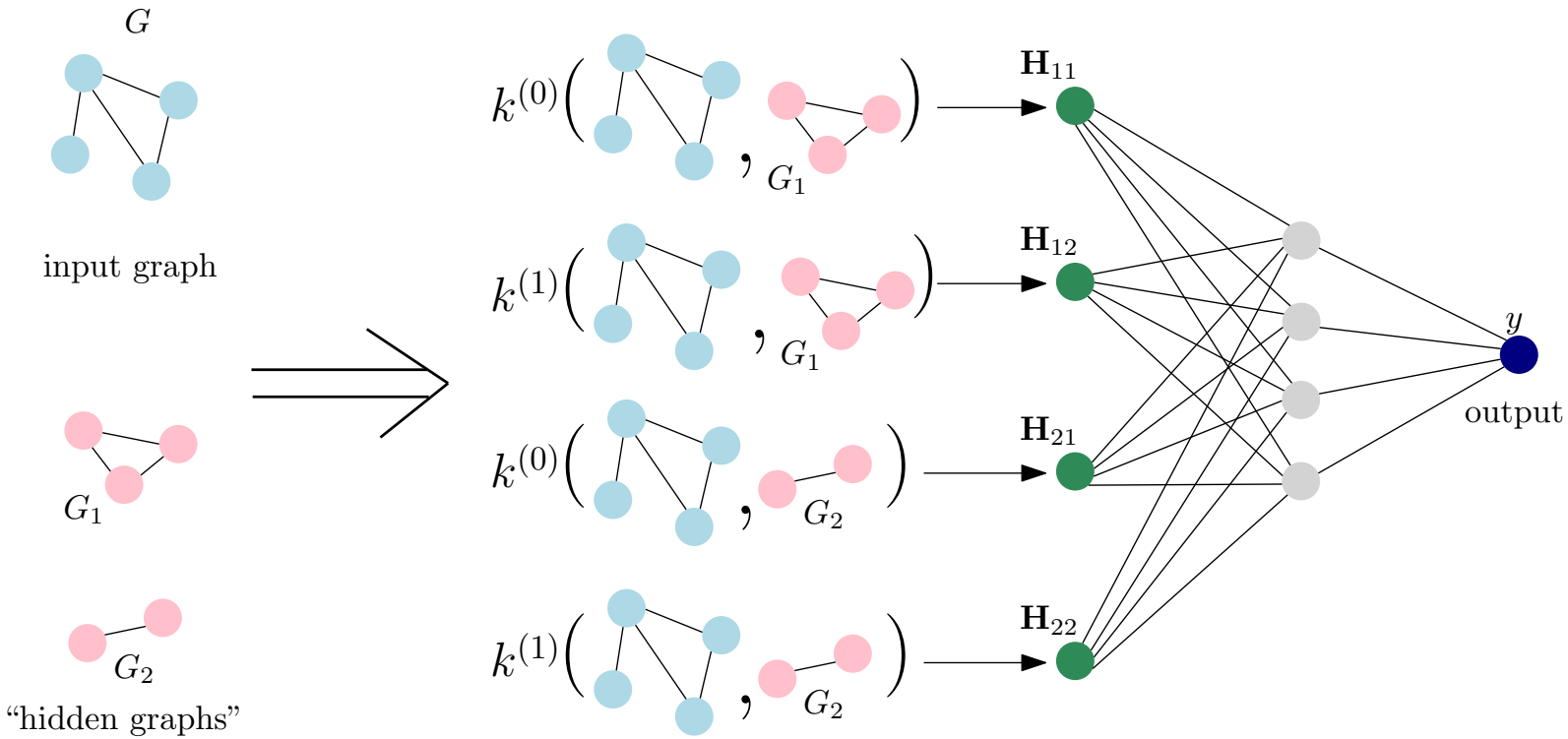
Idea: Existing graph neural networks are counter-intuitive
$\hookrightarrow$ features they learn are in the form of vectors

The Random Walk Graph Neural Network is a model

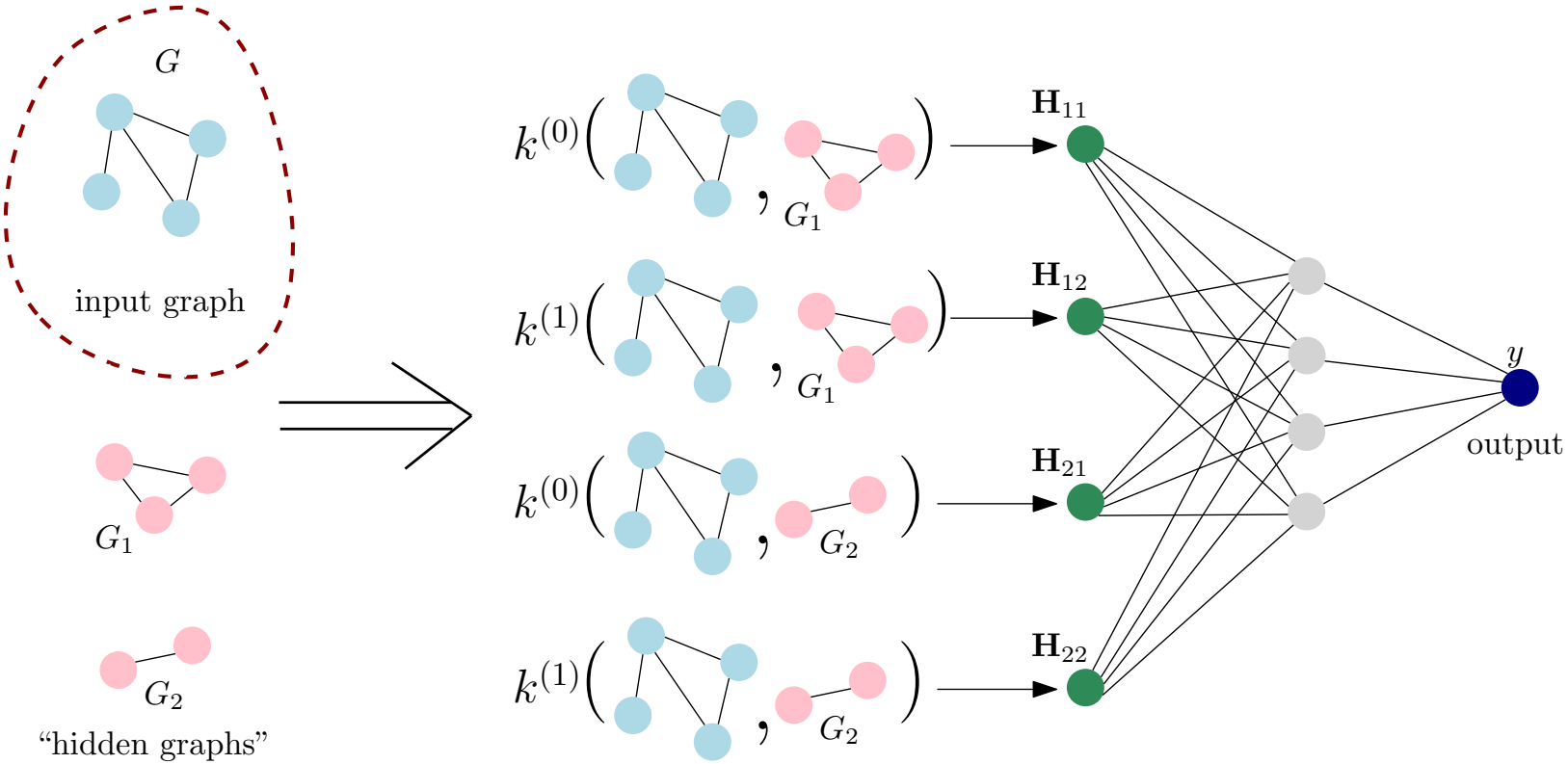
- employs a random walk kernel to learn *graph features* (in the form of fixed-size graphlets) that contribute to interpretable/intuitive graph representations
- An efficient computation scheme to reduce the time and space complexity of the proposed model

[Nikolentzos and Vazirgiannis, NeurIPS'20]

Random Walk Graph Neural Network (RWNN)

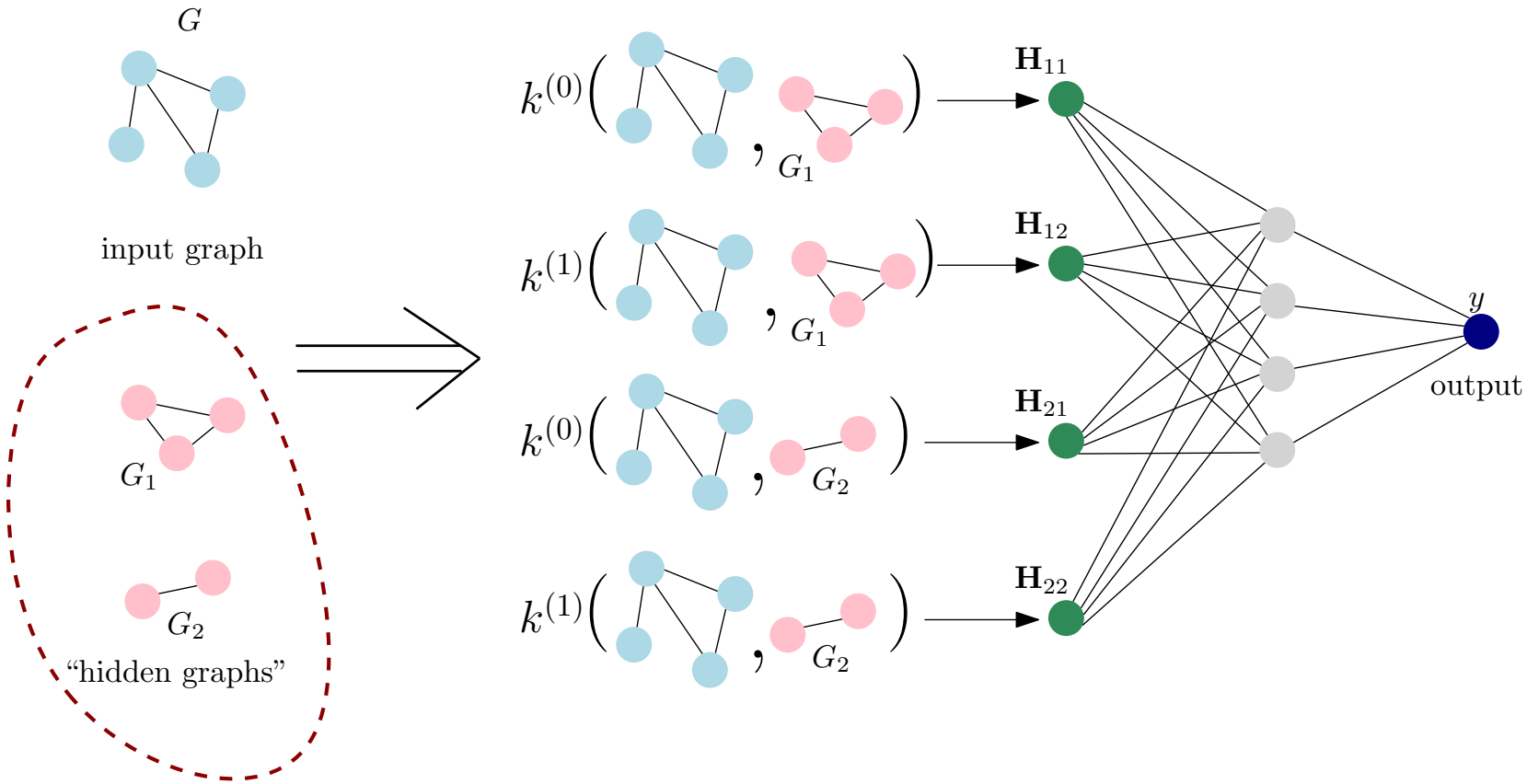


Random Walk Graph Neural Network (RWNN)



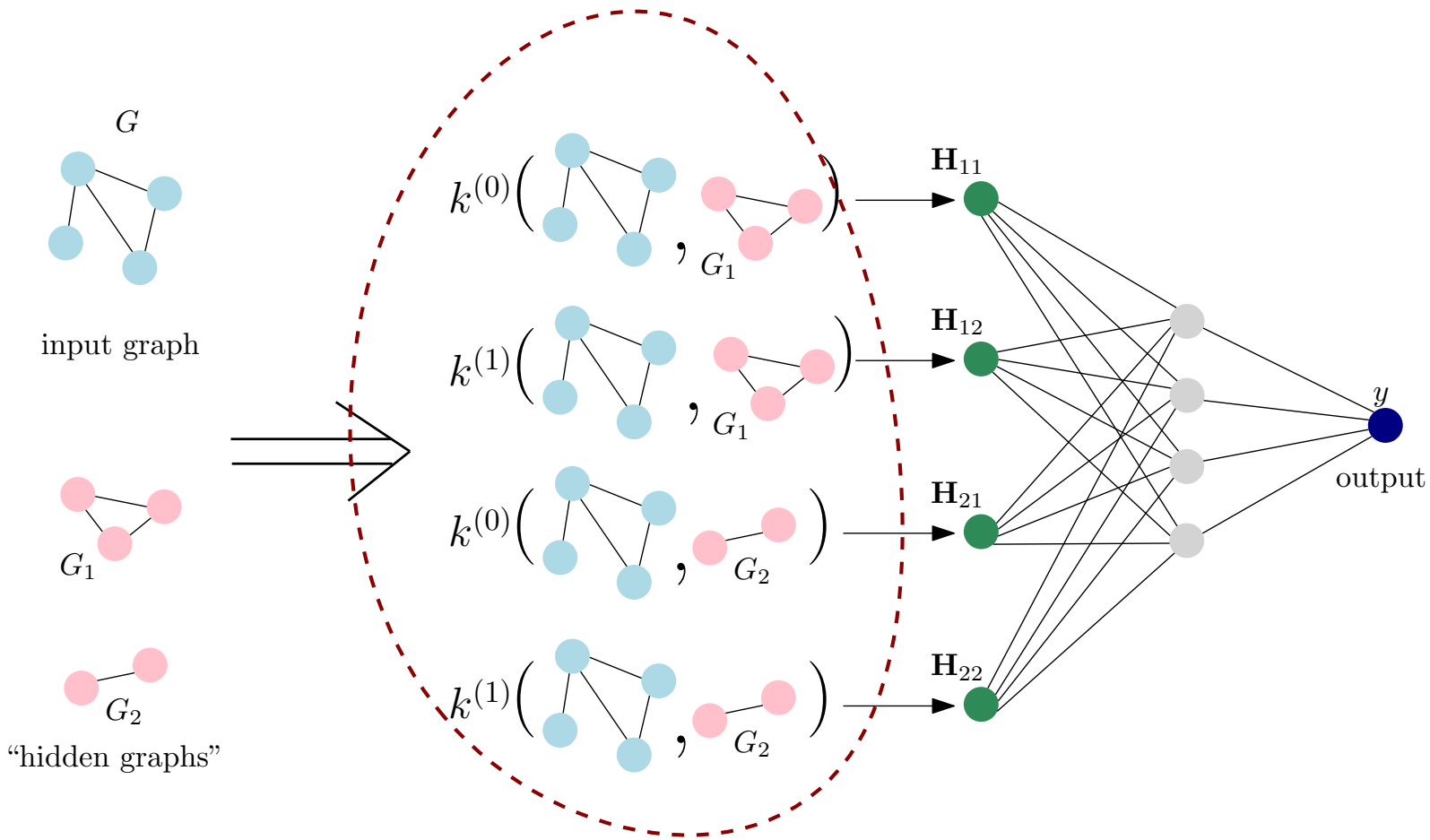
Given an input graph G

Random Walk Graph Neural Network (RWNN)



and a set of trainable "hidden graphs" $G_1, G_2, \ldots$

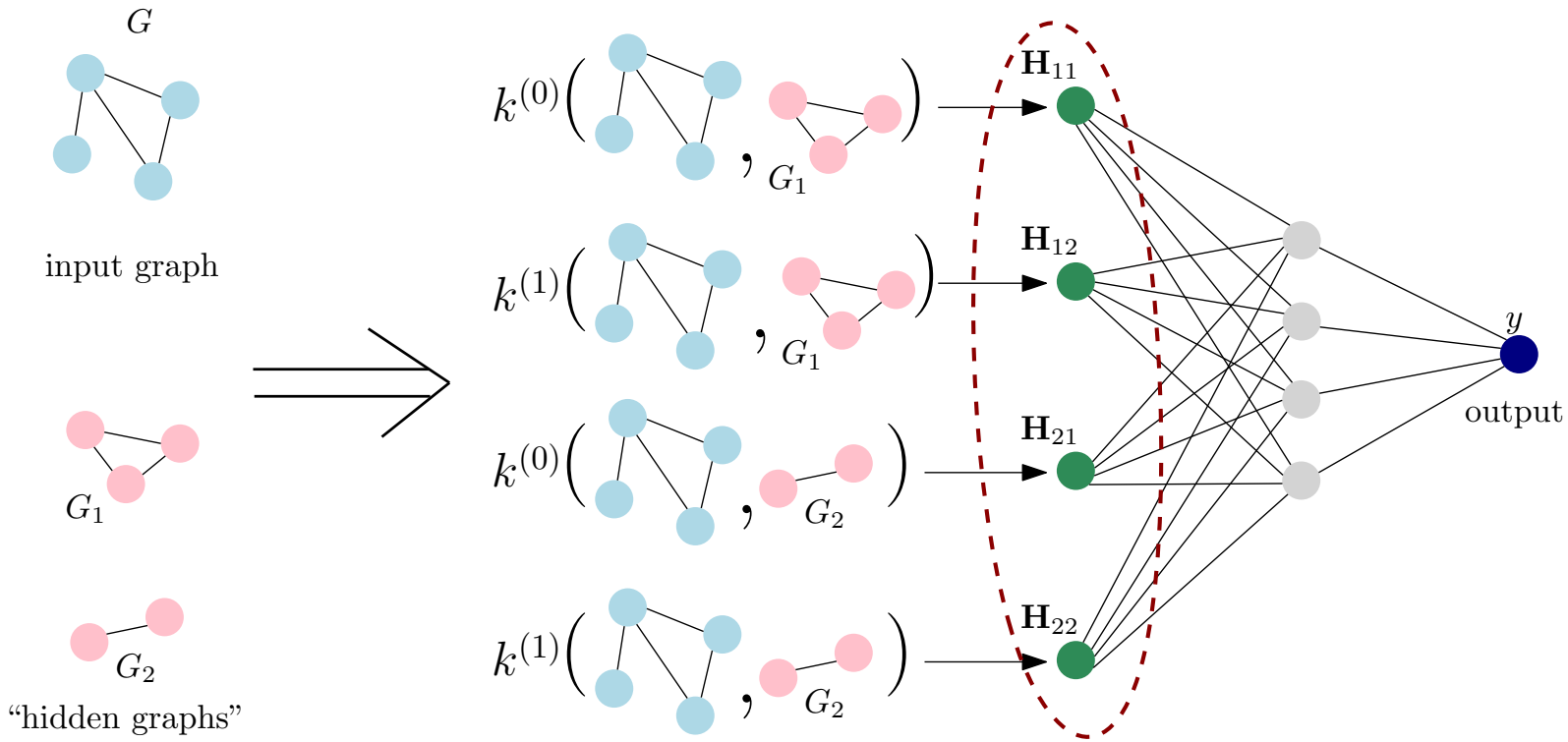
Random Walk Graph Neural Network (RWNN)



The model computes the following kernel between the input graph G and each "hidden graph" $G_{\mathtt{i}}$:

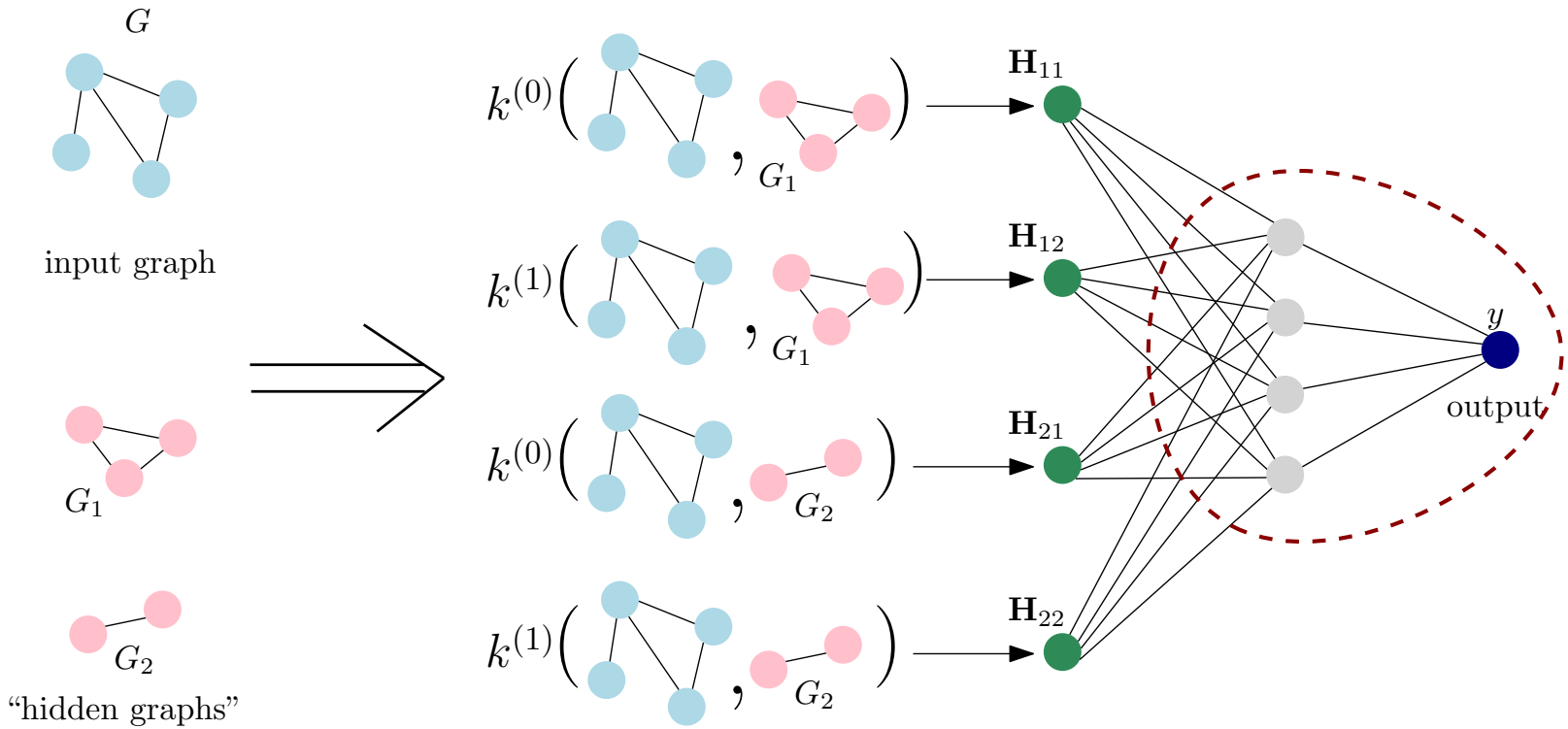
$$k^{(p)}(G, G_{\mathtt{i}}) = \sum_{i=1}^{|V_\times|} \sum_{j=1}^{|V_\times|} \mathsf{s}_i \mathsf{s}_j \left[\mathsf{A}_\times^p\right]_{ij}$$

Random Walk Graph Neural Network (RWNN)



For each input graph G, we build a matrix $\mathsf{H} \in \mathbb{R}^{N \times P+1}$ where $\mathsf{H}_{ij} = k^{(j-1)}(G, G_i)$

Random Walk Graph Neural Network (RWNN)



Matrix H is flattened and fed into a fully-connected neural network to produce the output

Experimental Evaluation - Graph Classification

Models evaluated on standard graph classification datasets from bio/chemo-informatics and social networks

Dataset	ENZYMES	NCI1	PROTEINS	D&D	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	COLLAB
Max # vertices	126	111	620	5,748	136	89	3,782	3,648	492
Min # vertices	2	3	4	30	12	7	6	22	32
Average # vertices	32.63	29.87	39.05	284.32	19.77	13.00	429.61	508.50	74.49
Max # edges	149	119	1,049	14,267	1,249	1,467	4,071	4,783	40,119
Min # edges	1	2	5	63	26	12	4	21	60
Average # edges	62.14	32.30	72.81	715.66	96.53	65.93	497.75	594.87	2,457.34
# labels	3	37	3	82	-	-	-	-	-
# graphs	600	4,110	1,113	1,178	1,000	1,500	2,000	4,999	5,000
# classes	6	2	2	2	2	3	2	5	3

Experimental Setup

Published as a conference paper at ICLR 2020

A Fair Comparison of Graph Neural Networks for Graph Classification

Federico Errica*
Department of Computer Science
University of Pisa
federico.errica@phd.unipi.it

Marco Podda*
Department of Computer Science
University of Pisa
marco.podda@di.unipi.it

Davide Bacciu*
Department of Computer Science
University of Pisa
bacciu@di.unipi.it

Alessio Micheli*
Department of Computer Science
University of Pisa
micheli@di.unipi.it

- 10-fold CV for model assessment and an inner holdout technique with a 90%/10% training/validation split for model selection
- After each model selection $\rightarrow$ train 3 times on the whole training fold, holding out a random fraction (10%) of the data to perform early stopping
- Final test fold score obtained as the mean of these 3 runs

Methods Under Comparison

- Graph Kernels
  - Shortest path kernel (SP) **[Borgwardt and Kriegel, ICDM'05]**
  - Graphlet kernel (GR) **[Shervashidze et al., AISTATS'09]**
  - Weisfeiler-Lehman subtree kernel (WL) **[Shervashidze et al., JMLR'11]**

- Graph Neural Networks
  - DGCNN **[Zhang et al., AAAI'18]**
  - DiffPool **[Ying et al., NeurIPS'18]**
  - ECC **[Simonovsky and Komodakis, CVPR'17]**
  - GIN **[Xu et al., ICLR'19]**
  - GraphSAGE **[Hamilton et al., NeurIPS'17]**
  - RWNN **[Nikolentzos and Vazirgiannis, NeurIPS'20]**
  - Nested GNN **[Zhang and Li, NeurIPS'21]**
  - SPN **[Abboud et al., LOG'22]**
  - GRWNN **[Nikolentzos and Vazirgiannis, AISTATS'23]**
  - WLHN **[Nikolentzos et al., AISTATS'23]**
  - PathNN **[Gaspard et al., ICML'23]**

Graph Classification - Real World Datasets (1/2)

Table: Classification accuracy (± standard deviation) of the different methods on the datasets from the TUDataset collection. Best performance is highlighted in **bold**. OOM means out-of-memory and NA means not available.

	MUTAG	**D&D**	**NCI1**	**PROTEINS**	**ENZYMES**
SP	80.2 (± 6.5)	78.1 (± 4.1)	72.7 (± 1.4)	75.3 (± 3.8)	38.3 (± 8.0)
GR	80.8 (± 6.4)	75.4 (± 3.4)	61.8 (± 1.7)	71.6 (± 3.1)	25.1 (± 4.4)
WL	84.6 (± 8.3)	78.1 (± 2.4)	**84.8** (± 2.5)	73.8 (± 4.4)	50.3 (± 5.7)
DGCNN	84.0 (± 6.7)	76.6 (± 4.3)	76.4 (± 1.7)	72.9 (± 3.5)	38.9 (± 5.7)
DiffPool	79.8 (± 7.1)	75.0 (± 3.5)	76.9 (± 1.9)	73.7 (± 3.5)	59.5 (± 5.6)
ECC	75.4 (± 6.2)	72.6 (± 4.1)	76.2 (± 1.4)	72.3 (± 3.4)	29.5 (± 8.2)
GIN	84.7 (± 6.7)	75.3 (± 2.9)	80.0 (± 1.4)	73.3 (± 4.0)	59.6 (± 4.5)
GraphSAGE	83.6 (± 9.6)	72.9 (± 2.0)	76.0 (± 1.8)	73.0 (± 4.5)	58.2 (± 6.0)
RWNN	**89.2** (± 4.3)	77.6 (± 4.7)	73.9 (± 1.3)	74.7 (± 3.3)	57.6 (± 6.3)
Nested GNN	NA	77.8 (± 3.9)	NA	74.2 (± 3.7)	31.2 (± 6.7)
SPN	NA	77.4 (± 3.8)	80.0 (± 1.5)	74.2 (± 2.7)	69.4 (± 6.2)
GRWNN	83.4 (± 5.6)	75.6 (± 4.6)	67.7 (± 2.2)	74.9 (± 3.5)	**62.7** (± 5.2)
WLHN	86.0 (± 7.4)	**78.5** (± 3.4)	79.2 (± 1.1)	**75.9** (± 1.9)	62.5 (± 5.0)

Graph Classification - Real World Datasets (2/2)

Table: Classification accuracy (± standard deviation) of the different methods on the datasets from the TUDataset collection. Best performance is highlighted in **bold**. OOM means out-of-memory and NA means not available.

	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	COLLAB
SP	57.7 (± 4.1)	39.8 (± 3.7)	89.0 (± 1.0)	51.1 (± 2.2)	**79.9** (± 2.7)
GR	63.3 (± 2.7)	39.6 (± 3.0)	76.6 (± 3.3)	38.1 (± 2.3)	71.1 (± 1.4)
WL	72.8 (± 4.5)	**51.2** (± 6.5)	74.9 (± 1.8)	49.6 (± 2.0)	78.0 (± 2.0)
DGCNN	69.2 (± 3.0)	45.6 (± 3.4)	87.8 (± 2.5)	49.2 (± 1.2)	71.2 (± 1.9)
DiffPool	68.4 (± 3.3)	45.6 (± 3.4)	89.1 (± 1.6)	53.8 (± 1.4)	68.9 (± 2.0)
ECC	67.7 (± 2.8)	43.5 (± 3.1)	OOR	OOR	OOR
GIN	71.2 (± 3.9)	48.5 (± 3.3)	89.9 (± 1.9)	**56.1** (± 1.7)	75.6 (± 2.3)
GraphSAGE	68.8 (± 4.5)	47.6 (± 3.5)	84.3 (± 1.9)	50.0 (± 1.3)	73.9 (± 1.7)
RWNN	70.8 (± 4.8)	48.8 (± 2.9)	90.4 (± 1.9)	53.4 (± 1.6)	71.9 (± 2.5)
GRWNN	72.8 (± 4.2)	49.0 (± 2.9)	90.0 (± 1.8)	54.4 (± 1.7)	72.1 (± 1.9)
WLHN	**73.4** (± 3.7)	49.7 (± 3.6)	**90.7** (± 1.9)	55.2 (± 1.2)	76.2 (± 2.3)

Outline

Sets

What is a set?

A set is a well-defined collection of distinct objects

Complex data sets decomposed into sets of simpler objects

↪ NLP: documents as sets of word embeddings

↪ Graph Mining: graphs as sets of node embeddings

↪ Computer Vision: images as sets of local features

Machine learning on sets has attracted a lot of attention recently

- Set classification
- Set regression

Set Classification

$S_1 = \{1, 4, 2\}$

$y_1 = -1$

$S_2 = \{5, 0, 8, 10\}$

$y_2 = -1$

$S_6 = \{2, 6, 3, 5\}$

$y_6 = ???$

$S_3 = \{3, 7\}$

$y_3 = 1$

$S_4 = \{3, 5, 6\}$

$y_4 = 1$

$S_7 = \{4, 2, 5\}$

$y_7 = ???$

$S_5 = \{5\}$

$y_5 = -1$

- Let $\mathcal{X}$ be a set
- Input data $S \in 2^{\mathcal{X}}$
- Output $y \in \{-1, 1\}$
- Training set $\{(S_1, y_1), \ldots, (S_n, y_n)\}$
- Goal: estimate a function $f : 2^{\mathcal{X}} \rightarrow \in \{-1, 1\}$ to predict y from $f(S)$

Limitations of Standard Machine Learning Models

Conventional machine learning models cannot handle sets:

- expect fixed dimensional data instances
  $\hookrightarrow$ sets allowed to vary in the number of elements
- not invariant to permutations of features
  - a learning algorithm for sets needs to produce identical representations for any permutation of the elements of an input set
  - for instance, a model f needs to satisfy the following for any permutation π of the set's elements:

$$f(\{x_1, \ldots, x_M\}) = f(\{x_{\pi(1)}, \ldots, x_{\pi(M)}\})$$

Outline

Neural Networks for Sets

Recent approaches:

- unordered sets $\rightarrow$ ordered sequences $\rightarrow$ RNN **[Vinyals et al., ICLR'16]**
- DeepSets **[Zaheer et al., NIPS'17]** and PointNet **[Qi et al., CVPR'17]** transform the vectors of the sets into new representations, then apply permutation-invariant functions
- PointNet++ **[Qi et al., NIPS'17]** and SO-Net **[Li et al., CVPR'18]** apply PointNet hierarchically in order to better capture local structures
- Set Transformer **[Lee et al., ICML'19]**, a neural network that uses self-attention to model interactions among the elements of the input set
- RepSet **[Skianis et al., AISTATS'20]**, a neural network that generates representations for sets by comparing them against some trainable sets

SOTA in supervised learning tasks:

- regression: population statistic estimation, sum of digits
- classification: point cloud classification, outlier detection

DeepSets

Theorem (Zaheer et al., NIPS'17)

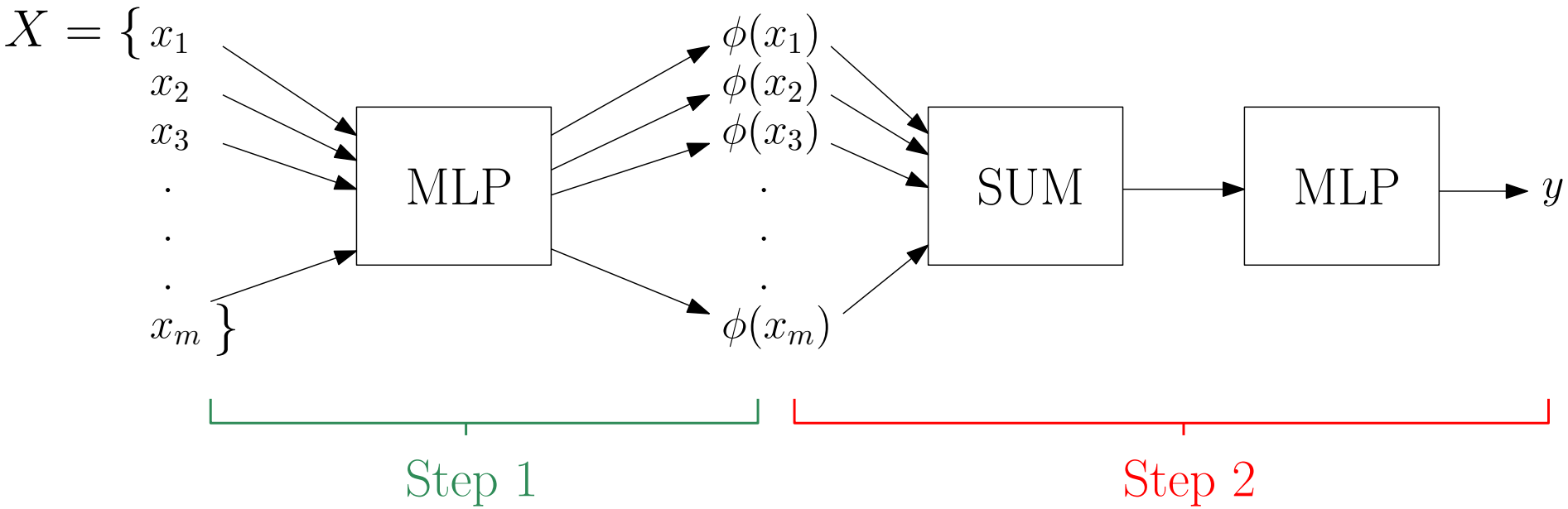
If $\mathfrak{X}$ is a countable set and $\mathcal{Y} = \mathbb{R}$, then a function $f(X)$ operating on a set X having elements from $\mathfrak{X}$ is a valid set function, i.e., invariant to the permutation of instances in X, if and only if it can be decomposed in the form $\rho(\sum_{x \in X} \phi(x))$, for suitable transformations ϕ and ρ.

DeepSets achieves permutation invariance by replacing ϕ and ρ with multi-layer perceptrons (universal approximators)

DeepSets consist of the following two steps:

1. Each element x_i of each set is transformed (possibly by several layers) into some representation $\phi(x_i)$
2. The representations $\phi(x_i)$ are added up and the output is processed using the ρ network in the same manner as in any deep network (e.g., fully connected layers, nonlinearities, etc.)

DeepSets



Step 1: The elements $x_1, \ldots, x_m$ of the input set X are transformed into representations $\phi(x_1), \ldots, \phi(x_m)$

Step 2: A representation for the entire set is produced as $z_X = \phi(x_1) + \ldots + \phi(x_m)$ and is also transformed as follows $y = \rho(z_X)$ to produce the output

Experiments - Point Cloud Classification

Objective: classify point-clouds
$\hookrightarrow$ point-clouds are sets of low-dimensional vectors (typically 3-dimensional vectors representing the x, y, z-coordinates of objects)

Dataset: ModelNet40 $\rightarrow$ consists of 3-dimensional representations of $9,843$ training and $2,468$ test instances belonging to 40 classes of objects

Setup: point-clouds directly passed on to DeepSets

Model	Instance Size	Representation	Accuracy
3DShapeNets [25]	30^3	voxels (using convolutional deep belief net)	77%
VoxNet [26]	32^3	voxels (voxels from point-cloud + 3D CNN)	83.10%
MVCNN [21]	$164\times164\times12$	multi-vew images (2D CNN + view-pooling)	90.1%
VRN Ensemble [27]	32^3	voxels (3D CNN, variational autoencoder)	95.54%
3D GAN [28]	64^3	voxels (3D CNN, generative adversarial training)	83.3%
DeepSets	$\mathbf{5000\times3}$	point-cloud	$90\pm.3\%$
DeepSets	$\mathbf{100\times3}$	point-cloud	$82\pm2\%$

Experiments - Text Concept Retrieval

Objective: retrieve words belonging to a "concept" given few words from the concept

Example: given the set of words $\{tiger, lion, cheetah\}$, retrieve other related words like jaguar and puma, which all belong to the concept of big cats

Setup: query word added to set and new set fed to DeepSet which produces a score

Method	LDA-1k (Vocab = 17k)					LDA-3k (Vocab = 38k)					LDA-5k (Vocab = 61k)				
	Recall (%)			MRR	Med.	Recall (%)			MRR	Med.	Recall (%)			MRR	Med.
	@10	@100	@1k			@10	@100	@1k			@10	@100	@1k		
Random	0.06	0.6	5.9	0.001	8520	0.02	0.2	2.6	0.000	28635	0.01	0.2	1.6	0.000	30600
Bayes Set	1.69	11.9	37.2	0.007	2848	2.01	14.5	36.5	0.008	3234	1.75	12.5	34.5	0.007	3590
w2v Near	**6.00**	**28.1**	**54.7**	0.021	**641**	4.80	21.2	43.2	0.016	2054	4.03	16.7	35.2	0.013	6900
NN-max	4.78	22.5	53.1	0.023	779	5.30	24.9	54.8	0.025	672	4.72	21.4	47.0	0.022	1320
NN-sum-con	4.58	19.8	48.5	0.021	1110	5.81	27.2	60.0	**0.027**	453	4.87	23.5	53.9	0.022	731
NN-max-con	3.36	16.9	46.6	0.018	1250	5.61	25.7	57.5	0.026	570	4.72	22.0	51.8	0.022	877
DeepSets	5.53	24.2	54.3	**0.025**	696	**6.04**	**28.5**	**60.7**	**0.027**	**426**	**5.54**	**26.1**	**55.5**	**0.026**	**616**

Experiments - Image Tagging

Objective: retrieve all relevant tags corresponding to an image

Setup: features of the image are concatenated to the embeddings of the tags, and then the whole set is passed on to DeepSets to assign a single score to the set

Method	ESP game				IAPRTC-12.5			
	P	R	F1	N+	P	R	F1	N+
Least Sq.	35	19	25	215	40	19	26	198
MBRM	18	19	18	209	24	23	23	223
JEC	24	19	21	222	29	19	23	211
FastTag	46	22	30	**247**	**47**	26	34	**280**
Least Sq.(D)	**44**	32	**37**	232	46	30	36	218
FastTag(D)	**44**	32	**37**	229	46	**33**	**38**	254
DeepSets	39	**34**	36	246	42	31	36	247

Experiments - Set Anomaly Detection

Objective: find the anomalous face in each set

Architecture: consists of 9 2d-convolution and max-pooling layers followed by the DeepSets model, and a softmax layer that assigns a probability value to each set member

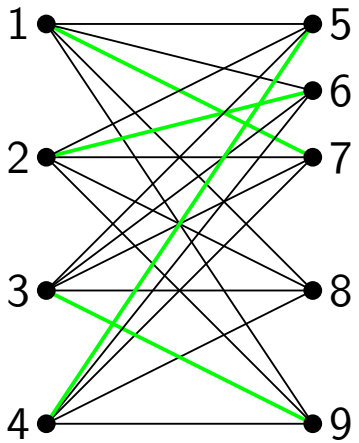
RepSet

- A permutation invariant neural network for sets
- Generates a number of "hidden sets" and it compares the input set with these sets using a network flow algorithm (e.g., bipartite matching)
- The outputs of the network flow algorithm form the penultimate layer and are fed to a fully-connected layer which produces the output
- The model is end-to-end trainable $\rightarrow$ "hidden sets" are updated during training
- For large sets, solving the flow problems can become prohibitive ☹
  $\hookrightarrow$ ApproxRepSet is a relaxed formulation (also permutation invariant) that scales to very large datasets

Permutation Invariant Layer

- A layer whose output is the same regardless of the ordering of the input's elements
- Contains m "hidden sets" $Y_1, Y_2, \ldots, Y_m$ of d-dimensional vectors
  $\hookrightarrow$ may have different cardinalities and their components are trainable
- Measure similarity between input set and each one of the "hidden sets" by comparing their building blocks, i.e., their elements $\rightarrow$ bipartite matching

1	1.76	0.40	0.97
2	2.24	1.86	−0.97
3	0.95	−0.15	−0.10
4	0.41	0.14	1.45



5	0.52	0.08	1.62
6	2.14	1.72	−1.05
7	1.55	−0.45	0.88
8	−0.34	−1.26	0.24
9	1.08	−0.21	−0.09

Figure: Example of a bipartite graph generated from 2 sets of 3-dimensional vectors, and of its maximum matching M. Green color indicates that an edge belongs to M

RepSet - Bipartite Matching Problem

- Input set $X = \{v_1, v_2, \ldots, v_{|X|}\}$ where $v_1, \ldots, v_{|X|}$ vectors
- "Hidden set" $Y = \{u_1, u_2, \ldots, u_{|Y|}\}$
- Maximum matching between the elements of X and Y by solving the following linear program:

$$\max \sum_{i=1}^{|X|} \sum_{j=1}^{|Y|} z_{ij} f(v_i, u_j) \text{ subject to:}$$

$$\sum_{i=1}^{|X|} z_{ij} \leq 1 \quad \forall j \in \{1, \ldots, |Y|\}$$

$$\sum_{j=1}^{|Y|} z_{ij} \leq 1 \quad \forall i \in \{1, \ldots, |X|\}$$

$$z_{ij} \geq 0 \quad \forall i \in \{1, \ldots, |X|\}, \forall j \in \{1, \ldots, |Y|\}$$

where $f(v_i, u_j)$ is a differentiable function (e.g., inner product), and $z_{ij} = 1$ if component i of X is assigned to component j of Y_i, and 0 otherwise

RepSet - Produced Set Representations

Given an input set X and the m "hidden sets" $Y_1, Y_2, \ldots, Y_m$

1. formulate m different bipartite matching problems
2. by solving all m problems, end up with an m-dimensional vector $\mathbf{x} \rightarrow$ hidden representation of set X
3. this m-dimensional vector can be used as features for different machine learning tasks (e.g., set regression, set classification)
   $\hookrightarrow$ For instance, in the case of a set classification problem with $|\mathcal{C}|$classes, output is computed as

$$\mathrm{p} = \mathrm{softmax}(\mathrm{W}\mathrm{x} + \mathrm{b})$$

   where W is a matrix of trainable parameters and b is the bias term

RepSet - Architecture

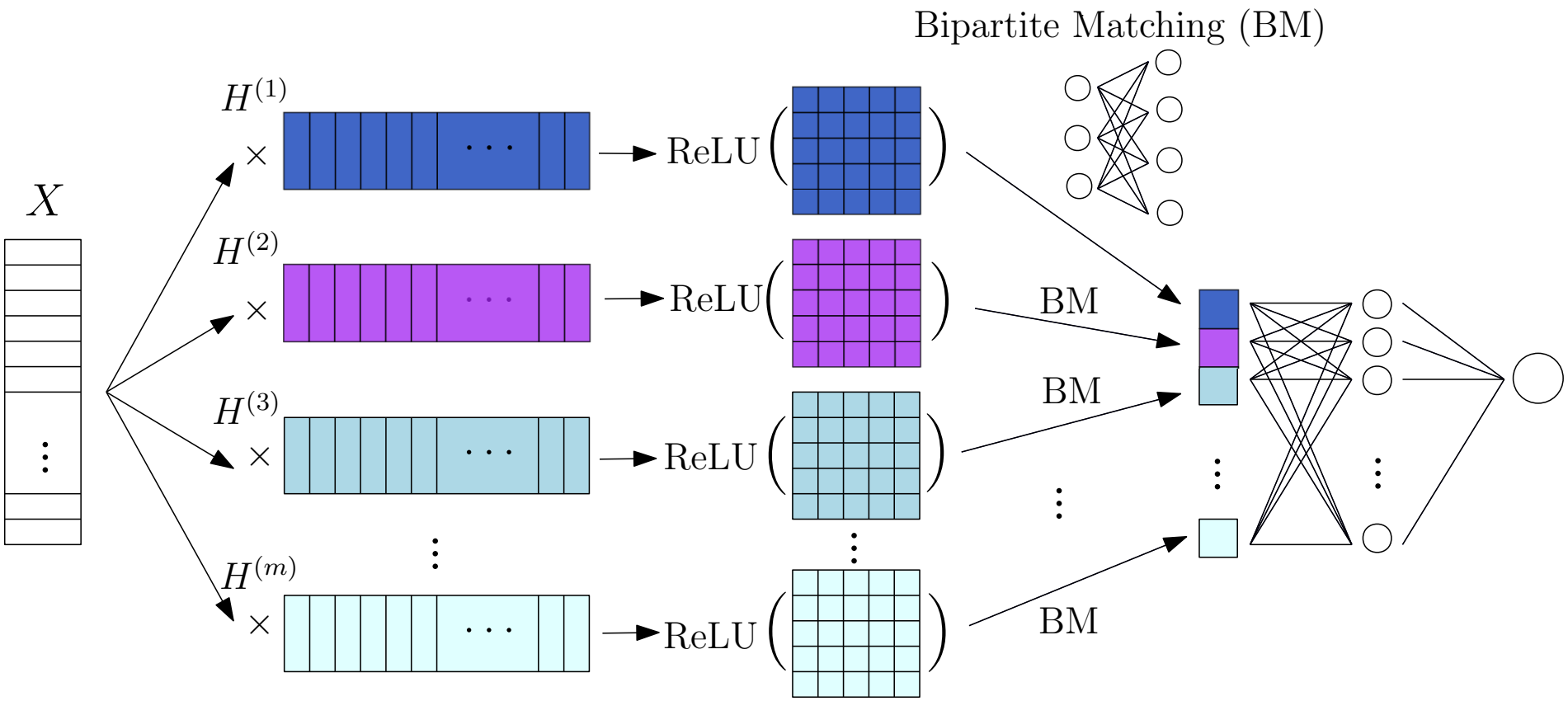


Figure: Each element of the input set is compared with the elements of all "hidden sets", and the emerging matrices serve as the input to bipartite matching. The values of the BM problems correspond to the representation of the input set.

RepSet - Relaxed Variant (ApproxRepSet)

- Input set $X = \{v_1, v_2, \ldots, v_{|X|}\}$ where $v_1, \ldots, v_{|X|}$ vectors
- "Hidden set" $Y = \{u_1, u_2, \ldots, u_{|Y|}\}$
- Identify which of the two sets has the highest cardinality.
- If $|X| \geq |Y|$, we solve the following problem:

$$\max \sum_{i=1}^{|X|} \sum_{j=1}^{|Y|} z_{ij} f(v_i, u_j) \text{ subject to:}$$

$$\sum_{i=1}^{|X|} z_{ij} \leq 1 \quad \forall j \in \{1, \ldots, |Y|\}$$

$$z_{ij} \geq 0 \quad \forall i \in \{1, \ldots, |X|\}, \forall j \in \{1, \ldots, |Y|\}$$

- Multiple elements of X (the bigger set) can be matched with the same element of Y
- Optimal solution matches an element y_j of Y with x_i of X if $f(v_i, u_j)$ is positive and $f(v_i, u_j) = \max_k f(v_k, u_j)$

Experimental Evaluation

- Text categorization
  - given a document, the input to the model is the set of embeddings of its terms
  - standard text categorization datasets (TWITTER, BBCSPORT etc.)

- Graph classification
  - represent each graph as a set of vectors (i.e., the embeddings of its nodes)
  - node embeddings are extracted by struc2vec **[Ribeiro et al., KDD'17]**
  - datasets derived from bioinformatics (MUTAG, PROTEINS) and social networks (IMDB-BINARY, -MULTI, REDDIT-BINARY)

Experiments - Text classification

	BBCSPORT	TWITTER	RECIPE	OHSUMED	CLASSIC	REUTERS	AMAZON	20NG
WMD	4.60 ± 0.70	28.70 ± 0.60	42.60 ± 0.30	44.50	**2.88** ± 0.10	3.50	7.40 ± 0.30	26.80
S-WMD	2.10 ± 0.50	27.50 ± 0.50	39.20 ± 0.30	34.30	3.20 ± 0.20	3.20	5.80 ± 0.10	26.80
DeepSets	25.45 ± 20.1	29.66 ± 1.62	70.25 ± 0.00	71.53	5.95 ± 1.50	10.00	8.58 ± 0.67	38.88
NN-mean	10.09 ± 2.62	31.56 ± 1.53	64.30 ± 7.30	45.37	5.35 ± 0.75	11.37	13.66 ± 3.16	38.40
NN-max	2.18 ± 1.75	30.27 ± 1.26	43.47 ± 1.05	35.88	4.21 ± 0.11	4.33	7.55 ± 0.63	32.15
NN-attention	4.72 ± 0.97	29.09 ± 0.62	43.18 ± 1.22	**31.36**	4.42 ± 0.73	3.97	6.92 ± 0.51	28.73
Set-Transformer	4.18 ± 1.23	27.79 ± 0.47	42.54 ± 1.35	35.68	5.23 ± 0.52	4.52	7.18 ± 0.44	30.01
RepSet	**2.00** ± 0.89	**25.42** ± 1.10	**38.57** ± 0.83	33.88	3.38 ± 0.50	3.15	**5.29** ± 0.28	**22.98**
ApproxRepSet	4.27 ± 1.73	27.40 ± 1.95	40.94 ± 0.40	35.94	3.76 ± 0.45	**2.83**	5.69 ± 0.40	23.82

Table: Classification test error of the proposed architecture and baselines on 8 TC datasets.

Hidden set	Terms similar to elements of hidden sets	Terms similar to centroids of hidden sets
1	chelsea, football, striker, club, champions	footballing
2	qualify, madrid, arsenal, striker, united, france	ARSENAL_Wenger
3	olympic, athlete, olympics, sport, pentathlon	Olympic_Medalist
4	penalty, cup, rugby, coach, goal	rugby
5	match, playing, batsman, batting, striker	batsman

Table: Terms of the employed pre-trained model that are most similar to the elements and centroids of 5 hidden sets.

Experiments - Graph Classification

	MUTAG	PROTEINS	IMDB BINARY	IMDB MULTI	REDDIT BINARY
PSCN $k = 10$	88.95 ($\pm$ 4.37)	75.00 ($\pm$ 2.51)	71.00 ($\pm$ 2.29)	45.23 ($\pm$ 2.84)	86.30 ($\pm$ 1.58)
Deep GR	82.66 ($\pm$ 1.45)	71.68 ($\pm$ 0.50)	66.96 ($\pm$ 0.56)	44.55 ($\pm$ 0.52)	78.04 ($\pm$ 0.39)
EMD	86.11 ($\pm$ 0.84)	-	-	-	-
DGCNN	85.80 ($\pm$ 1.70)	75.50 ($\pm$ 0.90)	70.03 ($\pm$ 0.86)	47.83 ($\pm$ 0.85)	-
SAEN	84.99 ($\pm$ 1.82)	75.31 ($\pm$ 0.70)	71.59 ($\pm$ 1.20)	48.53 ($\pm$ 0.76)	87.22 ($\pm$ 0.80)
RetGK	**90.30** ($\pm$ 1.10)	76.20 ($\pm$ 0.50)	72.30 ($\pm$ 0.60)	48.70 ($\pm$ 0.60)	**92.60** ($\pm$ 0.30)
DiffPool	-	**76.25**	-	-	-
DeepSets	86.26 ($\pm$ 1.09)	60.82 ($\pm$ 0.79)	69.84 ($\pm$ 0.64)	47.62 ($\pm$ 1.18)	52.01 ($\pm$ 1.47)
NN-mean	87.55 ($\pm$ 0.98)	73.00 ($\pm$ 1.21)	71.48 ($\pm$ 0.48)	49.92 ($\pm$ 0.82)	84.57 ($\pm$ 0.84)
NN-max	85.84 ($\pm$ 0.99)	71.05 ($\pm$ 0.54)	69.56 ($\pm$ 0.91)	48.28 ($\pm$ 0.43)	80.98 ($\pm$ 0.79)
NN-attention	85.92 ($\pm$ 1.16)	74.48 ($\pm$ 0.22)	**72.40** ($\pm$ 0.45)	49.56 ($\pm$ 0.47)	88.74 ($\pm$ 0.53)
Set-Transformer	87.71 ($\pm$ 1.14)	59.62 ($\pm$ 1.42)	71.21 ($\pm$ 1.28)	**50.25** ($\pm$ 0.74)	83.79 ($\pm$) 0.83
RepSet	88.63 ($\pm$ 0.86)	73.04 ($\pm$ 0.42)	**72.40** ($\pm$ 0.73)	49.93 ($\pm$ 0.60)	87.45 ($\pm$ 0.86)
ApproxRepSet	86.33 ($\pm$ 1.48)	70.74 ($\pm$ 0.85)	71.46 ($\pm$ 0.91)	48.92 ($\pm$ 0.28)	80.30 ($\pm$ 0.56)

Table: Classification accuracy ($\pm$ standard deviation) of the proposed architecture and the baselines on the 5 graph classification datasets.

THANK YOU !

Acknowledgements
G. Salha

http://www.lix.polytechnique.fr/dascim/