Deep Learning for Graphs and Sets with GNNs

M. Vazirgiannis

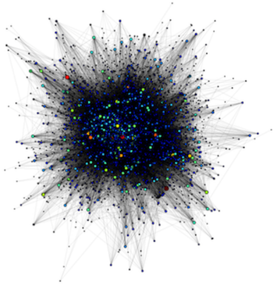
Data Science and Mining Team (DASCIM), LIX
École Polytechnique, Institute Polytechnique de Paris
http://www.lix.polytechnique.fr/dascim
Google Scholar: https://bit.ly/2rwmvQU
Twitter: @mvazirg

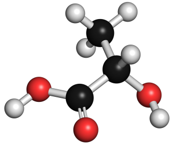
June 2024

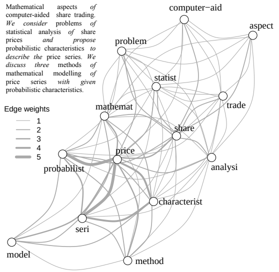
Outline

Outline

Graphs Are Everywhere






Mathematical aspects of computer-aided share trading. *We consider* problems *of* statistical analysis *of* share prices *and propose* probabilistic characteristics *to describe the* price series. *We discuss three* methods *of* mathematical modelling *of* price series *with given* probabilistic characteristics.

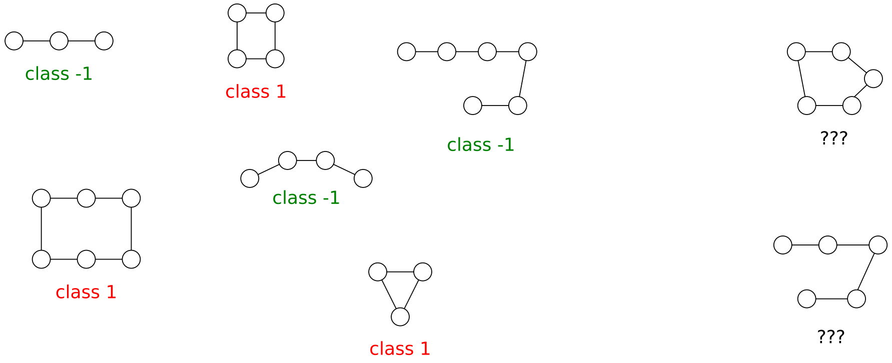
Why graphs?

Machine Learning on Graphs

Machine learning tasks on graphs:

- Node classification: given a graph with labels on some nodes, provide a high quality labeling for the rest of the nodes
- Graph clustering: given a graph, group its vertices into clusters taking into account its edge structure in such a way that there are many edges within each cluster and relatively few between the clusters
- Link Prediction: given a pair of vertices, predict if they should be linked with an edge
- **Graph classification**: given a set of graphs with known class labels for some of them, decide to which class the rest of the graphs belong

Graph Classification



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

Graph Comparison

Definition (Graph Comparison Problem)

Given two graphs G_1 and G_2 from the space of graphs $\mathcal{G}$, the problem of graph comparison is to find a mapping

$$s : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$$

such that $s(G_1, G_2)$ quantifies the similarity of G_1 and G_2.

Graph comparison is a topic of high significance

- It is the central problem for all learning tasks on graphs such as clustering and classification
- Most machine learning algorithms make decisions based on the similarities or distances between pairs of instances (e.g. k-nn)

Not an Easy Problem

Although graph comparison seems a tractable problem, it is very complex

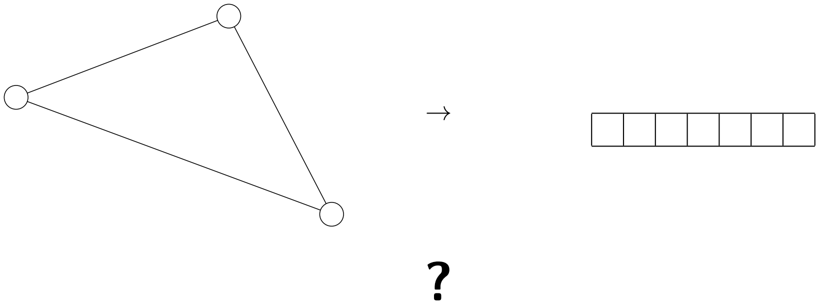
Many problems related to it are **NP-complete**

- subgraph isomorphism
- finding largest common subgraph

We are interested in algorithms capable of measuring the similarity between two graphs in **polynomial** time

Graphs to Vectors

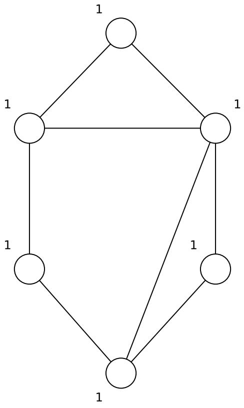
- To analyze and extract knowledge from graphs, one needs to perform machine learning tasks
- Most machine learning algorithms require the input to be represented as a fixed-length feature vector
- There is no straightforward way to transform graphs to such a representation



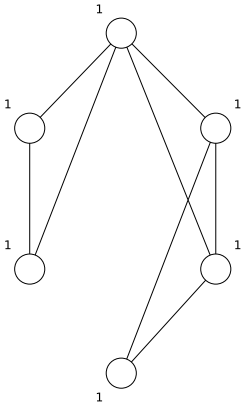
Weisfeiler-Lehman Test of Isomorphism

May answer if two graphs are not isomorphic

Run the Weisfeiler-Lehman algorithm for the following pair of graphs



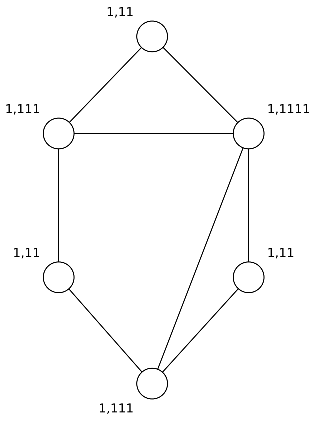
G_1



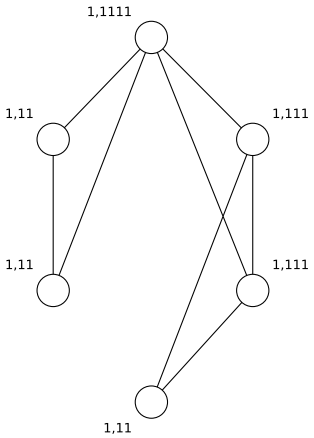
G_2

Iteration 1

First step: Augment the labels of the vertices by the sorted set of labels of neighbouring vertices



G_1

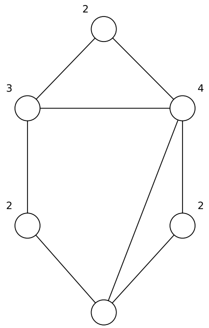


G_2

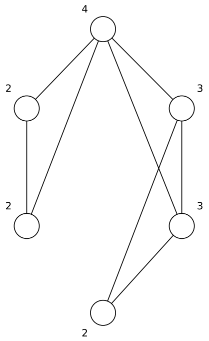
Iteration 1

Second step: Compress the augmented labels into new, short labels:

- $1, 11 \rightarrow 2$
- $1, 111 \rightarrow 3$
- $1, 1111 \rightarrow 4$



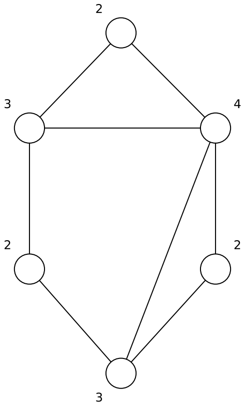
G_1



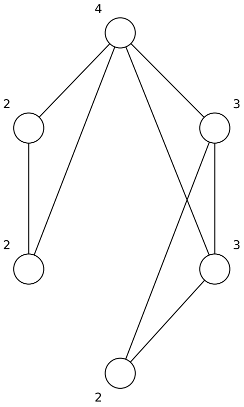
G_2

Iteration 1

Are the label sets of G_1 and G_2 identical?



G_1



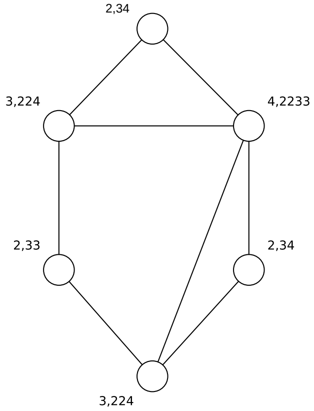
G_2

Yes!!!

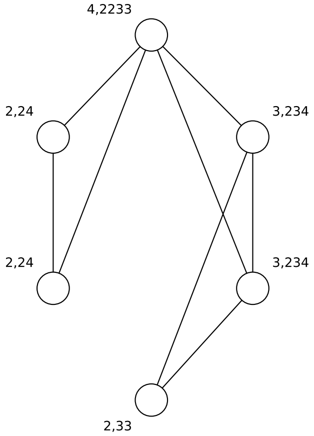
Continue to the next iteration

Iteration 2

First step: Augment the labels of the vertices by the sorted set of labels of neighbouring vertices



G_1

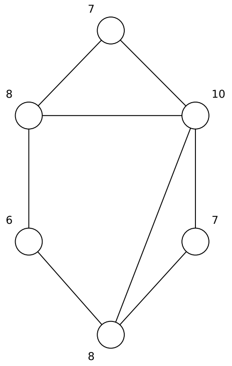


G_2

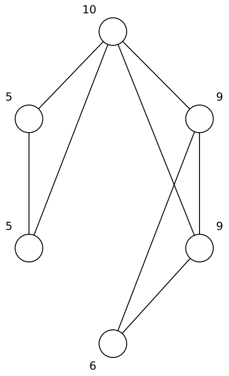
Iteration 2

Second step: Compress the augmented labels into new, short labels:

- $2, 24 \rightarrow 5$
- $2, 33 \rightarrow 6$
- $2, 34 \rightarrow 7$
- $3, 234 \rightarrow 9$
- $4, 2233 \rightarrow 10$



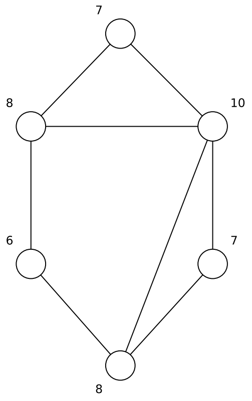
G_1



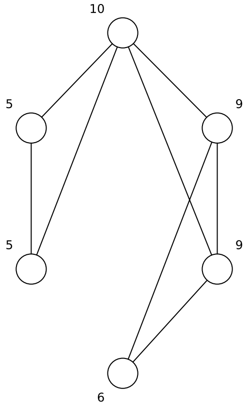
G_2

Iteration 2

Are the label sets of G_1 and G_2 identical?



G_1



G_2

No!!!

Graphs are not isomorphic

Weisfeiler-Lehman Framework

Let $G^1, G^2, \ldots, G^h$ be the graphs emerging from graph G at the iteration $1, 2, \ldots, h$ of the Weisfeiler-Lehman algorithm

Then, the Weisfeiler-Lehman kernel is defined as:

$$k_{WL}^h(G_1, G_2) = k(G_1, G_2) + k(G_1^1, G_2^1) + k(G_1^2, G_2^2) + \ldots + k(G_1^h, G_2^h)$$

where $k(\cdot, \cdot)$ is a base kernel (e.g. subtree kernel, shortest path kernel, ...)

At each iteration of the Weisfeiler-Lehman algorithm:

- run a graph kernel for labeled graphs
- the new kernel values are added to the ones of the previous iteration

[Shervashidze et al., JMLR 12.Sep (2011)]

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 $\mathbf{h}_v^t$ based on its previous representation and the representations of its neighbors:

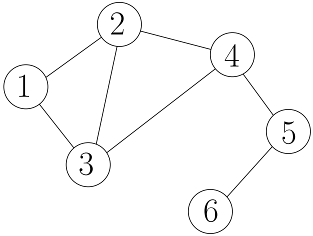
$$\mathbf{m}_v^{(t+1)} = \mathrm{AGGREGATE}\Big(\Big\{\mathbf{h}_u^{(t)} \Big| u \in \mathcal{N}(v)\Big\}\Big)$$
$$\mathbf{h}_v^{(t+1)} = \mathrm{COMBINE}\Big(\mathbf{h}_v^{(t)}, \mathbf{m}_v^{(t+1)}\Big)$$

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

$$\mathbf{h}_1^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_1^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_2^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_3^{(t)})$$
$$\mathbf{h}_2^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_2^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_1^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_3^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_4^{(t)})$$
$$\mathbf{h}_3^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_3^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_1^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_2^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_4^{(t)})$$
$$\mathbf{h}_4^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_4^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_2^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_3^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_5^{(t)})$$
$$\mathbf{h}_5^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_5^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_4^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_6^{(t)})$$
$$\mathbf{h}_6^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_6^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_5^{(t)})$$



Remark: Biases are omitted for clarity

Graph Convolutional Network (GCN)

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

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

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

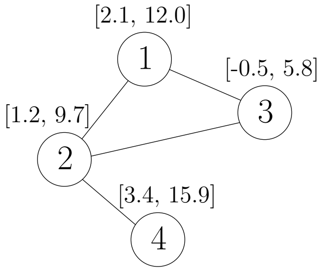
In matrix form, the above is equivalent to:

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

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

[Kipf and Welling, ICLR'17]

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



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

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

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

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \qquad \tilde{\mathbf{D}} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

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

And then matrix $\hat{\mathbf{A}}$:

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

The parameters of the message passing layer are initialised as follows:

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

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

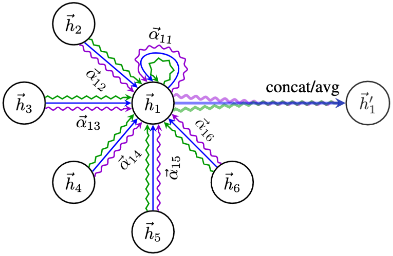
$$\mathbf{H} = \mathrm{ReLU}\Big(\hat{\mathbf{A}}\big(\mathbf{XW} + \mathbf{b}\big)\Big) = \begin{bmatrix} 0 & 5.024 & 9.466 \\ 0 & 7.859 & 13.588 \\ 0 & 5.024 & 9.466 \\ 0 & 6.971 & 11.281 \end{bmatrix}$$

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:

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

where $[\cdot||\cdot]$ denotes concatenation of two vectors and $\mathbf{a}$ is a trainable vector (context)



Graph Attention Network (GAT)

Then the representations of the nodes are updated as follows:

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

In matrix form, the above is equivalent to:

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

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

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

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

where $[\alpha_k^{(t)}]_{ij}$ are the attention coefficients computed by the k^{th} attention mechanism, and $\mathbf{W}_k^{(t)}$ 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:

$$\begin{aligned}
\mathbf{m}_v^{(t)} &= \text{AGGREGATE}^{(t)}\left(\left\{\mathbf{h}_u^{(t)} \middle| u \in \mathcal{N}^k(v)\right\}\right) \\
\mathbf{h}_v^{(t+1)} &= \sigma\left(\mathbf{W}^{(t)}\left[\mathbf{h}_v^{(t)}||\mathbf{m}_v^{(t)}\right]\right) \\
\mathbf{h}_v^{(t+1)} &= \frac{\mathbf{h}_v^{(t+1)}}{\left|\left|\mathbf{h}_v^{(t+1)}\right|\right|_2}
\end{aligned}$$

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):

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

   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:

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

   where max denotes the elementwise max operator

Skip Connections

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

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 $\mathbf{h}_v = [\mathbf{h}_v^{(1)} || \mathbf{h}_v^{(2)} || \dots || \mathbf{h}_v^{(T)}]$

[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

$\vdots$

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

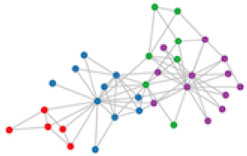
- input: $n \times n$ adjacency matrix $\mathbf{A}$ and (potentially) an $n \times d$ node features matrix $\mathbf{X}$, stacking-up d-dimensional vectors associated to each node
- objective: derive an $n \times d$ latent representation matrix $\mathbf{Z}$ (*encoding step*) from which we can reconstruct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

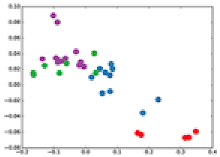
Auto-encoders are i. unsupervised since they reconstruct the raw input data but also ii supervised due to the knowledge from the adjacency matrix

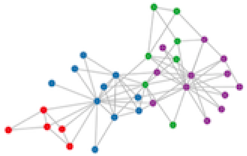
Graph Autoencoders (GAE)

$$\begin{tikzcd} A \arrow[r] & \boxed{\text{Encoder}} \arrow[r] & Z \arrow[r] & \boxed{\text{Decoder}} \arrow[r] & \hat{A} \end{tikzcd}$$

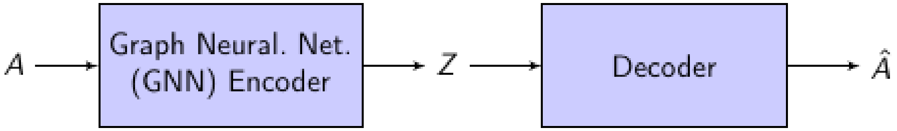
$$A = \begin{pmatrix} 0 & 1 & 0 & \dots & 1 \\ 1 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 0.523 & -1.012 \\ 2.127 & 0.316 \\ 0.912 & 0.127 \\ \dots & \dots \\ -1.210 & 0.026 \end{pmatrix} \qquad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







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:** $\boxed{\mathbf{Z} = \mathrm{GNN}(\mathbf{A}, \mathbf{X})}$
2. **decoder:** $\boxed{\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^\top)}$
   i.e., for all node pairs (i, j), we have $\hat{\mathbf{A}}_{ij} = \sigma(\mathbf{z}_i^\top \mathbf{z}_j)$

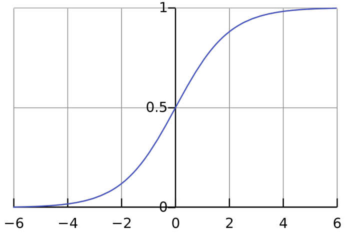


Figure: Sigmoid activation: $\sigma(x) = \frac{1}{1+e^{-x}}$

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

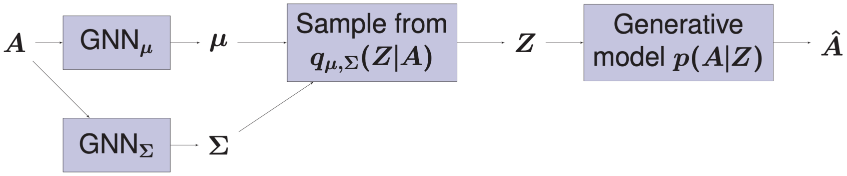
- e.g., **cross-entropy loss**: $-\sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{A}_{ij}\log(\hat{\mathbf{A}}_{ij}) + (1-\mathbf{A}_{ij})\log(1-\hat{\mathbf{A}}_{ij}))$
- or **MSE** loss: $\sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{A}_{ij} - \hat{\mathbf{A}}_{ij})^2$

[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**):

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

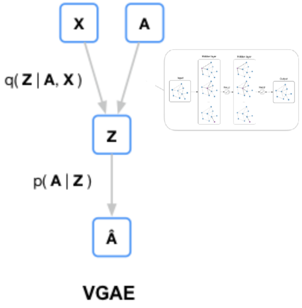
[Kipf and Welling, Bayesian Deep Learning Workshop'16]

Graph Variational Autoencoders (GVAE)

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

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

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



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

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

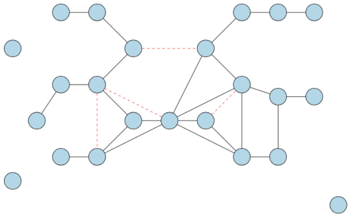
Applications of Graph Autoencoders

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

- link prediction
- node clustering
- recommendation

  ⋮

Recall: Link Prediction

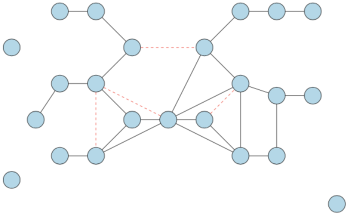


Test set:

- 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 with GVAE - Cora Graph

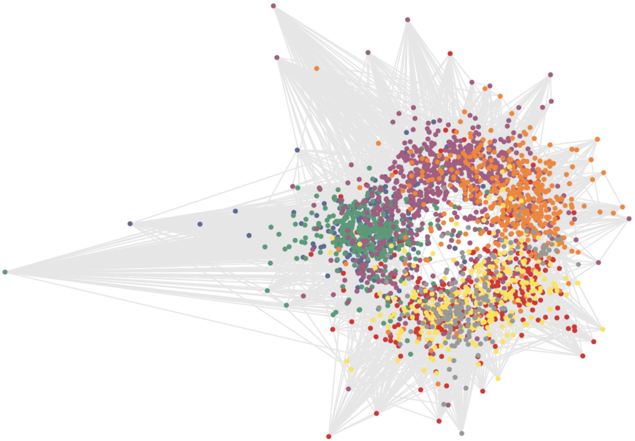


Figure: Projection of latent space representations, from Graph VAE model trained on Cora citation network. Colors denote document classes i.e. node labels (not provided during training)

Graph Autoencoders for Recommendation

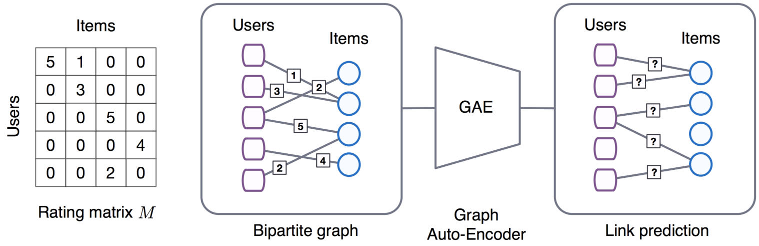
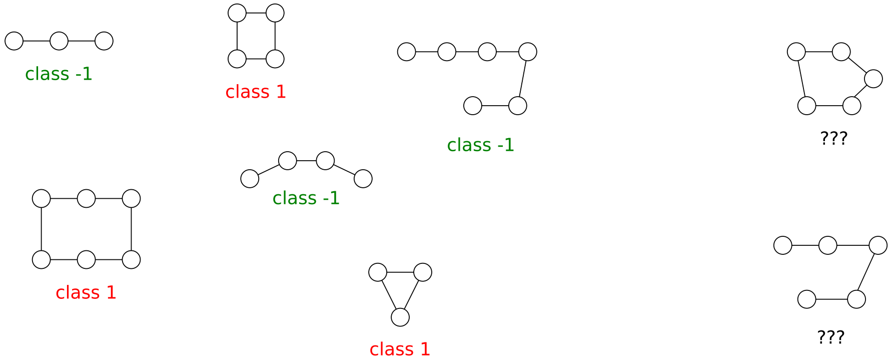


Figure: Using GAE for Matrix Completion and Recommendation

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

Outline

Graph Classification

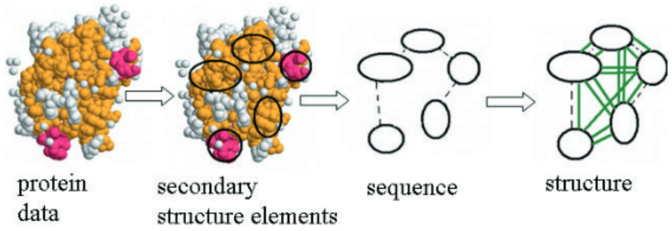


- Input data $G \in \mathcal{G}$
- Output $y \in \{-1, 1\}$
- Training set $\mathcal{S} = \{(G_1, y_1), \ldots, (G_n, y_n)\}$
- Goal: estimate a function $f : \mathcal{G} \rightarrow \in \{-1, 1\}$ to predict y from $f(G)$

Motivation - Protein Function Prediction

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

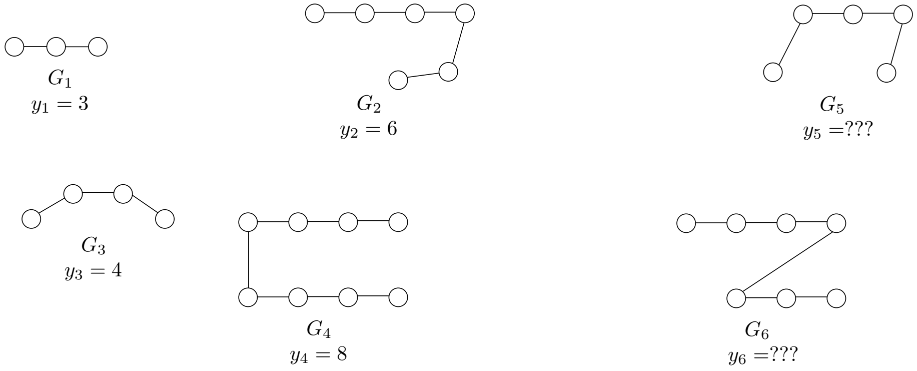
- structure
- sequence
- chemical properties



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

[Borgwardt et al., Bioinformatics 21]

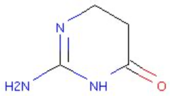
Graph Regression



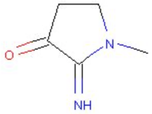
- Input data $G \in \mathcal{G}$
- Output $y \in \mathbb{R}$
- Training set $\mathcal{S} = \{(G_1, y_1), \ldots, (G_n, y_n)\}$
- Goal: estimate a function $f : \mathcal{G} \rightarrow \mathbb{R}$ to predict y from $f(G)$

Motivation - Molecular Property Prediction

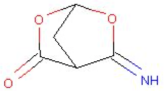
12 targets corresponding to molecular properties: ['mu', 'alpha', 'HOMO', 'LUMO', 'gap', 'R2', 'ZPVE', 'U0', 'U', 'H', 'G', 'Cv']



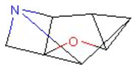
SMILES: NC1=NCCC(=O)N1
Targets: [2.54 64.1 -0.236 -2.79e-03 2.34e-01 900.7 0.12 -396.0 -396.0 -396.0 -396.0 26.9]



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]

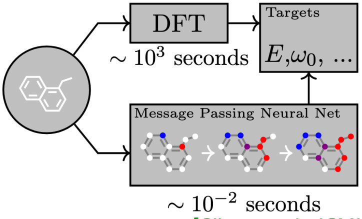


SMILES: N=C1OC2CC1C(=O)O2
Targets: [4.274 61.94 -0.282 -0.026 0.256 887.402 0.104 -473.876 -473.87 -473.869 -473.907 24.823]



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 $\mathbf{h}_v^t$ based on its previous representation and the representations of its neighbors:

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

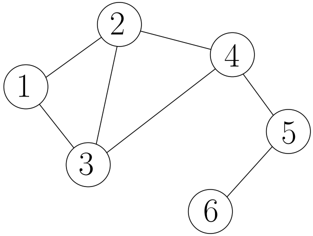
where $\mathcal{N}(v)$ is the set of neighbors of v, and AGGREGATE and COMBINE are message functions and vertex update functions respectively

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

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

Example of Message Passing Layer

$$\mathbf{h}_1^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_1^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_2^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_3^{(t)})$$
$$\mathbf{h}_2^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_2^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_1^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_3^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_4^{(t)})$$
$$\mathbf{h}_3^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_3^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_1^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_2^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_4^{(t)})$$
$$\mathbf{h}_4^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_4^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_2^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_3^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_5^{(t)})$$
$$\mathbf{h}_5^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_5^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_4^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_6^{(t)})$$
$$\mathbf{h}_6^{(t+1)} = f(\mathbf{W}_0^{(t)}\mathbf{h}_6^{(t)} + \mathbf{W}_1^{(t)}\mathbf{h}_5^{(t)})$$

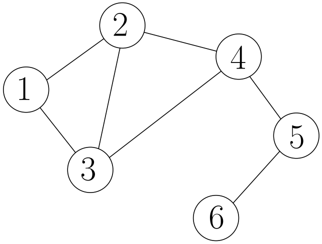


Remark: Biases are omitted for clarity

Readout Step Example

Output of message passing phase:

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



Graph representation:

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

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:
$\mathbf{h}_1^{(T)} = \begin{bmatrix} 1.2 & 1.4 & -1.0 \end{bmatrix}$ $\mathbf{h}_2^{(T)} = \begin{bmatrix} -2.4 & -0.6 & 1.3 \end{bmatrix}$
$\mathbf{h}_3^{(T)} = \begin{bmatrix} 1.5 & 1.3 & -0.9 \end{bmatrix}$

Then, we can produce graph representations as follows:

- sum aggerator:

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

- mean aggerator:

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

- max aggerator:

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

Convolutional Networks for Learning Molecular Fingerprints

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

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

$$\mathbf{h}_v^{(t+1)} = \sigma\Big(\mathbf{E}_{d(v)}^{(t)} \mathbf{m}_v^{(t+1)}\Big)$$

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

Step 2: The network computes the graph representation as:

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

The output $\mathbf{h}_G$ 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:

$$\mathbf{H}^{(t+1)} = f\big(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\mathbf{H}^{(t)}\mathbf{W}^{(t)}\big)$$

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

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

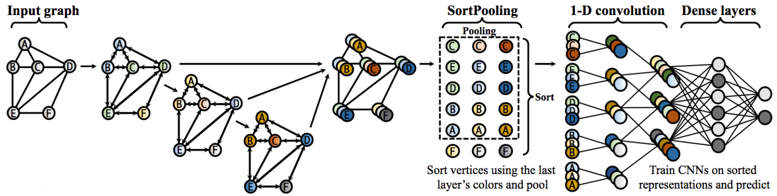
$$\mathbf{H} = \big[\mathbf{H}^{(1)}||\mathbf{H}^{(2)}||\ldots||\mathbf{H}^{(T)}\big]$$

[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



Input graph
SortPooling
Pooling
Sort
1-D convolution
Dense layers
Sort vertices using the last layer's colors and pool
Train CNNs on sorted representations and predict

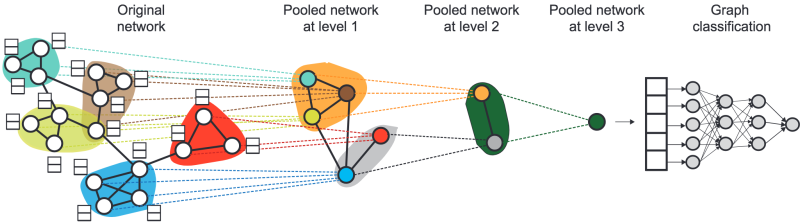
Differentiable Graph Pooling (DiffPool)

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


Original network
Pooled network at level 1
Pooled network at level 2
Pooled network at level 3
Graph classification

Differentiable Graph Pooling (DiffPool)

A matrix $\mathbf{S}^{(t)} \in \mathbb{R}^{n_t \times n_{t+1}}$ 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:

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

where $\mathbf{A}^{(t+1)}$ is the coarsened adjacency matrix, and $\mathbf{X}^{(t+1)}$ 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 $\mathbf{X}^{(t)}$ and coarsened adjacency matrix $\mathbf{A}^{(t)}$

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

  where the softmax function is applied in a row-wise fashion

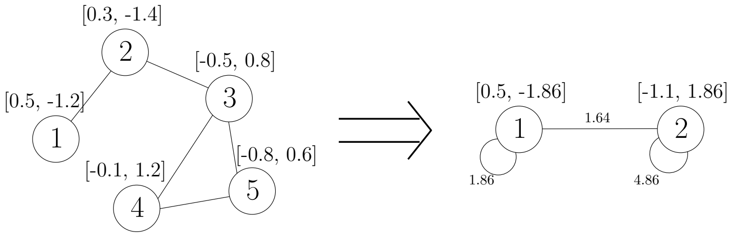
- $\text{GNN}_{\text{embed}}^{(t)}$ generates new representations for the input nodes

- $\text{GNN}_{\text{pool}}^{(t)}$ generates a probabilistic assignment of the input nodes to n_{t+1} clusters

Example of Coarsening Procedure of DiffPool

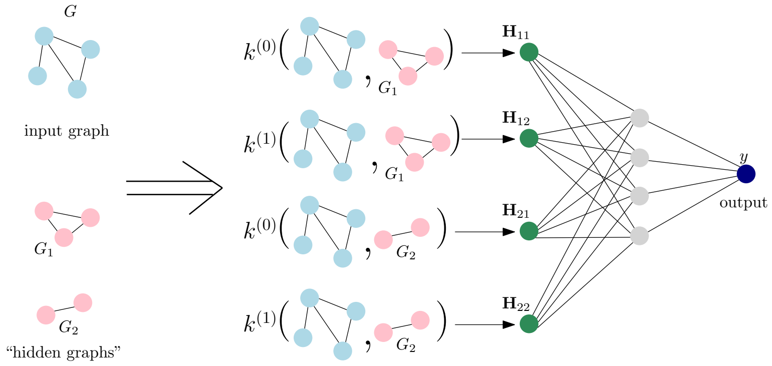
$$\mathbf{A}^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \mathbf{Z}^{(1)} = \begin{bmatrix} 0.5 & -1.2 \\ 0.3 & -1.4 \\ -0.5 & 0.8 \\ -0.1 & 1.2 \\ -0.8 & 0.6 \end{bmatrix} \quad \mathbf{S}^{(1)} = \begin{bmatrix}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

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

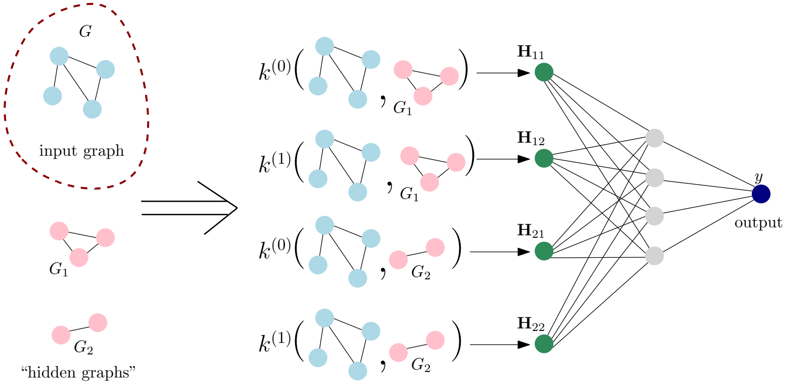


Outline

Random Walk Graph Neural Network (RWNN)

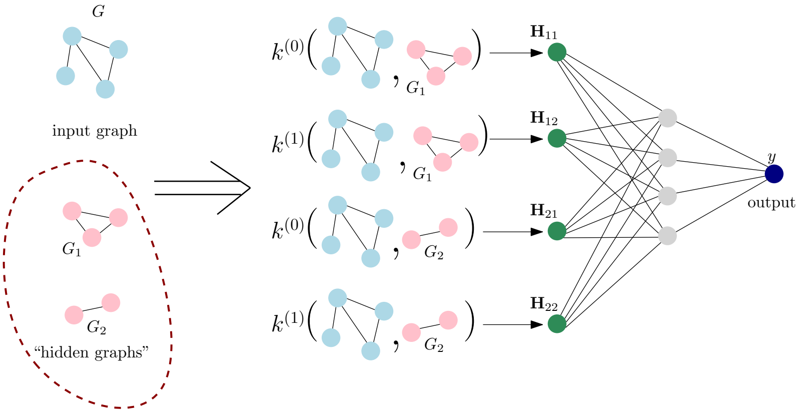


Random Walk Graph Neural Network (RWNN)



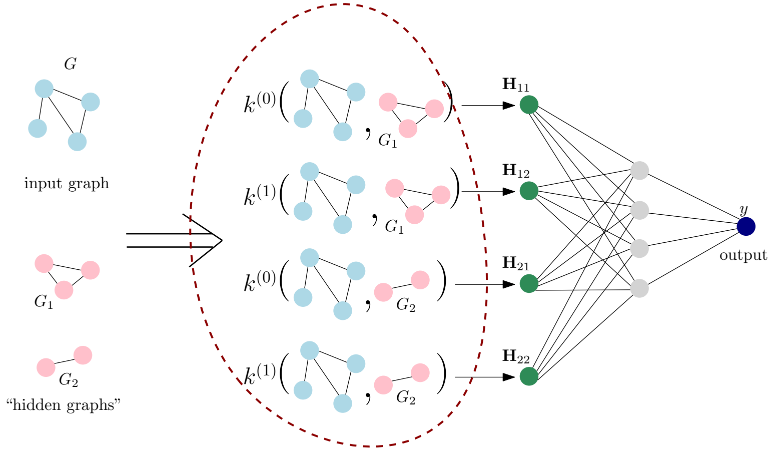
Given an input graph G

Random Walk Graph Neural Network (RWNN)


G
input graph
G_1
G_2
"hidden graphs"
$k^{(0)}\left(\quad , G_1 \right)$
$k^{(1)}\left(\quad , G_1 \right)$
$k^{(0)}\left(\quad , G_2 \right)$
$k^{(1)}\left(\quad , G_2 \right)$
$\mathbf{H}_{11}$
$\mathbf{H}_{12}$
$\mathbf{H}_{21}$
$\mathbf{H}_{22}$
y
output

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

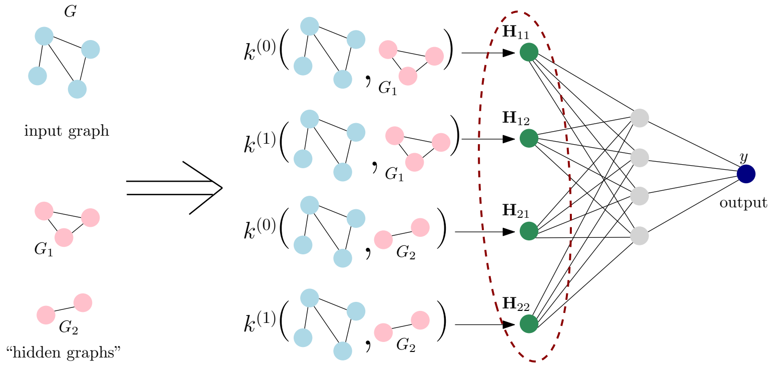
Random Walk Graph Neural Network (RWNN)



The model computes the following random-walk kernel between the input graph G and each "hidden graph" G_i:

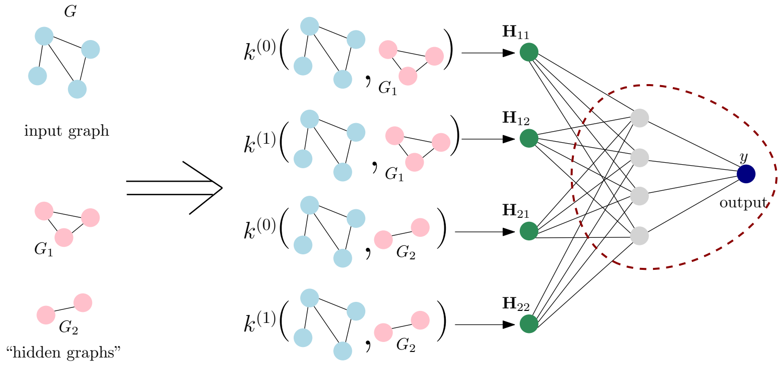
$$k^{(p)}(G, G_i) = \sum_{i=1}^{|V_\times|} \sum_{j=1}^{|V_\times|} \mathbf{s}_i \mathbf{s}_j \left[\mathbf{A}_\times^p\right]_{ij}$$

Random Walk Graph Neural Network (RWNN)



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

Random Walk Graph Neural Network (RWNN)



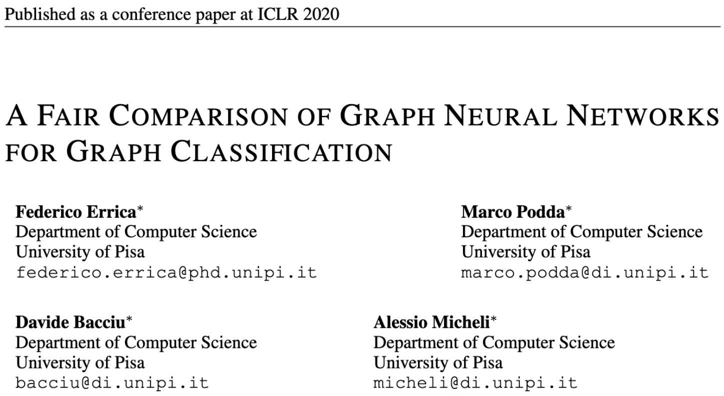
Matrix **H** is flattened and fed into a fully-connected neural network to produce the output

Experimental Evaluation - Graph Classification

We evaluated RWNN on the following 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

Baselines

- 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., NIPS'18]**
  - ECC **[Simonovsky and Komodakis, CVPR'17]**
  - GIN **[Xu et al., ICLR'19]**
  - GraphSAGE **[Hamilton et al., NIPS'17]**

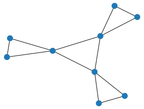
Graph Classification - Real World Datasets

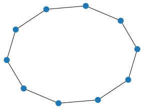
	MUTAG	D&D	NCI1	PROTEINS	ENZYMES
SP	80.2 (± 6.5)	**78.1** (± 4.1)	72.7 (± 1.4)	**75.3** (± 3.8)	38.3 (± 8.0)
GR	80.8 (± 6.4)	75.4 (± 3.4)	61.8 (± 1.7)	71.6 (± 3.1)	25.1 (± 4.4)
WL	84.6 (± 8.3)	**78.1** (± 2.4)	**84.8** (± 2.5)	73.8 (± 4.4)	50.3 (± 5.7)
DGCNN	84.0 (± 6.7)	76.6 (± 4.3)	76.4 (± 1.7)	72.9 (± 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)
1-step RWNN	**89.2** (± 4.3)	77.6 (± 4.7)	71.4 (± 1.8)	74.7 (± 3.3)	56.7 (± 5.2)
2-step RWNN	88.1 (± 4.8)	76.9 (± 4.6)	73.0 (± 2.0)	74.1 (± 2.8)	57.4 (± 4.9)
3-step RWNN	88.6 (± 4.1)	77.4 (± 4.9)	73.9 (± 1.3)	74.3 (± 3.3)	57.6 (± 6.3)

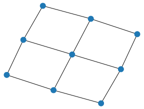
	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	COLLAB
SP	57.7 (± 4.1)	39.8 (± 3.7)	89.0 (± 1.0)	51.1 (± 2.2)	**79.9** (± 2.7)
GR	63.3 (± 2.7)	39.6 (± 3.0)	76.6 (± 3.3)	38.1 (± 2.3)	71.1 (± 1.4)
WL	**72.8** (± 4.5)	**51.2** (± 6.5)	74.9 (± 1.8)	49.6 (± 2.0)	78.0 (± 2.0)
DGCNN	69.2 (± 3.0)	45.6 (± 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)
1-step RWNN	70.8 (± 4.8)	47.8 (± 3.8)	**90.4** (± 1.9)	51.7 (± 1.5)	71.7 (± 2.1)
2-step RWNN	70.6 (± 4.4)	48.8 (± 2.9)	90.3 (± 1.8)	51.7 (± 1.4)	71.3 (± 2.1)
3-step RWNN	70.7 (± 3.9)	47.8 (± 3.5)	89.7 (± 1.2)	53.4 (± 1.6)	71.9 (± 2.5)

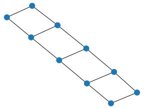
Results - Synthetic Datasets

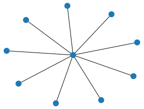
Structures planted into synthetic graphs


Caveman graph


Cycle graph

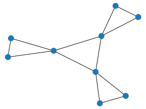

Grid graph

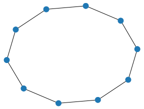

Ladder graph

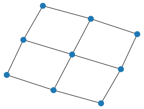

Star graph

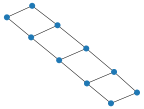
Results - Synthetic Datasets

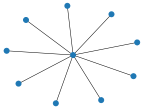
Structures planted into synthetic graphs


Caveman graph

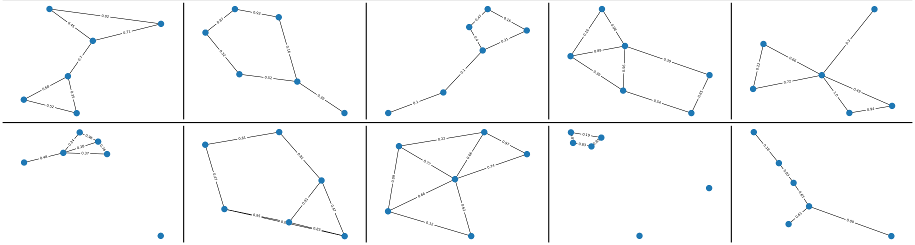

Cycle graph


Grid graph

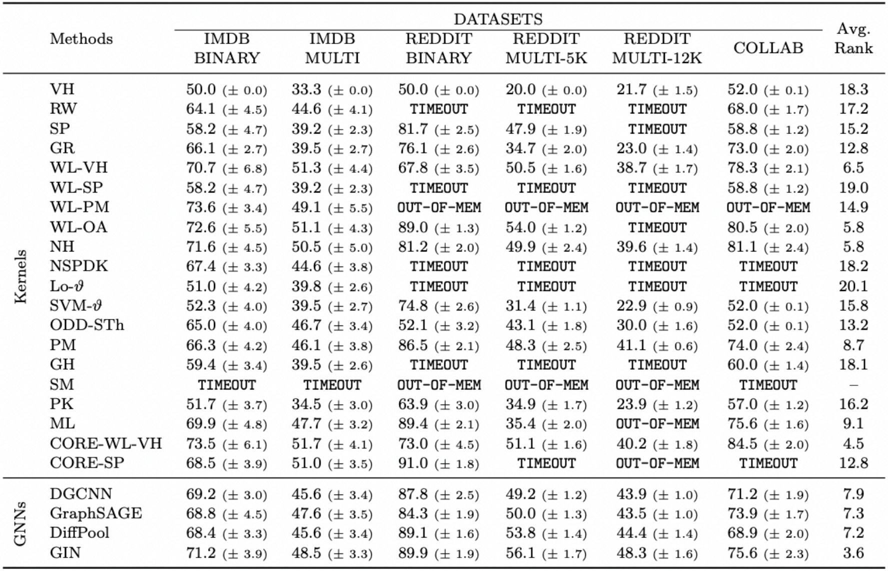

Ladder graph


Star graph

Examples of “hidden graphs” extracted from the proposed model



Graph Classification - Kernels vs. GNNs



	Methods	DATASETS						Avg. Rank
		IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	REDDIT MULTI-12K	COLLAB	
Kernels	VH	50.0 (± 0.0)	33.3 (± 0.0)	50.0 (± 0.0)	20.0 (± 0.0)	21.7 (± 1.5)	52.0 (± 0.1)	18.3
	RW	64.1 (± 4.5)	44.6 (± 4.1)	TIMEOUT	TIMEOUT	TIMEOUT	68.0 (± 1.7)	17.2
	SP	58.2 (± 4.7)	39.2 (± 2.3)	81.7 (± 2.5)	47.9 (± 1.9)	TIMEOUT	58.8 (± 1.2)	15.2
	GR	66.1 (± 2.7)	39.5 (± 2.7)	76.1 (± 2.6)	34.7 (± 2.0)	23.0 (± 1.4)	73.0 (± 2.0)	12.8
	WL-VH	70.7 (± 6.8)	51.3 (± 4.4)	67.8 (± 3.5)	50.5 (± 1.6)	38.7 (± 1.7)	78.3 (± 2.1)	6.5
	WL-SP	58.2 (± 4.7)	39.2 (± 2.3)	TIMEOUT	TIMEOUT	TIMEOUT	58.8 (± 1.2)	19.0
	WL-PM	73.6 (± 3.4)	49.1 (± 5.5)	OUT-OF-MEM	OUT-OF-MEM	OUT-OF-MEM	OUT-OF-MEM	14.9
	WL-OA	72.6 (± 5.5)	51.1 (± 4.3)	89.0 (± 1.3)	54.0 (± 1.2)	TIMEOUT	80.5 (± 2.0)	5.8
	NH	71.6 (± 4.5)	50.5 (± 5.0)	81.2 (± 2.0)	49.9 (± 2.4)	39.6 (± 1.4)	81.1 (± 2.4)	5.8
	NSPDK	67.4 (± 3.3)	44.6 (± 3.8)	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT	18.2
	Lo-ϑ	51.0 (± 4.2)	39.8 (± 2.6)	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT	20.1
	SVM-ϑ	52.3 (± 4.0)	39.5 (± 2.7)	74.8 (± 2.6)	31.4 (± 1.1)	22.9 (± 0.9)	52.0 (± 0.1)	15.8
	ODD-STh	65.0 (± 4.0)	46.7 (± 3.4)	52.1 (± 3.2)	43.1 (± 1.8)	30.0 (± 1.6)	52.0 (± 0.1)	13.2
	PM	66.3 (± 4.2)	46.1 (± 3.8)	86.5 (± 2.1)	48.3 (± 2.5)	41.1 (± 0.6)	74.0 (± 2.4)	8.7
	GH	59.4 (± 3.4)	39.5 (± 2.6)	TIMEOUT	TIMEOUT	TIMEOUT	60.0 (± 1.4)	18.1
	SM	TIMEOUT	TIMEOUT	OUT-OF-MEM	OUT-OF-MEM	OUT-OF-MEM	TIMEOUT	–
	PK	51.7 (± 3.7)	34.5 (± 3.0)	63.9 (± 3.0)	34.9 (± 1.7)	23.9 (± 1.2)	57.0 (± 1.2)	16.2
	ML	69.9 (± 4.8)	47.7 (± 3.2)	89.4 (± 2.1)	35.4 (± 2.0)	OUT-OF-MEM	75.6 (± 1.6)	9.1
	CORE-WL-VH	73.5 (± 6.1)	51.7 (± 4.1)	73.0 (± 4.5)	51.1 (± 1.6)	40.2 (± 1.8)	84.5 (± 2.0)	4.5
	CORE-SP	68.5 (± 3.9)	51.0 (± 3.5)	91.0 (± 1.8)	TIMEOUT	OUT-OF-MEM	TIMEOUT	12.8
GNNs	DGCNN	69.2 (± 3.0)	45.6 (± 3.4)	87.8 (± 2.5)	49.2 (± 1.2)	43.9 (± 1.0)	71.2 (± 1.9)	7.9
	GraphSAGE	68.8 (± 4.5)	47.6 (± 3.5)	84.3 (± 1.9)	50.0 (± 1.3)	43.5 (± 1.0)	73.9 (± 1.7)	7.3
	DiffPool	68.4 (± 3.3)	45.6 (± 3.4)	89.1 (± 1.6)	53.8 (± 1.4)	44.4 (± 1.4)	68.9 (± 2.0)	7.2
	GIN	71.2 (± 3.9)	48.5 (± 3.3)	89.9 (± 1.9)	56.1 (± 1.7)	48.3 (± 1.6)	75.6 (± 2.3)	3.6

"Graph Kernels: a Survey", *G.Nikolentzos*, *M.Vazirgiannis*, JAIR 2021

Weisfeiler and Leman go Hyperbolic: Learning Distance Preserving Graph Representations, G. Nikolentzos, M. Chatzianastasis, M. Vazirgiannis, **AISTATS2023**

Message-Passing Neural Networks

- Let a graph $G = (V, E)$.
- For each node $u \in V$, we define its neighborhood as $\mathcal{N}(u) = \{v : (u, v) \in E\}$
- Neighborhood Features: $\mathbf{X}_{\mathcal{N}(u)} = \{\!\{\mathbf{x}_v : v \in \mathcal{N}(u)\}\!\}$ where $\{\!\{\cdot\}\!\}$ denotes a multiset

Weisfeiler and Leman go Hyperbolic: Learning Distance Preserving Graph Representations, G. Nikolentzos, M. Chatzianastasis, M. Vazirgiannis, **AISTATS2023**

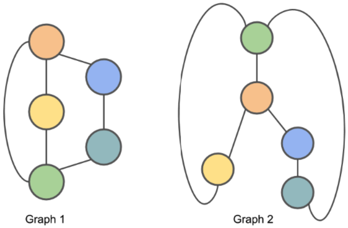
Message-Passing Neural Networks

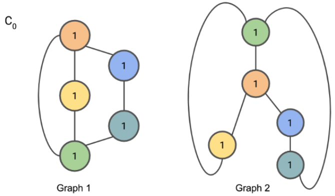
- Let a graph $G = (V, E)$.
- For each node $u \in V$, we define its neighborhood as $\mathcal{N}(u) = \{v : (u, v) \in E\}$
- Neighborhood Features: $\mathbf{X}_{\mathcal{N}(u)} = \{\!\{\mathbf{x}_v : v \in \mathcal{N}(u)\}\!\}$ where $\{\!\{\cdot\}\!\}$ denotes a multiset
- Graph Neural Networks usually perform computations on each node's neighbourhood.
- At layer k of a GNN, each node aggregates the messages from its neighbours and combines it with its previous state.

$$\mathbf{m}_u^k = \mathrm{AGGREGATE}^k\left(\{\mathbf{x}_v^{k-1} : v \in \mathcal{N}(u)\}\right),$$
$$\mathbf{x}_u^k = \mathrm{COMBINE}^k\left(\mathbf{x}_u^{k-1}, \mathbf{m}_u^k\right).$$

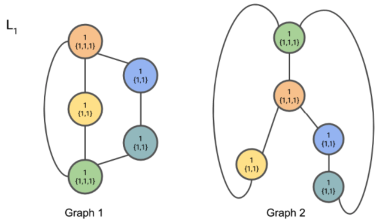
Weisfeiler-Leman (1-WL) Algorithm

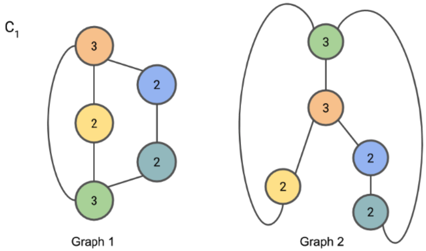
- Test of graph isomorphism
- The classical WL (or 1-WL) keeps a state for each node that refines by aggregating their neighbors state. It outputs an embedding of the graph that corresponds to the state of every node. We say that the WL succeeds at distinguishing a pair of non-isomorphic graphs $G, \hat{G}$ if $WL(G) \neq WL(\hat{G})$.
1-WL Example



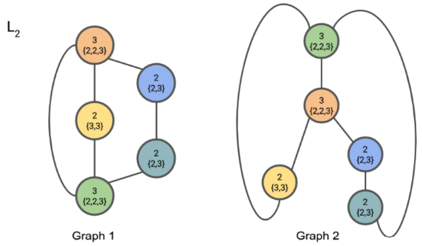


1-WL Example

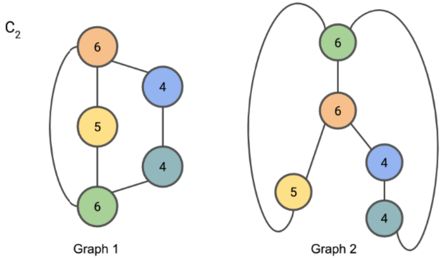

L_1
1 {1,1,1}
1 {1,1}
1 {1,1}
1 {1,1}
1 {1,1,1}
Graph 1
1 {1,1,1}
1 {1,1,1}
1 {1,1}
1 {1,1}
1 {1,1}
Graph 2


C_1
3
2
2
2
3
Graph 1
3
3
2
2
2
Graph 2

1-WL Example

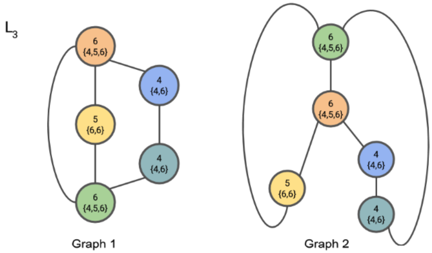


L_2
3 {2,2,3}
2 {2,3}
2 {3,3}
2 {2,3}
3 {2,2,3}
Graph 1
3 {2,2,3}
3 {2,2,3}
2 {2,3}
2 {3,3}
2 {2,3}
Graph 2

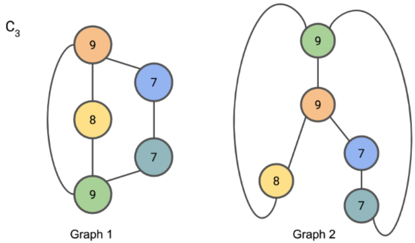


C_2
6
4
5
4
6
Graph 1
6
6
4
5
4
Graph 2

1-WL Example



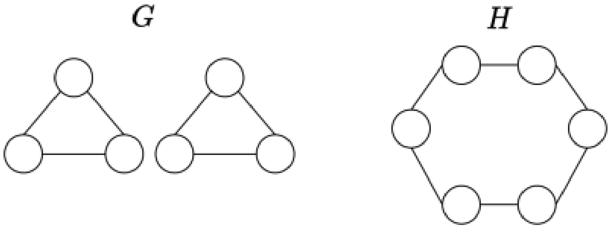
L_3
6 {4,5,6}
4 {4,6}
5 {6,6}
4 {4,6}
6 {4,5,6}
Graph 1
6 {4,5,6}
6 {4,5,6}
4 {4,6}
5 {6,6}
4 {4,6}
Graph 2



C_3
9
7
8
7
9
Graph 1
9
9
7
8
7
Graph 2

Expressive Power of GNNs

- Several studies have investigated how GNNs are related to the WL test of isomorphism and its higher-order variants.
- It was shown that standard GNNs are at most as powerful as the WL algorithm in terms of distinguishing non-isomorphic graphs
- Other studies proposed families of GNNs whose message passing scheme is equivalent to high-order variants of the WL algorithm, and can thus distinguish more pairs of non-isomorphic graphs than standard MPNNs



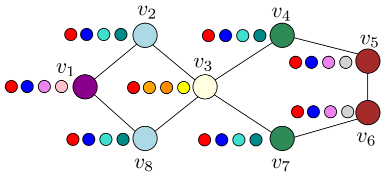
Distances between graphs are ignored

- The above studies investigate the power of GNNs in terms of distinguishing non-isomorphic graphs.
- Even though there exist several powerful models which can distinguish almost all non-isomorphic graphs from each other, these models largely ignore the distance between nodes.
- In graph classification/regression problems, we are not that much interested in testing whether two (sub)graphs are isomorphic to each other.
- It has been observed that stronger GNNs (in the aforementioned sense) do not necessarily outperform weaker GNNs
- **Capturing such distances between nodes is of paramount importance for machine learning since similar graphs usually belong to the same class or are associated with similar target values**.

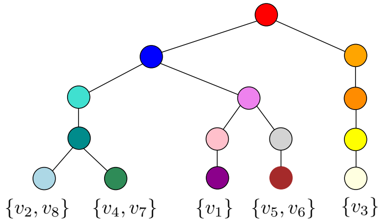
Graph Distances

- A natural question is: how is the distance between two nodes defined?
- Unfortunately, there is no clear answer to the above question.
- Several distance functions were proposed for comparing graphs, subgraphs or nodes (i.e., subgraphs centered at nodes).
- Most of those functions are hard to compute (NP-hard).

1-WL Tree Hierarchy



(a) A graph G



(b) hierarchy H_G

Figure: An illustration of (a) a graph G with uniform initial colors c_0 and refined colors c_i for $i \in [4]$, and of (b) its corresponding WL tree hierarchy H_G. The nodes of G are the leaves of the hierarchy.

WL Distance

We define a distance function between nodes which can be derived from the hierarchy generated by the WL algorithm.

Definition (WL distance)

Let $\mathcal{V}$ be the set of nodes of all graphs of the corpus. Let $H = (V, E)$ be a rooted tree representing the hierarchy produced by the WL algorithm. Then, each element of $\mathcal{V}$ corresponds to a leaf of tree T. Suppose that two nodes $v_1, v_2 \in \mathcal{V}$ correspond to leaves $u_1, u_2 \in V$, respectively. Then, the WL distance between nodes v_1 and v_2 is defined as $d_{\text{WL}}(v_1, v_2) = \text{sp}(u_1, u_2)$ where $\text{sp}(\cdot, \cdot)$ denotes the shortest path distance between two nodes of tree T.

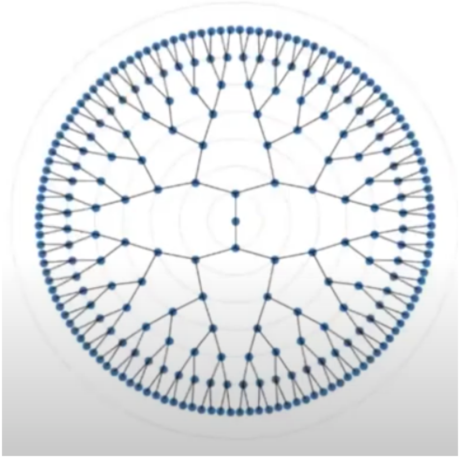
Hierarchical Graph Structures in Euclidean Space

Task: Embed a tree in Euclidean Space.

- Hierarchical nature of the tree: parent-child relationships. Children and their parents should be close in the embedding space.
- Relative "distance" between the nodes. Leaf nodes in totally different branches of the tree should be very far apart.

Euclidean Space is too "Narrow" for Hierarchical Graph Structures

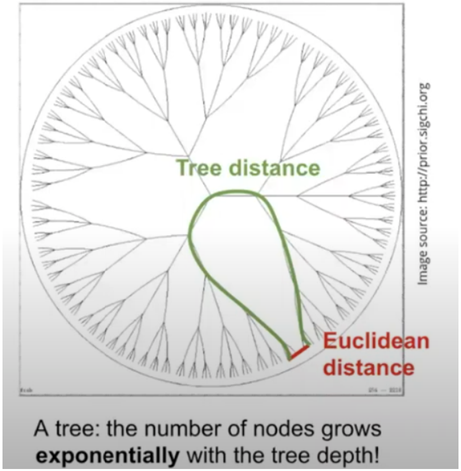
The distortion is even bigger as we add more nodes. But why?



http://building-babylon.net/

Euclidean Space is too "Narrow" for Hierarchical Graph Structures

- Euclidean Ball Volume: $V_d^{\mathbb{E}}(r) = \Theta(r^d)$
- Number of nodes grows exponentially with the tree depth: $l = b^r$, where b is the branching factor, l are the leaf nodes.

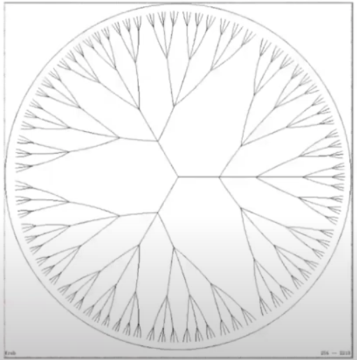


A tree: the number of nodes grows **exponentially** with the tree depth!

Hierarchial Graph Structures in Hyperbolic Space

image source: http://inspirehep.net/record/1355197/plots

The volume of a ball in the hyperbolic space grows **exponentially** with its radius!



Similarly as for a tree: the number of nodes grows **exponentially** with the tree depth!

https://people.csail.mit.edu/oct/

Trees Into Hyperbolic Space

- It is well-known that trees can be embedded into the Poincaré disk $\mathbb{D}^2$ with arbitrarily low distortion [sarkar2011]
- Constructive algorithm (no learning involved)
- The idea is to embed the root at the origin and recursively embed the children of each node in the tree by spacing them around a sphere centered at the parent.

Euclidean vs Hyperoblic GNNs

- GNNs on Euclidean space:
  $\mathbf{h}_u^{k+1} = \sigma(\sum_{v \in V} \mathbf{A}_{uv} \mathbf{W}^k \mathbf{h}_v^k)$

- GNNs on Hyperoblic space:
  $\mathbf{h}_u^{k+1} = \sigma\left(\exp_{\mathbf{x}}\left(\sum_{v \in V} \mathbf{A}_{uv} \mathbf{W}^k \log_{\mathbf{x}}\left(\mathbf{h}_v^k\right)\right)\right)$

Weisfeiler-Leman Hyperbolic Network

- We propose a new MPNN that learns node representations that respect the distances between nodes, as those are defined by the WL algorithm.
- 1-WL or GNNs produce a tree hierarchy of the nodes.
- Trees can be embedded with arbitrarily low distortion into the hyperbolic space, while Euclidean space cannot achieve such a low distortion [Sarkar2011]
- Euclidean MPNNs cannot encode accurately the information contained in the WL hierarchy. We propose a MPNN which delivers the "best of both worlds" from Euclidean space and hyperbolic space.
- The proposed model takes into account the distance of the input nodes according to the hierarchy induced by WL, but also according to the representations that emerge from the neighborhood aggregation procedure.
- To embed the WL tree hierarchy into a vector space, we capitalize on recent advances in hyperbolic representations using the Poincare ball model [Ganea2018,Chami2019]

Distance-Aware Hyperbolic Graph Neural Network

1. **Neighborhood aggregation using GNN**:

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

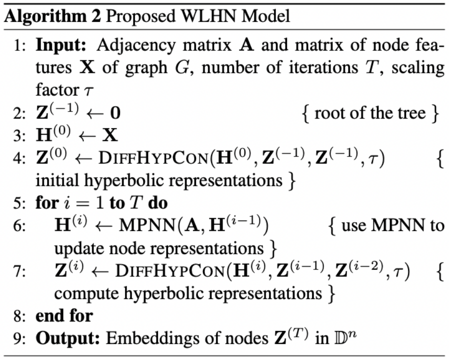
2. Adaptation of Sarkar's construction to embed those representations into the hyperbolic space:

$$\mathbf{z}_v^{(t)} = f(\mathbf{h}_v^{(t)}, \mathbf{h}_v^{(t-1)}, \mathbf{h}_v^{(t-2)})$$

   where f is our proposed construction.

Each neighborhood aggregation operation is followed by an embedding phase where the emerging node representations are mapped to the Poincaré ball using the proposed construction.

WLHN algorithm



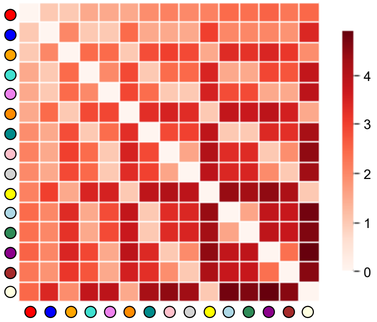
Algorithm 2 Proposed WLHN Model

```
1: Input: Adjacency matrix $\mathbf{A}$ and matrix of node features $\mathbf{X}$ of graph $G$, number of iterations $T$, scaling factor $\tau$
2: $\mathbf{Z}^{(-1)} \leftarrow \mathbf{0}$    { root of the tree }
3: $\mathbf{H}^{(0)} \leftarrow \mathbf{X}$
4: $\mathbf{Z}^{(0)} \leftarrow \textsc{DiffHypCon}(\mathbf{H}^{(0)}, \mathbf{Z}^{(-1)}, \mathbf{Z}^{(-1)}, \tau)$    { initial hyperbolic representations }
5: for $i = 1$ to $T$ do
6:     $\mathbf{H}^{(i)} \leftarrow \text{MPNN}(\mathbf{A}, \mathbf{H}^{(i-1)})$    { use MPNN to update node representations }
7:     $\mathbf{Z}^{(i)} \leftarrow \textsc{DiffHypCon}(\mathbf{H}^{(i)}, \mathbf{Z}^{(i-1)}, \mathbf{Z}^{(i-2)}, \tau)$    { compute hyperbolic representations }
8: end for
9: Output: Embeddings of nodes $\mathbf{Z}^{(T)}$ in $\mathbb{D}^n$
```

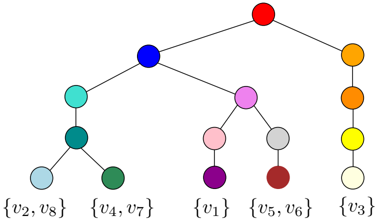
DiffHypCon: Proposed Differentiable Hyperbolic Construction - mapping
Readout function: $h_G = \sum_{u \in V} log_o(z_u^T)$

Experimental Results

Figure: Heatmap that illustrates the distances between all pairs of nodes of the hierarchy H_G. Distances are computed between the nodes' generated hyperbolic representations.



(a) Heatmap



(b) hierarchy H_G

Experimental Results

Table: Classification accuracy (± standard deviation) of the proposed model and the baselines on the 10 benchmark datasets. OOR means Out of Resources, either time (>72 hours for a single training) or GPU memory.

	MUTAG	**D&D**	**NCI1**	**PROTEINS**	**ENZYMES**
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)
HGCN (PoincareBall)	83.4 (± 6.7)	78.0 (± 2.8)	74.2 (± 2.4)	74.4 (± 3.1)	39.7 (± 5.5)
HGCN (Hyperboloid)	83.4 (±6.2)	77.8 (±4.3)	72.3 (± 4.3)	74.7 (± 3.4)	32.3 (± 5.4)
WLHN	**86.0** (± 7.4)	**78.5** (± 3.4)	79.2 (± 1.1)	**75.9** (± 1.9)	**62.5** (± 5.0)

	IMDB-B	**IMDB-M**	**REDDIT-B**	**REDDIT-5K**	**COLLAB**
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)
HGCN (PoincareBall)	73.0 (± 3.2)	**50.3** (± 3.8)	87.9 (± 2.8)	49.4 (± 2.6)	80.2 (± 1.9)
HGCN (Hyperboloid)	73.3 (± 3.5)	**50.3** (± 4.0)	86.3 (± 1.6)	52.7 (± 2.0)	**80.3** (± 1.8)
WLHN	**73.4** (± 3.7)	49.7 (± 3.6)	**90.7** (± 1.9)	55.2 (± 1.2)	76.2 (± 2.3)

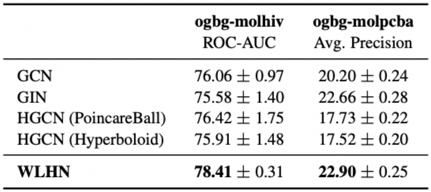
Running Time

Table: Average running time per epoch (in seconds).

	MUTAG	**D&D**	**NCI1**	**PROTEINS**	**ENZYMES**
GIN	0.09	5.51	0.73	0.24	0.14
HGCN	0.13	6.58	1.32	0.44	0.23
WLHN	0.18	5.85	2.07	0.61	0.16

	IMDB-B	**IMDB-M**	**REDDIT-B**	**REDDIT-5K**	**COLLAB**
GIN	0.22	0.34	0.77	2.01	15.30
HGCN	0.43	0.63	2.10	7.10	16.20
WLHN	0.50	0.73	2.30	6.83	16.55

Experimental results



	ogbg-molhiv ROC-AUC	**ogbg-molpcba** Avg. Precision
GCN	76.06 ± 0.97	20.20 ± 0.24
GIN	75.58 ± 1.40	22.66 ± 0.28
HGCN (PoincareBall)	76.42 ± 1.75	17.73 ± 0.22
HGCN (Hyperboloid)	75.91 ± 1.48	17.52 ± 0.20
WLHN	**78.41** ± 0.31	**22.90** ± 0.25

Performance of the proposed model and the baselines on the large ogbg-molhiv and ogbg-molpcba datasets

Conclusion

- We defined a distance function between the nodes and proposed a novel GNN model which can accurately capture that distances by embedding the nodes of the graphs in the hyperbolic space.
- We create a level of a tree hierarchy in each message passing layer, and embed the nodes of this level to hyperbolic space.
- Our method can be incorporated in various GNNs models.
- In contrast with other Hyperbolic Graph Neural Networks, we explicitly construct a tree hierarchy from the graph instead of trying to capture it implicitly.
- Our results demonstrate that the proposed model can indeed encode meaningful distances in the learned representations, while it achieves high levels of performance in graph classification problems.

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

$\hookrightarrow$ NLP: documents as sets of word embeddings

$\hookrightarrow$ Graph Mining: graphs as sets of node embeddings

$\hookrightarrow$ 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 $f : 2^{\mathcal{X}} \to \in \{-1, 1\}$ 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:

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

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)

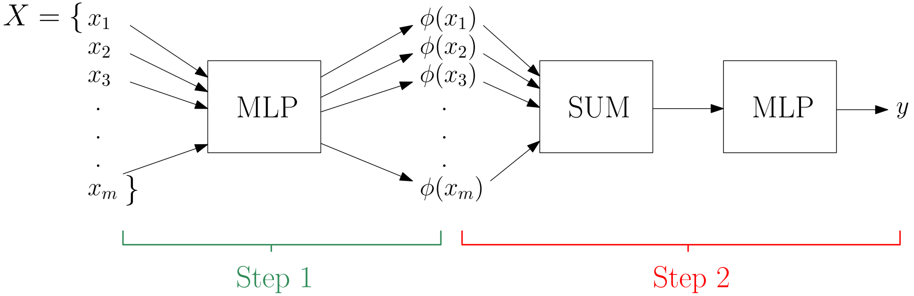
If $\mathfrak{X}$ is a countable set and $\mathcal{Y} = \mathbb{R}$, then a function $f(X)$ operating on a set X having elements from $\mathfrak{X}$ is a valid set function, i.e., invariant to the permutation of instances in X, if and only if it can be decomposed in the form $\rho(\sum_{x \in X} \phi(x))$, 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 $\phi(x_1), \ldots, \phi(x_m)$

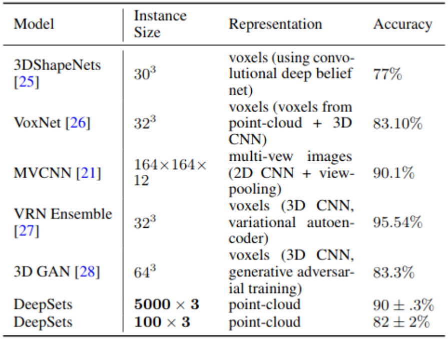
Step 2: A representation for the entire set is produced as $z_X = \phi(x_1) + \ldots + \phi(x_m)$ and is also transformed as follows $y = \rho(z_X)$ to produce the output

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



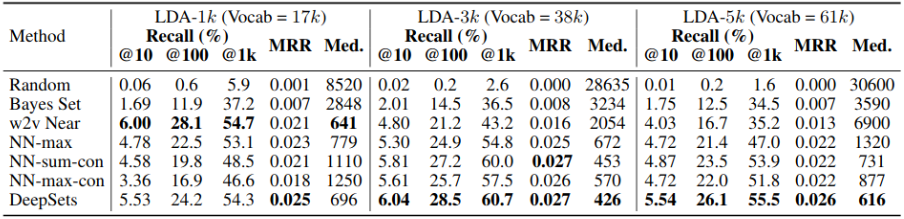
Model	Instance Size	Representation	Accuracy
3DShapeNets [25]	30^3	voxels (using convolutional deep belief net)	77%
VoxNet [26]	32^3	voxels (voxels from point-cloud + 3D CNN)	83.10%
MVCNN [21]	$164 \times 164 \times 12$	multi-vew images (2D CNN + view-pooling)	90.1%
VRN Ensemble [27]	32^3	voxels (3D CNN, variational autoencoder)	95.54%
3D GAN [28]	64^3	voxels (3D CNN, generative adversarial training)	83.3%
DeepSets	$\mathbf{5000 \times 3}$	point-cloud	$90 \pm .3\%$
DeepSets	$\mathbf{100 \times 3}$	point-cloud	$82 \pm 2\%$

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

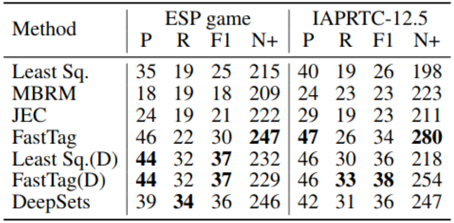


Method	LDA-1k (Vocab = 17k)					LDA-3k (Vocab = 38k)					LDA-5k (Vocab = 61k)				
	Recall (%)			**MRR**	**Med.**	**Recall (%)**			**MRR**	**Med.**	**Recall (%)**			**MRR**	**Med.**
	@10	**@100**	**@1k**			**@10**	**@100**	**@1k**			**@10**	**@100**	**@1k**		
Random	0.06	0.6	5.9	0.001	8520	0.02	0.2	2.6	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

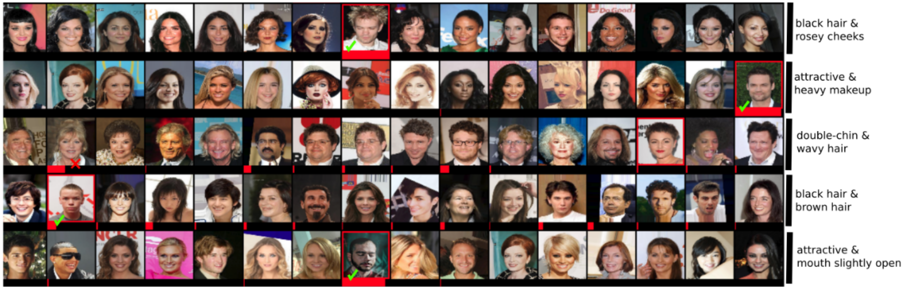


Method	ESP game				IAPRTC-12.5			
	P	R	F1	N+	P	R	F1	N+
Least Sq.	35	19	25	215	40	19	26	198
MBRM	18	19	18	209	24	23	23	223
JEC	24	19	21	222	29	19	23	211
FastTag	46	22	30	**247**	**47**	26	34	**280**
Least Sq.(D)	**44**	32	**37**	232	46	30	36	218
FastTag(D)	**44**	32	**37**	229	46	**33**	**38**	254
DeepSets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



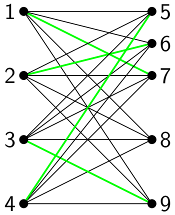
RepSet - towards interpretable set representation learning

- 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" $Y_1, Y_2, \dots, Y_m$ 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 $X = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{|X|}\}$ where $\mathbf{v}_1, \dots, \mathbf{v}_{|X|}$ vectors
- "Hidden set" $Y = \{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{|Y|}\}$
- Maximum matching between the elements of X and Y by solving the following linear program:

$$\max \sum_{i=1}^{|X|} \sum_{j=1}^{|Y|} z_{ij} f(\mathbf{v}_i, \mathbf{u}_j) \text{ subject to:}$$

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

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

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

where $f(\mathbf{v}_i, \mathbf{u}_j)$ 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” $Y_1, Y_2, \dots, Y_m$

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
   $$\mathbf{p} = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$
   where $\mathbf{W}$ is a matrix of trainable parameters and $\mathbf{b}$ is the bias term

RepSet - Architecture

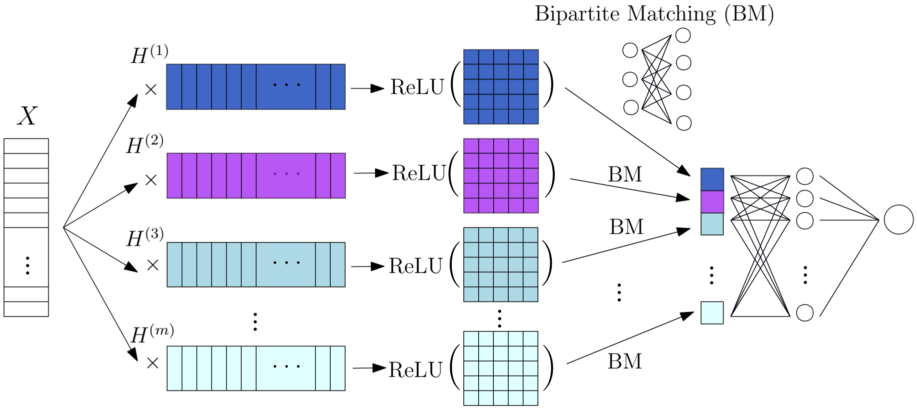


Figure: Each element of the input set is compared with the elements of all "hidden sets", and the emerging matrices serve as the input to bipartite matching. The values of the BM problems correspond to the representation of the input set.

RepSet - Relaxed Variant (ApproxRepSet)

- Input set $X = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{|X|}\}$ where $\mathbf{v}_1, \dots, \mathbf{v}_{|X|}$ vectors
- "Hidden set" $Y = \{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{|Y|}\}$
- Identify which of the two sets has the highest cardinality.
- If $|X| \geq |Y|$, we solve the following problem:

$$\max \sum_{i=1}^{|X|} \sum_{j=1}^{|Y|} z_{ij} f(\mathbf{v}_i, \mathbf{u}_j) \text{ subject to:}$$

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

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

- 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 $f(\mathbf{v}_i, \mathbf{u}_j)$ is positive and $f(\mathbf{v}_i, \mathbf{u}_j) = \max_k f(\mathbf{v}_k, \mathbf{u}_j)$

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

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

Conclusions

THANK YOU !

Acknowledgements
Dr. Giannis Nikolentzos, M. Chatzianastasis, G. Salha

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