Random Walk Graph Neural Networks

Giannis Nikolentzos

Complex Network Analysis (CNA) group meeting

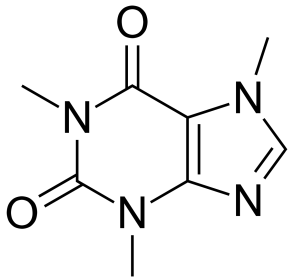
6 October 2021

Outline

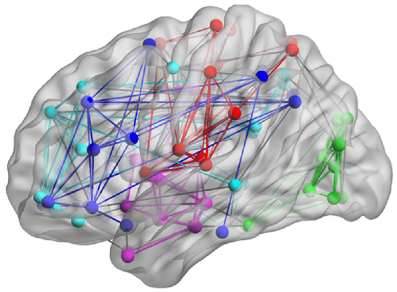
Outline

Machine Learning on Graphs

- Graphs are everywhere!!



Molecule



Brain network

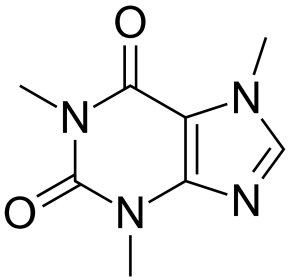
Social network

- Several different problems:
  - Node classification/regression
  - Link prediction
  - Community detection
  - Graph classification/regression ✓

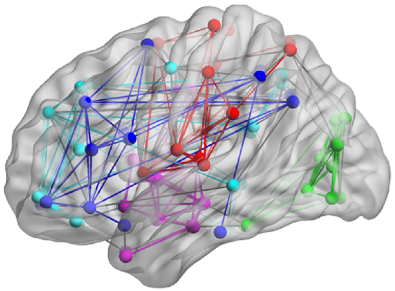
- Several applications:
  - predicting the quantum mechanical properties of molecules **[Gilmer at al., ICML'17]**
  - modeling physical systems **[Battaglia et al., NIPS'16]**

Machine Learning on Graphs

- Graphs are everywhere!!



Molecule



Brain network

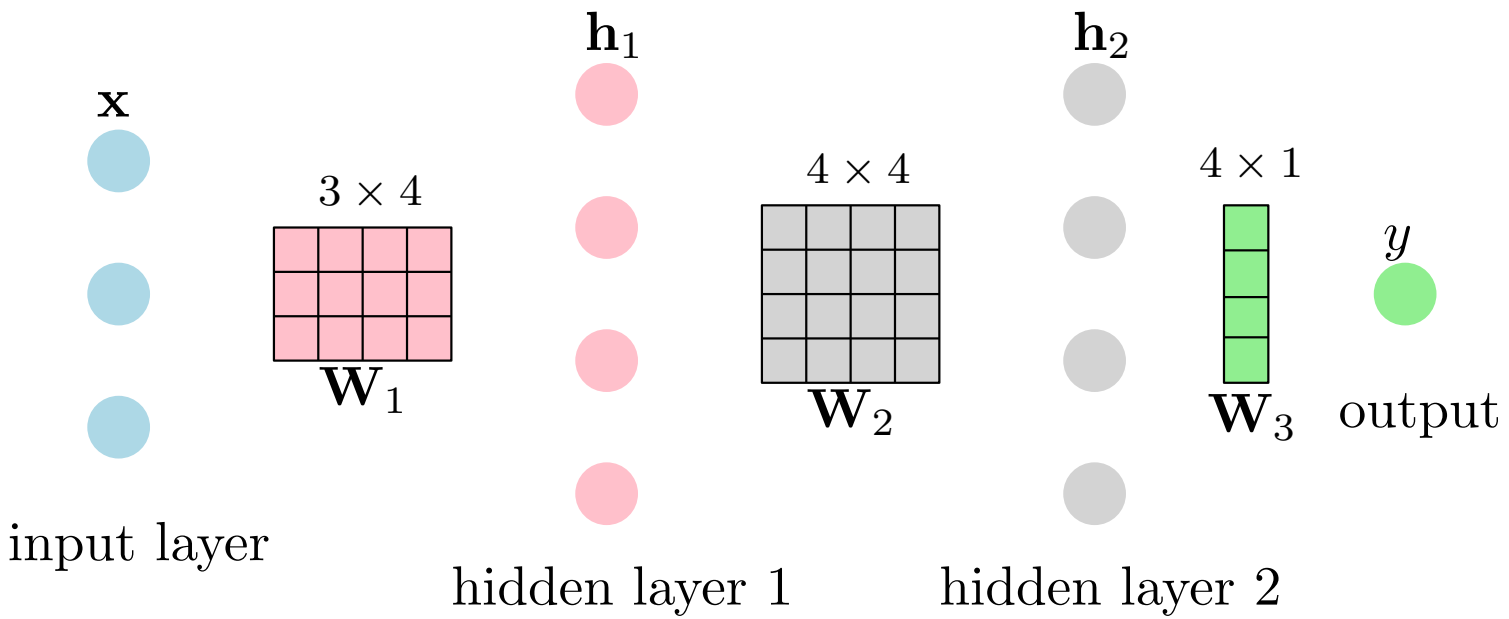
Social network

- Several different problems:
  - Node classification/regression
  - Link prediction
  - Community detection
  - Graph classification/regression ✓

- Several applications:
  - predicting the quantum mechanical properties of molecules **[Gilmer at al., ICML'17]**
  - modeling physical systems **[Battaglia et al., NIPS'16]**

Question: What should a neural network for graphs be like?

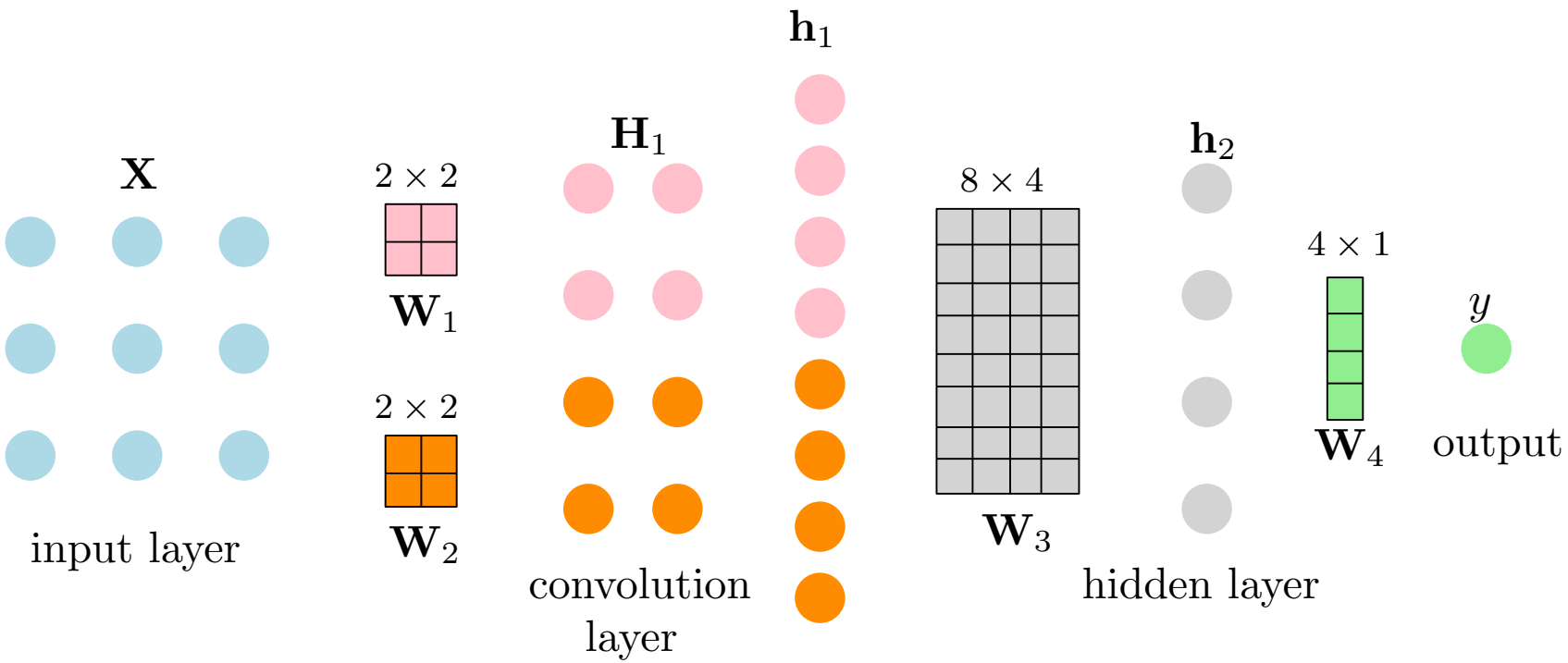
Vectors - MLPs



$$\mathbf{h}_1 = f(\mathbf{x}^\top \mathbf{W}_1)$$
$$\mathbf{h}_2 = f(\mathbf{h}_1^\top \mathbf{W}_2)$$
$$y = f(\mathbf{h}_2^\top \mathbf{W}_3)$$

Warning: Biases and activation functions are omitted

Images - CNNs



$$\mathbf{h}_1^{(1)} = f\Big(\sum_{i=1}^{2}\sum_{j=1}^{2}\big[\mathbf{X}_{1:2,1:2} \odot \mathbf{W}_1\big]_{ij}\Big)$$

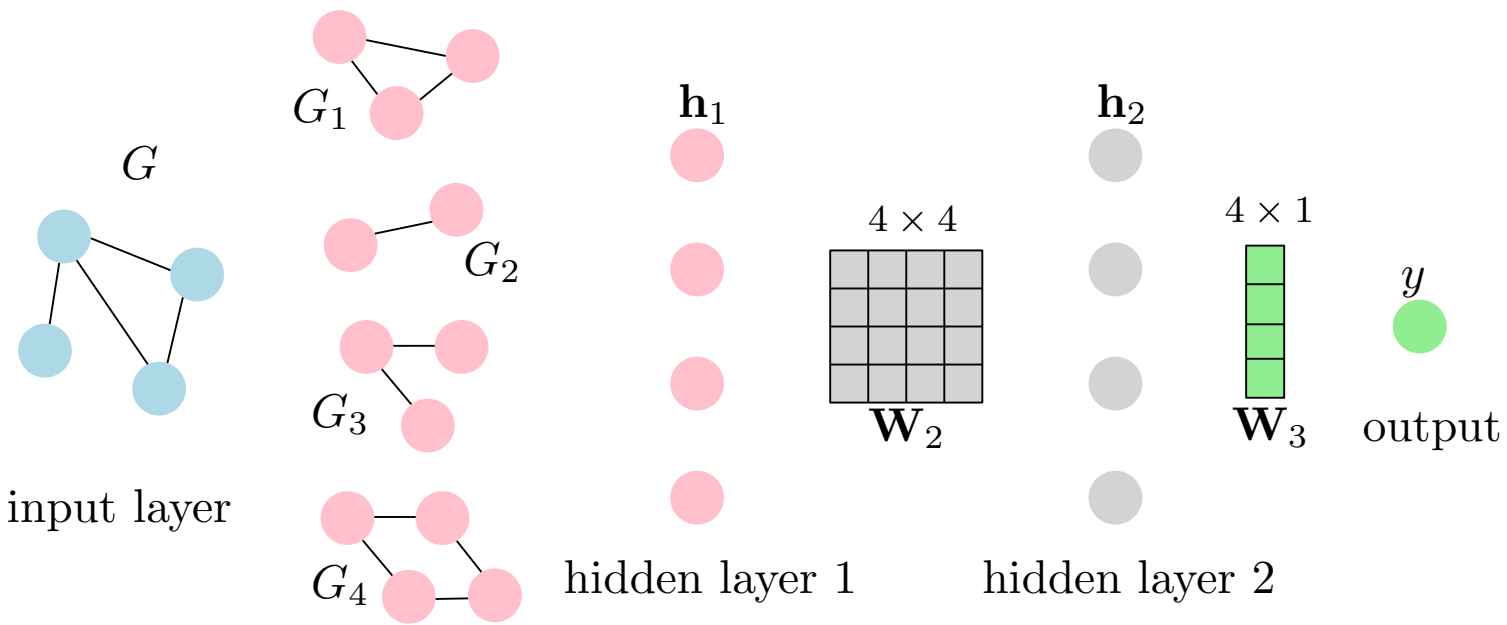
$$\mathbf{h}_1^{(8)} = f\Big(\sum_{i=1}^{2}\sum_{j=1}^{2}\big[\mathbf{X}_{3:4,3:4} \odot \mathbf{W}_2\big]_{ij}\Big)$$

$$\mathbf{h}_2 = f(\mathbf{h}_1^\top \mathbf{W}_3)$$

$$y = f(\mathbf{h}_2^\top \mathbf{W}_4)$$

Warning: Biases and activation functions are omitted

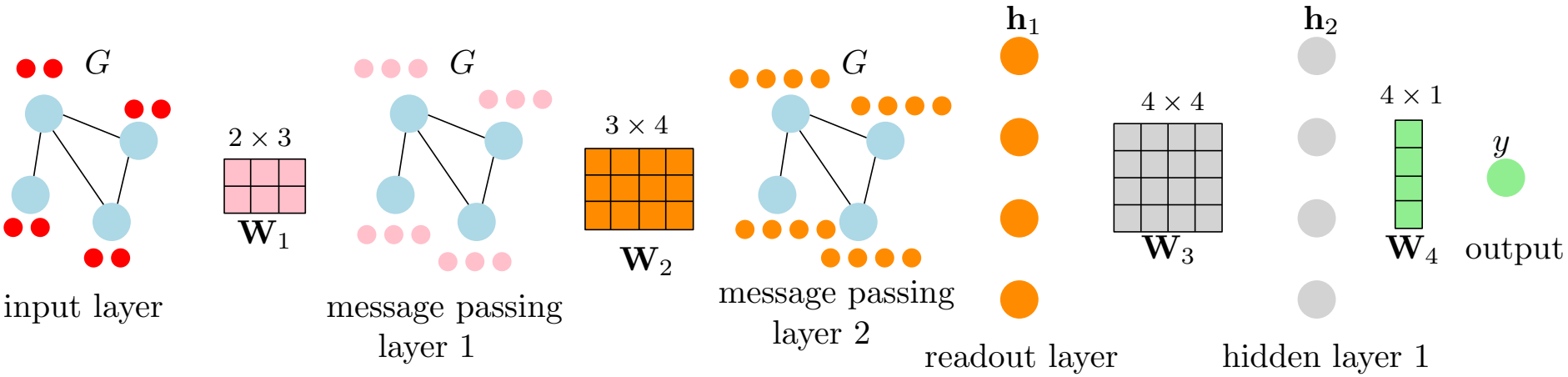
Graphs - What we would expect



$$\mathbf{h}_1^{(1)} = g(G, G_1)$$
$$\mathbf{h}_1^{(2)} = g(G, G_2)$$
$$\mathbf{h}_1^{(3)} = g(G, G_3)$$
$$\mathbf{h}_1^{(4)} = g(G, G_4)$$
$$\mathbf{h}_2 = f(\mathbf{h}_1^\top \mathbf{W}_2)$$
$$y = f(\mathbf{h}_2^\top \mathbf{W}_3)$$

Warning: Biases and activation functions are omitted

Graphs - What we have - Message Passing Neural Networks



$$\mathbf{x}_i^{(2)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} f(\mathbf{x}_j^{(1)\top} \mathbf{W}_1)$$

$$\mathbf{x}_i^{(3)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} f(\mathbf{x}_j^{(2)\top} \mathbf{W}_2)$$

$$\mathbf{h}_1 = \sum_{i=1}^{4} \mathbf{x}_i^{(3)}$$

$$\mathbf{h}_2 = f(\mathbf{h}_1^\top \mathbf{W}_3)$$

$$y = f(\mathbf{h}_2^\top \mathbf{W}_4)$$

Warning: Biases and activation functions are omitted

Why?

- The function that compares graphs to each other needs to be differentiable
  $\hookrightarrow$ e. g., $g(G, G_i)$ needs to be differentiable

- Graph comparison functions are usually expensive to compute
  $\hookrightarrow$ Many problems related to it are **NP-complete**
  - subgraph isomorphism
  - finding largest common subgraph

- Message Passing Framework have proven successful in several tasks
  - Led to the discovery of a new antibiotic **[Stokes et al., Cell 180(4)]**
  - Deployed to improve Google Maps Estimated Time of Arrival **[Derrow-Pinion et al., arXiv:2108.11482]**

Why?

- The function that compares graphs to each other needs to be differentiable
  $\hookrightarrow$ e. g., $g(G, G_i)$ needs to be differentiable

- Graph comparison functions are usually expensive to compute
  $\hookrightarrow$ Many problems related to it are **NP-complete**
  - subgraph isomorphism
  - finding largest common subgraph

- Message Passing Framework have proven successful in several tasks
  - Led to the discovery of a new antibiotic **[Stokes et al., Cell 180(4)]**
  - Deployed to improve Google Maps Estimated Time of Arrival **[Derrow-Pinion et al., arXiv:2108.11482]**

Open Problem: Can we design a differentiable function which is also efficient?

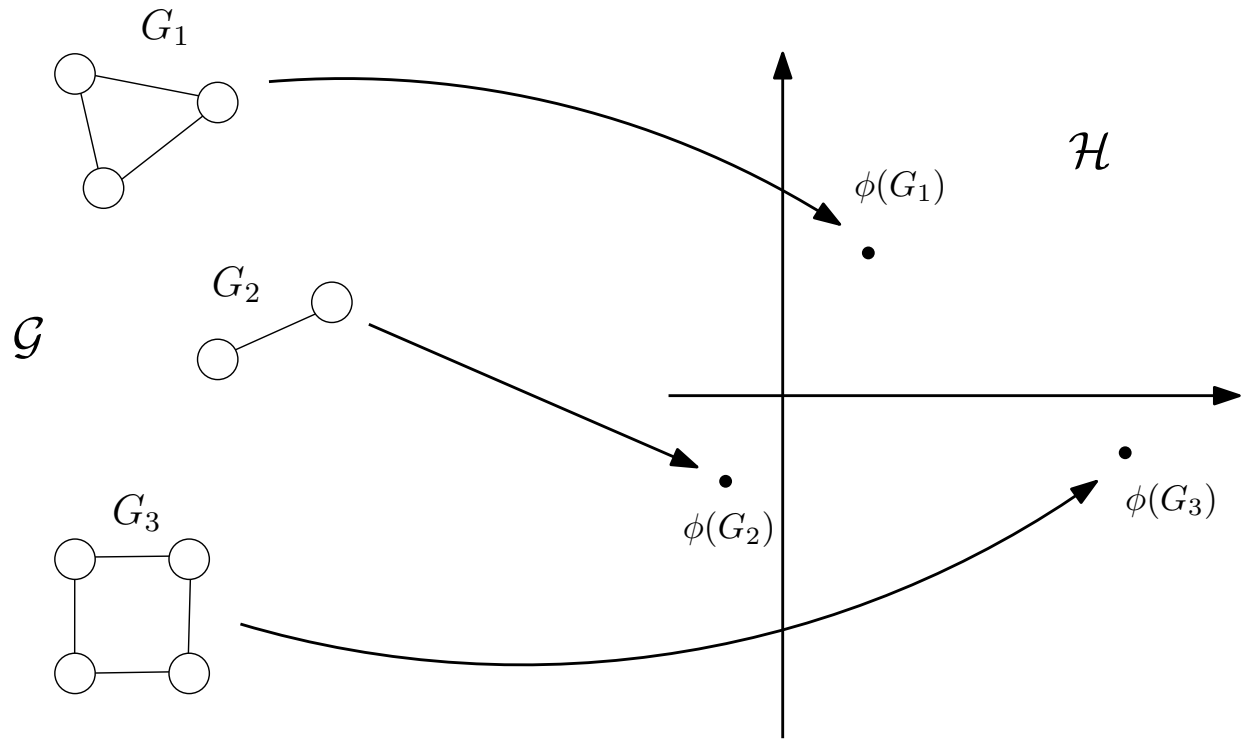
Outline

Random Walk Kernel

Definition (Graph Kernel)

A graph kernel $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is a kernel function over a set of graphs $\mathcal{G}$

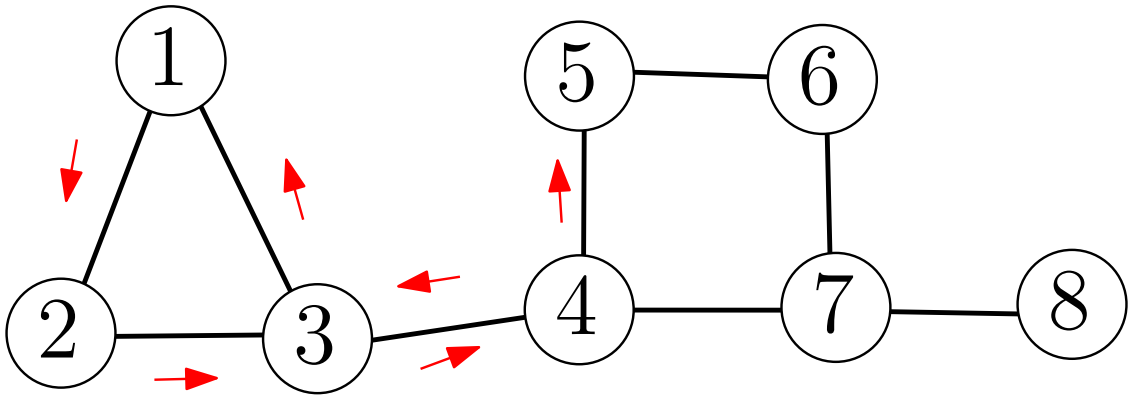
- It is equivalent to an inner product of the embeddings $\phi : \mathcal{G} \to \mathcal{H}$ of a pair of graphs into a Hilbert space, e. g., $k(G_1, G_2) = \langle \phi(G_1), \phi(G_2) \rangle$



- **Random Walk Kernel** → Counts matching walks in two graphs **[Vishwanathan et al., JMLR 11]**

Walks on Graphs

- Let $G = (V, E)$ be a graph
- A walk is a sequence of vertices $v_1, \ldots, v_k$ such that consecutive vertices in the sequence are joined by an edge, e. g., $(v_i, v_{i+1}) \in E \ \forall i \in \{1, \ldots, k-1\}$



Example of a walk: $4 \to 3 \to 1 \to 2 \to 3 \to 4 \to 5$

Product Graph

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, their direct product $G_\times$ is a graph with vertex set:

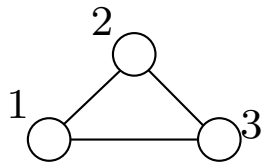
$$V_\times = \{(v_1, v_2) \in V_1 \times V_2\}$$

and edge set:

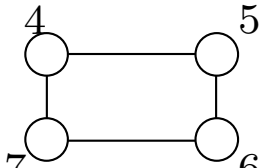
$$E_\times = \left\{((v_1, v_2), (u_1, u_2)) \in V_\times \times V_\times \mid (v_1, u_1) \in E_1, (v_2, u_2) \in E_2\right\}$$

- vertices: pairs of vertices from G_1 and G_2
- draw edge if corresponding vertices of G_1 and G_2 are adjacent in G_1 and G_2

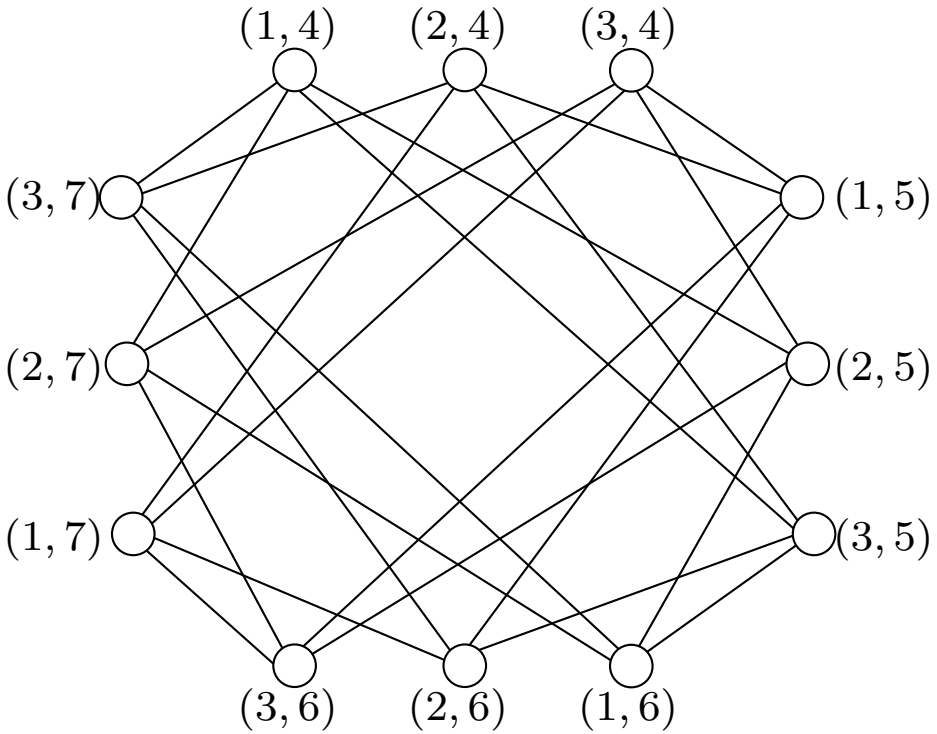
Example



G_1



G_2



$G_\times$

P-step Random Walk Kernel

- The p-th power of the adjacency matrix $\mathbf{A}$ of G computes walks of length p
  $\hookrightarrow \mathbf{A}^p_{ij}$ = number of walks of length p from vertex i to vertex j

  Example: $\mathbf{A}_2^2 = \begin{pmatrix} 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \end{pmatrix}$

- Common walks of length p can be computed using $\mathbf{A}^p_\times$ where $\mathbf{A}_\times$ is the adjacency matrix of the product graph

- For each $P \in \mathbb{N}$, the P-step random walk kernel between two graphs G and $G'$is defined as:

$$k^P(G, G') = \sum_{i=1}^{|V_\times|} \sum_{j=1}^{|V_\times|} \left[\sum_{p=0}^{P} \lambda_p \mathbf{A}^p_\times \right]_{ij}$$

  with a sequence of positive, real-valued weights $\lambda_0, \lambda_1, \ldots, \lambda_P$

P-step Random Walk Kernel

- The p-th power of the adjacency matrix $\mathbf{A}$ of G computes walks of length p
  $\hookrightarrow \mathbf{A}^p_{ij}$ = number of walks of length p from vertex i to vertex j

  Example: $\mathbf{A}_2^2 = \begin{pmatrix} 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \end{pmatrix}$

- Common walks of length p can be computed using $\mathbf{A}^p_\times$ where $\mathbf{A}_\times$ is the adjacency matrix of the product graph

- For each $P \in \mathbb{N}$, the P-step random walk kernel between two graphs G and $G'$is defined as:

$$k^P(G, G') = \sum_{i=1}^{|V_\times|} \sum_{j=1}^{|V_\times|} \left[\sum_{p=0}^{P} \lambda_p \mathbf{A}^p_\times \right]_{ij}$$

  with a sequence of positive, real-valued weights $\lambda_0, \lambda_1, \ldots, \lambda_P$

- **Observation**: the P-step random walk kernel is differentiable

Outline

Random Walk Neural Networks

The proposed model contains N "hidden graphs" $G_1, G_2, \ldots, G_N$:

- undirected graphs without self-loops
- "hidden graph" with n vertices represented by $\frac{n(n-1)}{2}$ trainable parameters
- edge weights can take only nonnegative real values (i. e. using the ReLU function)

Random Walk Neural Networks

The proposed model contains N "hidden graphs" $G_1, G_2, \ldots, G_N$:

- undirected graphs without self-loops
- "hidden graph" with n vertices represented by $\frac{n(n-1)}{2}$ trainable parameters
- edge weights can take only nonnegative real values (i. e. using the ReLU function)

The model computes the following kernel between the input graph G and each "hidden graph" G_b:

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

- For each $p \in \{0, 1, \ldots, P\}$, we obtain a different kernel value
- These kernel values can be thought of as features of the input graph
- 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 Neural Networks

The proposed model contains N "hidden graphs" $G_1, G_2, \ldots, G_N$:

- undirected graphs without self-loops
- "hidden graph" with n vertices represented by $\frac{n(n-1)}{2}$ trainable parameters
- edge weights can take only nonnegative real values (i. e. using the ReLU function)

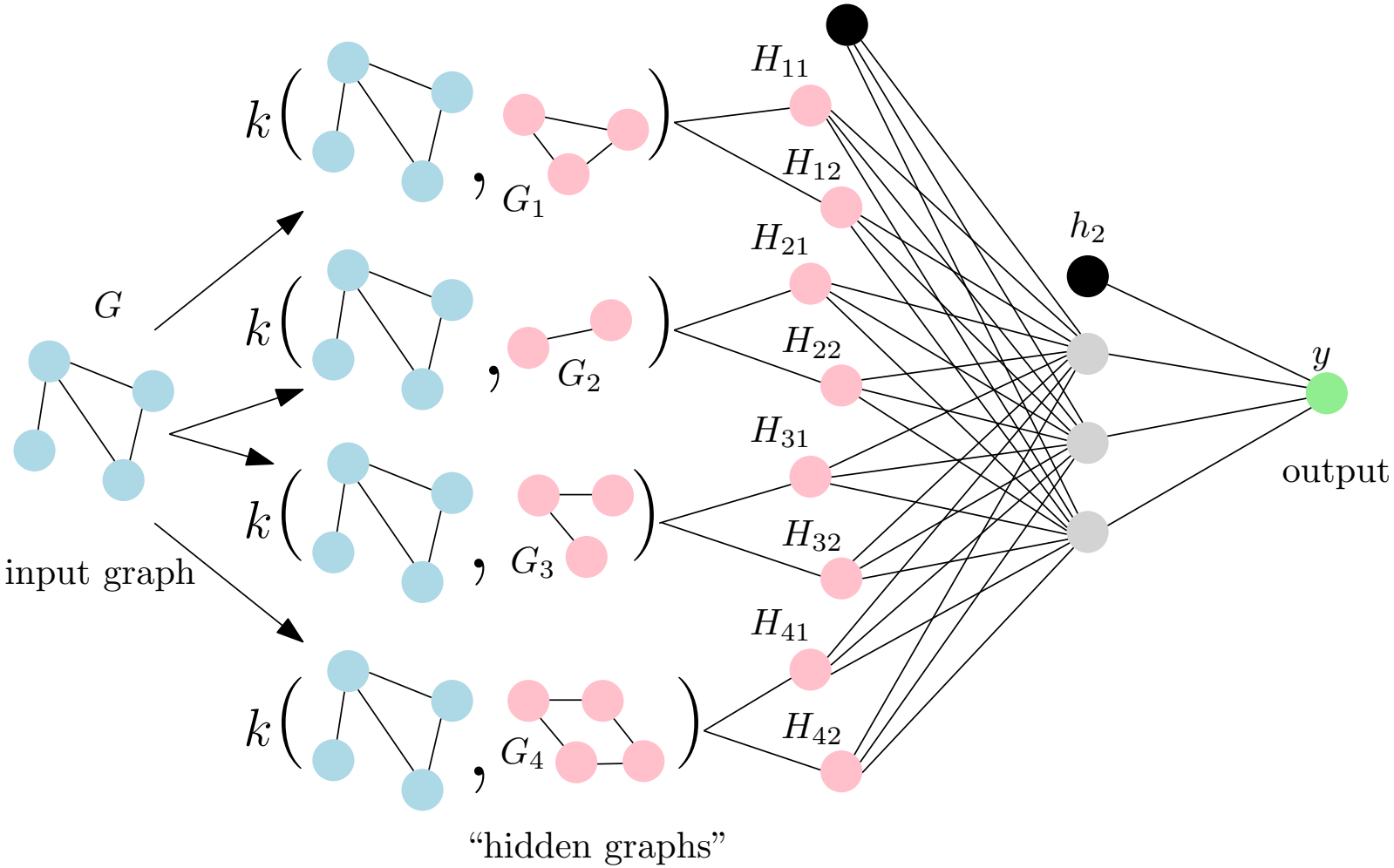
The model computes the following kernel between the input graph G and each "hidden graph" G_b:

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

- For each $p \in \{0, 1, \ldots, P\}$, we obtain a different kernel value
- These kernel values can be thought of as features of the input graph
- For each input graph G, we build a matrix $\mathbf{H} \in \mathbb{R}^{N \times P+1}$ where $\mathbf{H}_{ij} = k^{(j-1)}(G, G_i)$

Matrix $\mathbf{H}$ is flattened and fed into a fully-connected neural network to produce the output

Random Walk Neural Networks



Attributed Graphs

Real-world graphs may contain node attributes:

- chemo-informatics $\rightarrow$ chemical properties of atoms
- citation networks $\rightarrow$ bag-of-words representations of abstracts

Every node v_i in the graph is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$
$\hookrightarrow$ every graph is associated with a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ where $\mathbf{X}_{i:} = \mathbf{x}_i$

Question: Can the random walk kernel handle these attributes?

Attributed Graphs

Real-world graphs may contain node attributes:

- chemo-informatics $\rightarrow$ chemical properties of atoms
- citation networks $\rightarrow$ bag-of-words representations of abstracts

Every node v_i in the graph is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$
$\hookrightarrow$ every graph is associated with a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ where $\mathbf{X}_{i:} = \mathbf{x}_i$

Question: Can the random walk kernel handle these attributes?

- We also equip the nodes of the "hidden graphs" with attributes of the same dimensionality
- Let $\mathbf{Z}$ be the matrix that contains these attributes for some "hidden graph" G_b
- Let also $\mathbf{S} = \mathbf{Z}\,\mathbf{X}^\top$ where $\mathbf{S} \in \mathbb{R}^{n_b \times n}$
- Let also $\mathbf{s} = \mathrm{vec}(\mathbf{S})$ where $\mathbf{s} \in \mathbb{R}^{n_b n}$
- Then kernel is computed as follows:

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

Attributed Graphs

Real-world graphs may contain node attributes:

- chemo-informatics $\rightarrow$ chemical properties of atoms
- citation networks $\rightarrow$ bag-of-words representations of abstracts

Every node v_i in the graph is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$
$\hookrightarrow$ every graph is associated with a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ where $\mathbf{X}_{i:} = \mathbf{x}_i$

Question: Can the random walk kernel handle these attributes?

- We also equip the nodes of the "hidden graphs" with attributes of the same dimensionality
- Let $\mathbf{Z}$ be the matrix that contains these attributes for some "hidden graph" G_b
- Let also $\mathbf{S} = \mathbf{Z}\,\mathbf{X}^\top$ where $\mathbf{S} \in \mathbb{R}^{n_b \times n}$
- Let also $\mathbf{s} = \mathrm{vec}(\mathbf{S})$ where $\mathbf{s} \in \mathbb{R}^{n_b n}$
- Then kernel is computed as follows:

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

⚠ **However**, computational and storage complexity high

Implementation Details: Product Graphs and Kronecker Products

Definition

Given real matrices $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{p \times q}$, the Kronecker product $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{np \times mq}$ defined as:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A}_{11}\,\mathbf{B} & \mathbf{A}_{12}\,\mathbf{B} & \dots & \mathbf{A}_{1m}\,\mathbf{B} \\ \mathbf{A}_{21}\,\mathbf{B} & \mathbf{A}_{22}\,\mathbf{B} & \dots & \mathbf{A}_{2m}\,\mathbf{B} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{A}_{n1}\,\mathbf{B} & \mathbf{A}_{n2}\,\mathbf{B} & \dots & \mathbf{A}_{nm}\,\mathbf{B} \end{bmatrix}$$

- Let $\mathbf{A}$ and $\mathbf{A}'$ be the adjacency matrices of two graphs G and G'

Implementation Details: Product Graphs and Kronecker Products

Definition

Given real matrices $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{p \times q}$, the Kronecker product $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{np \times mq}$ defined as:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A}_{11}\,\mathbf{B} & \mathbf{A}_{12}\,\mathbf{B} & \dots & \mathbf{A}_{1m}\,\mathbf{B} \\ \mathbf{A}_{21}\,\mathbf{B} & \mathbf{A}_{22}\,\mathbf{B} & \dots & \mathbf{A}_{2m}\,\mathbf{B} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{A}_{n1}\,\mathbf{B} & \mathbf{A}_{n2}\,\mathbf{B} & \dots & \mathbf{A}_{nm}\,\mathbf{B} \end{bmatrix}$$

- Let $\mathbf{A}$ and $\mathbf{A}'$ be the adjacency matrices of two graphs G and G'
- Let also $\mathbf{A}_\times$ be the adjacency matrix of their direct product $G_\times$

Implementation Details: Product Graphs and Kronecker Products

Definition

Given real matrices $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{p \times q}$, the Kronecker product $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{np \times mq}$ defined as:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A}_{11}\,\mathbf{B} & \mathbf{A}_{12}\,\mathbf{B} & \dots & \mathbf{A}_{1m}\,\mathbf{B} \\ \mathbf{A}_{21}\,\mathbf{B} & \mathbf{A}_{22}\,\mathbf{B} & \dots & \mathbf{A}_{2m}\,\mathbf{B} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{A}_{n1}\,\mathbf{B} & \mathbf{A}_{n2}\,\mathbf{B} & \dots & \mathbf{A}_{nm}\,\mathbf{B} \end{bmatrix}$$

- Let $\mathbf{A}$ and $\mathbf{A}'$ be the adjacency matrices of two graphs G and G'
- Let also $\mathbf{A}_\times$ be the adjacency matrix of their direct product $G_\times$
- Then, $\mathbf{A}_\times = \mathbf{A} \otimes \mathbf{A}'$ holds

Properties of Kronecker Product

Property 1: The Kronecker product and vec operator are linked by the well-known property:

$$\text{vec}(\mathbf{ABC}) = (\mathbf{C}^\top \otimes \mathbf{A})\text{vec}(\mathbf{B})$$

Property 2: Given matrices $\mathbf{A}, \text{vec}^{-1}(\mathbf{b})$ and $\mathbf{C}$, the above equation can be written as:

$$\text{vec}\big(\mathbf{A}\,\text{vec}^{-1}(\mathbf{b})\,\mathbf{C}\big) = (\mathbf{C}^\top \otimes \mathbf{A})\,\text{vec}\big(\text{vec}^{-1}(\mathbf{b})\big) = (\mathbf{C}^\top \otimes \mathbf{A})\mathbf{b}$$

vec: maps a matrix into a vector by stacking its columns
vec^{-1}: inverse vectorization operator which transforms a vector into a matrix

Implementation Details: Kernel Computation

Kernel for $p = 1$:

$$k^{(1)}(G, G_b) = \sum_{i=1}^{|V_\times|} \sum_{j=1}^{|V_\times|} s_i s_j \left[\mathbf{A}_\times\right]_{ij} = \mathbf{s}^\top \mathbf{A}_\times \mathbf{s}$$

$$= \mathbf{s}^\top \big(\mathbf{A} \otimes \mathrm{ReLU}(\mathbf{A}_b)\big)\mathbf{s}$$
$$= \mathbf{s}^\top (\mathbf{A}^\top \otimes \mathrm{ReLU}(\mathbf{A}_b))\mathbf{s}$$
$$= \mathbf{s}^\top \mathrm{vec}\big(\mathrm{ReLU}(\mathbf{A}_b)\,\mathrm{vec}^{-1}(\mathbf{s})\,\mathbf{A}\big)$$

We can generalize since $\mathbf{A}_\times^p = \mathbf{A}_1^p \otimes \mathbf{A}_2^p$ holds for all $p \in \mathbb{N}$:

$$k^{(p)}(G, G_b) = \sum_{i=1}^{|V_\times|} \sum_{j=1}^{|V_\times|} s_i s_j \left[\mathbf{A}_\times^p\right]_{ij} = \mathbf{s}^\top \mathrm{vec}\big(\mathrm{ReLU}(\mathbf{A}_b)^p\,\mathrm{vec}^{-1}(\mathbf{s})\,\mathbf{A}^p\big)$$

$$= \mathbf{s}^\top \mathrm{vec}\big(\mathrm{ReLU}(\mathbf{A}_b)^p\,\mathbf{S}\,\mathbf{A}^p\big)$$
$$= \mathbf{s}^\top \mathrm{vec}\big(\mathrm{ReLU}(\mathbf{A}_b)^p\,\mathbf{Z}\,\mathbf{X}^\top\,\mathbf{A}^p\big)$$
$$= \mathrm{vec}(\mathbf{S})^\top \mathrm{vec}\big(\mathrm{ReLU}(\mathbf{A}_b)^p\,\mathbf{Z}\,\mathbf{X}^\top\,\mathbf{A}^p\big)$$
$$= \mathrm{vec}(\mathbf{Z}\,\mathbf{X}^\top)^\top \mathrm{vec}\big(\mathrm{ReLU}(\mathbf{A}_b)^p\,\mathbf{Z}\,\mathbf{X}^\top\,\mathbf{A}^p\big)$$
$$= \sum_{i=1}^{c} \sum_{j=1}^{n} \Big[\mathbf{Z}\,\mathbf{X}^\top \odot \big(\mathrm{ReLU}(\mathbf{A}_b)^p\,\mathbf{Z}\,\mathbf{X}^\top\,\mathbf{A}^p\big)\Big]_{ij}$$

Implementation Details: Updating Node Attributes

To reduce the dimensionality of the vertex attributes or to make the model more complex, we can:

- use a 1-layer perceptron as follows:

$$\tilde{\mathbf{X}} = f(\mathbf{X}\,\mathbf{W} + \mathbf{b})$$

  where $\mathbf{W} \in \mathbb{R}^{d \times \tilde{d}}$, $\mathbf{b} \in \mathbb{R}^{\tilde{d}}$ is a weight matrix and bias vector, respectively, and f is a non-linear activation function

- use an Message Passing Neural Network to update these vertex representations:

$$\tilde{\mathbf{X}} = \mathrm{MPNN}(\mathbf{A}, \mathbf{X})$$

Outline

Experiments

Two types of datasets:

- Synthetic datasets $\rightarrow$ interpretability of the model
- Real-world datasets $\rightarrow$ performance of the model

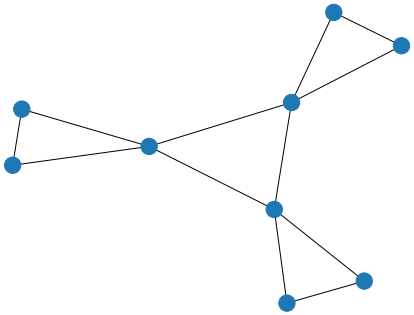
Synthetic Datasets

Synthetic of **5** binary classification datasets, each containing **1000** graphs

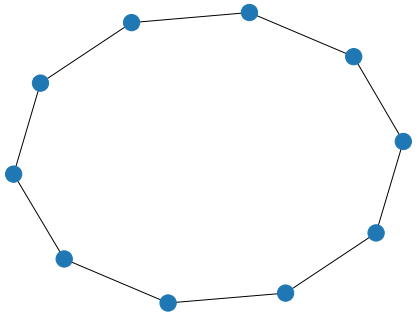
Generation process:

1. An Erdös-Rényi graph is generated:
   - number of nodes randomly drawn from $[100, \ldots, 200]$ with uniform probability
   - edge probability set equal to 0.1
2. A positive and a negative instance is generated by planting the following structures on the generated graph:
   - a dataset-specific special structure (class 0)
   - an Erdös-Rényi graph with the same number of nodes and similar density to the special structure (class 1)
3. Nodes of the structures are connected to nodes of the Erdös-Rényi graph with probability 0.05

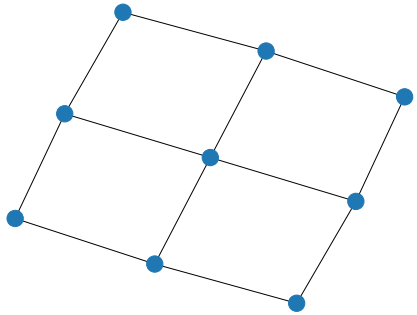
Special Structures



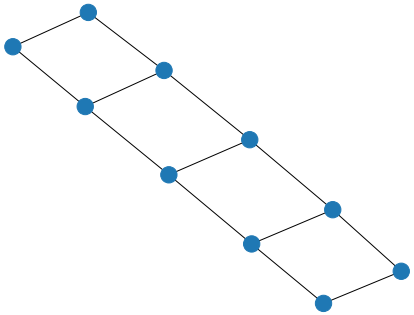
Caveman graph



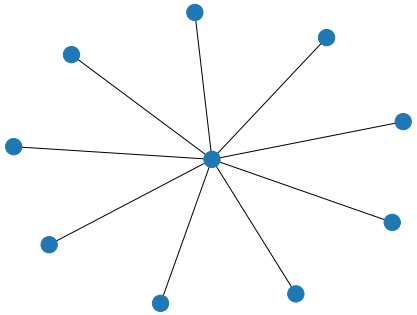
Cycle graph



Grid graph



Ladder graph



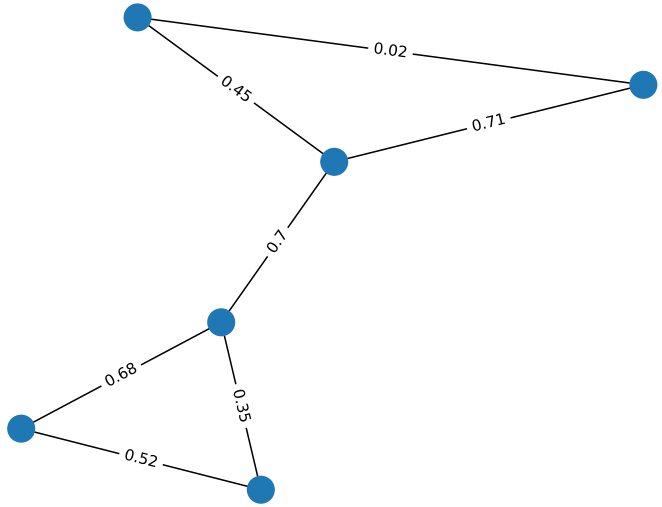
Star graph

Experimental Setup

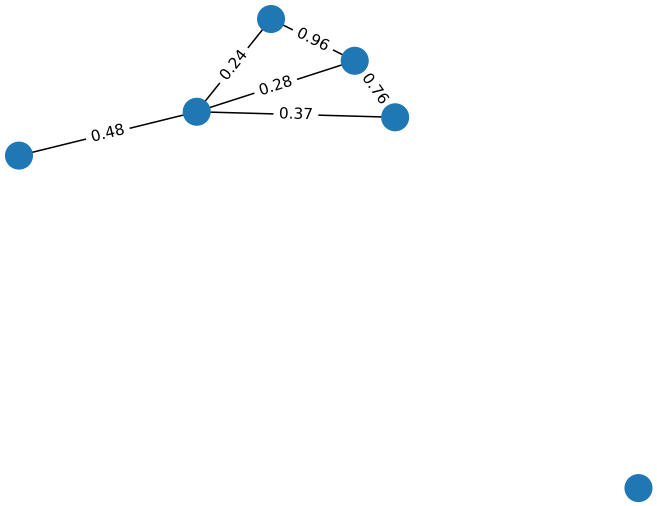
- Train/val set: 90%/10% of each dataset
- No of epochs: 50
- Learning rate: 10^{-3}
- Size of "hidden graphs": 6 nodes
- ~~node attributes~~
- Hyperparameter P: 3
- Store and retrieve the model that performed best on val set

Task: Extract and visualize "hidden graphs"

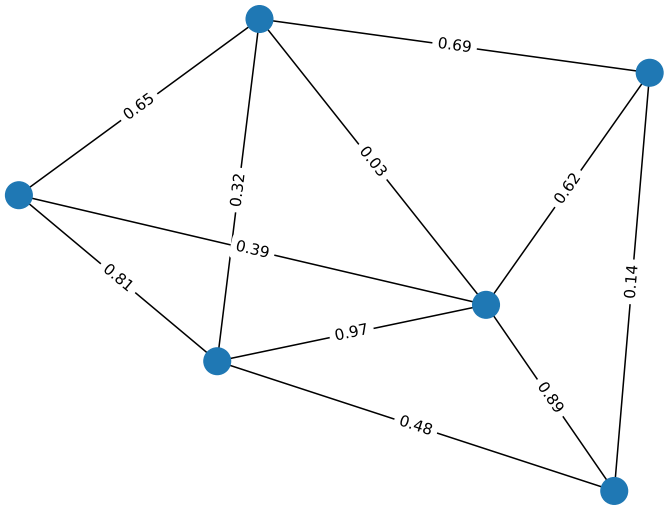
Caveman Graph



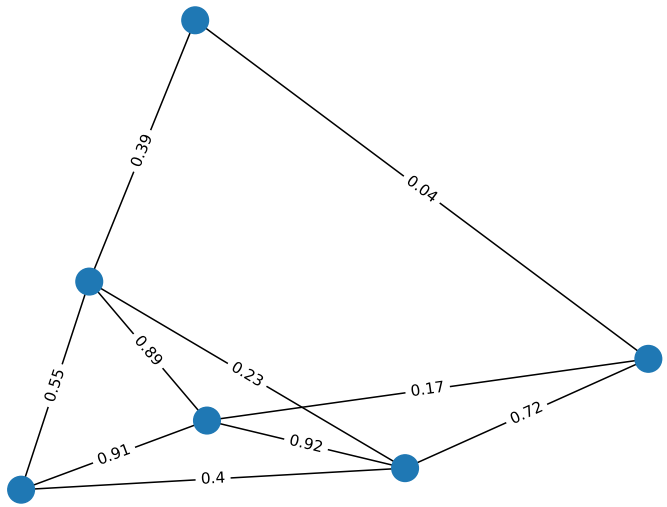
"hidden graph" #1



"hidden graph" #2

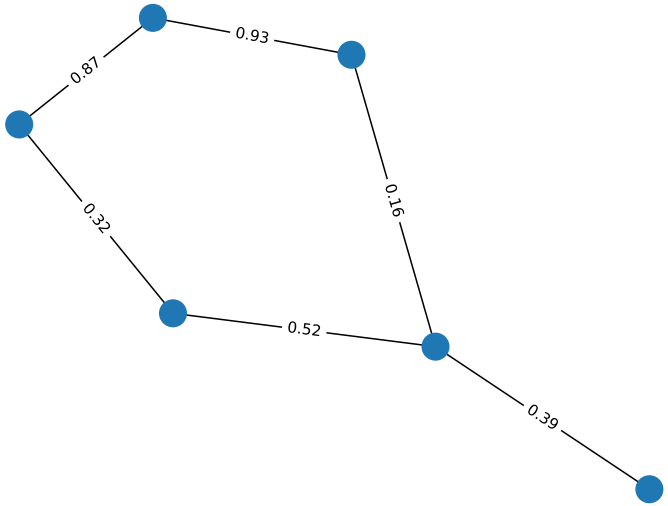


"hidden graph" #3

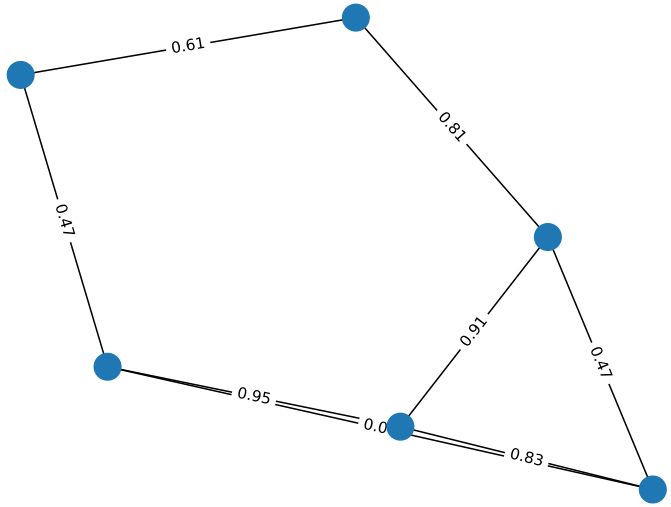


"hidden graph" #4

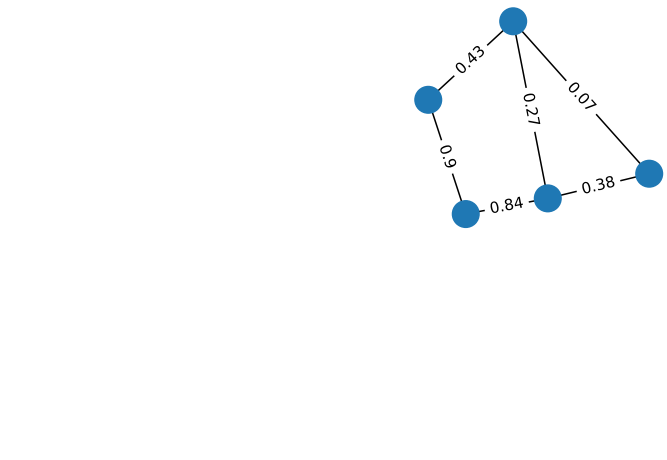
Cycle Graph



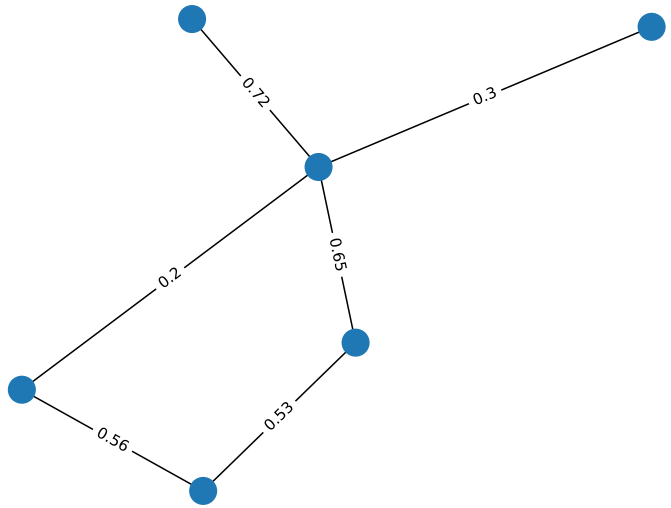
"hidden graph" #1



"hidden graph" #2

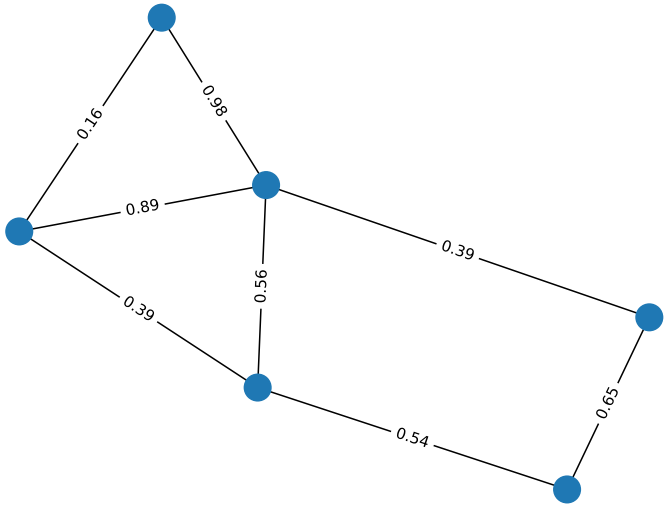


"hidden graph" #3

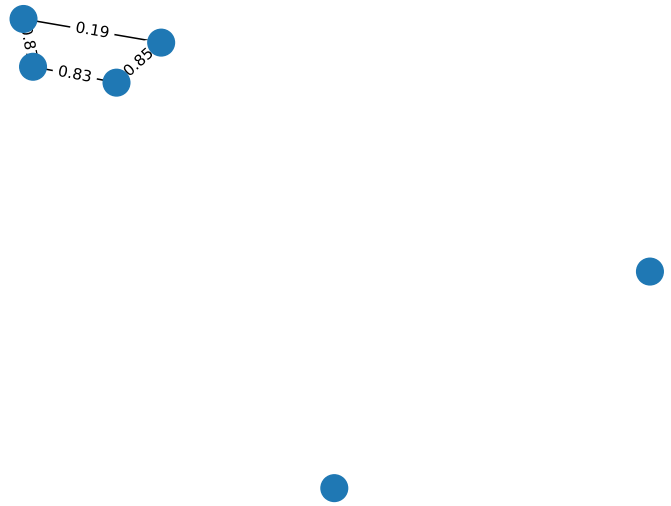


"hidden graph" #4

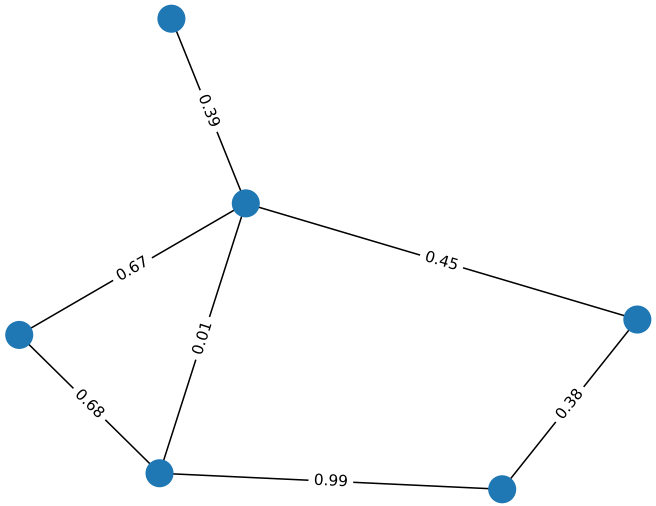
Ladder Graph



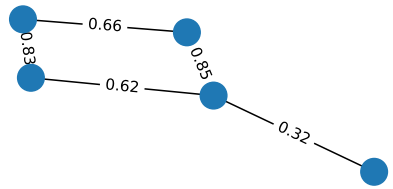
“hidden graph” #1



“hidden graph” #2

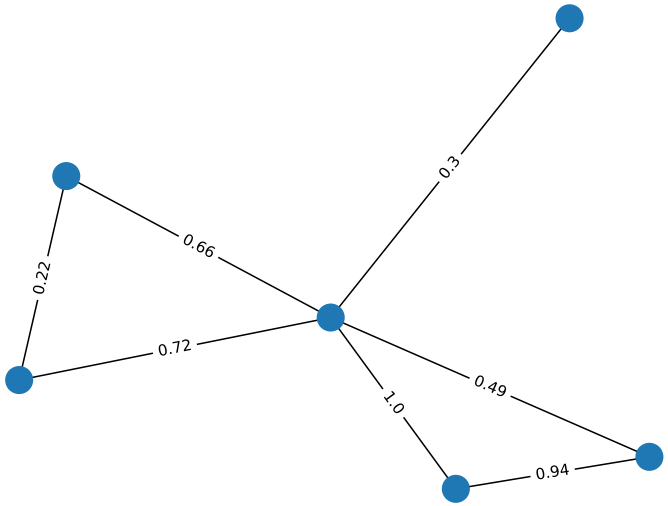


“hidden graph” #3

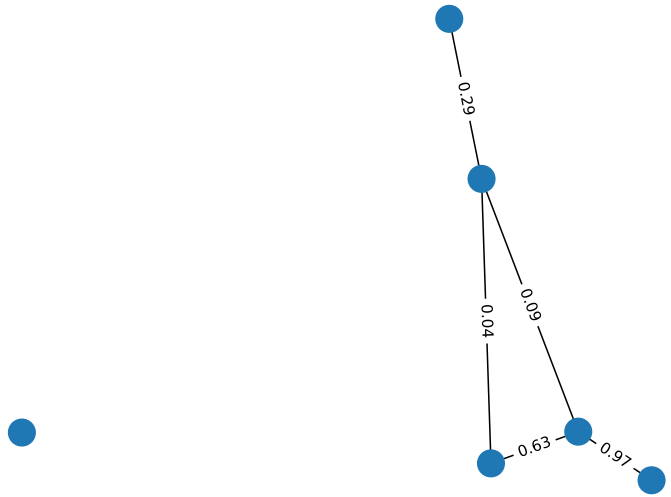


“hidden graph” #4

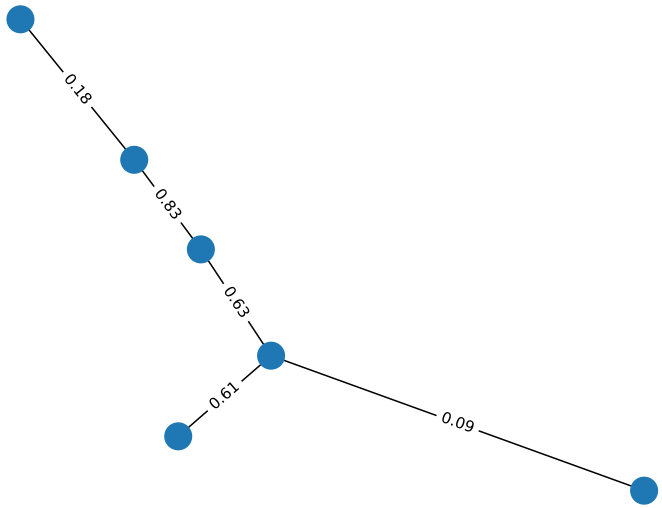
Star Graph



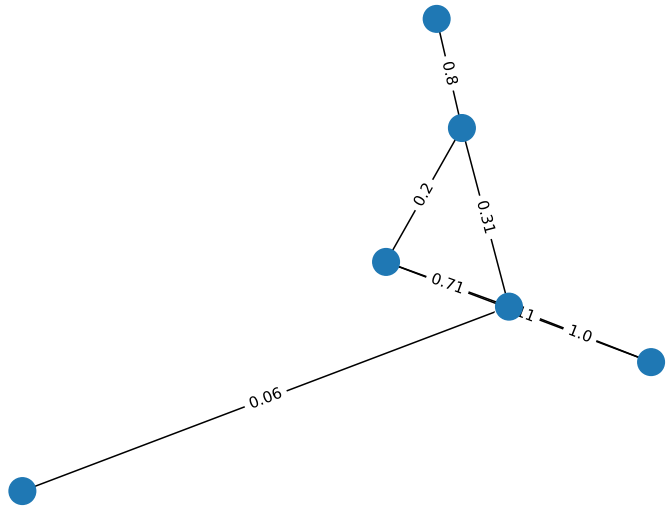
“hidden graph” #1



“hidden graph” #2

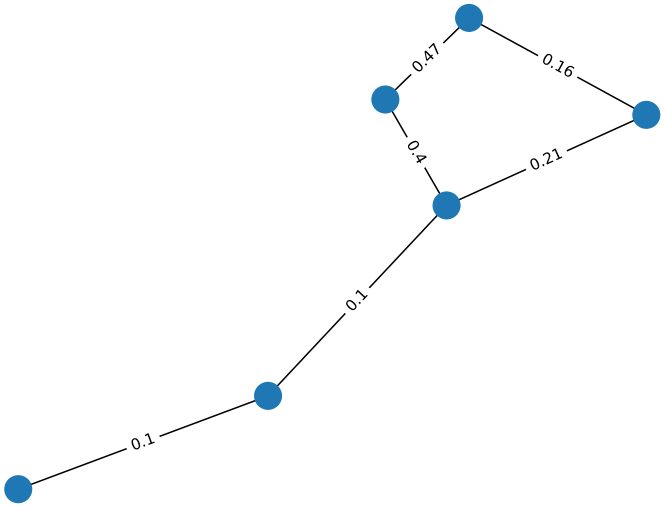


“hidden graph” #3

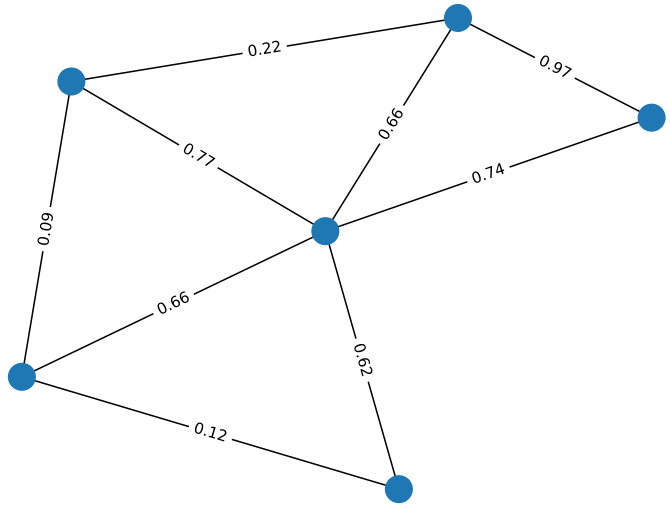


“hidden graph” #4

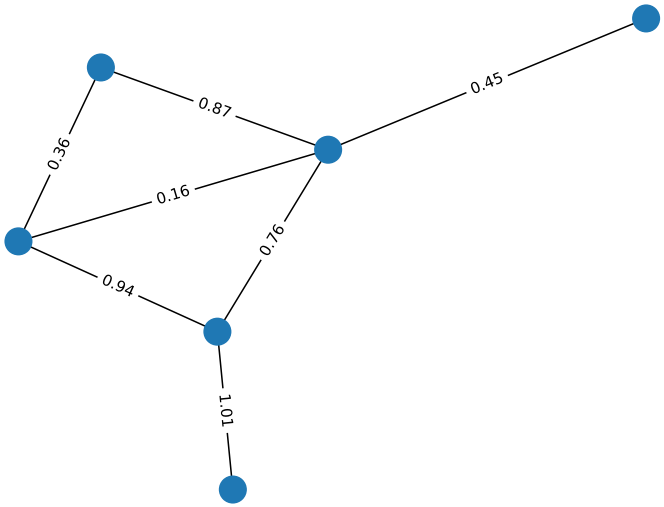
Grid Graph



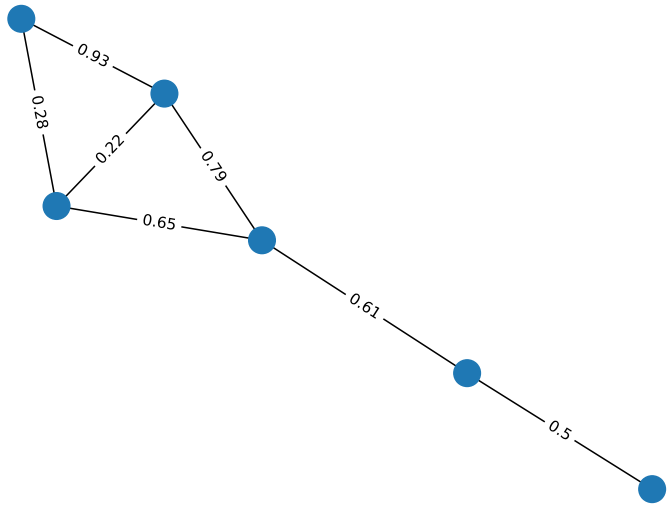
"hidden graph" #1



"hidden graph" #2



"hidden graph" #3



"hidden graph" #4

Real-World Datasets

Dataset	MUTAG	D&D	NCI1	PROTEINS	ENZYMES	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	COLLAB
Max # vertices	28	5,748	111	620	126	136	89	3,782	3,648	492
Min # vertices	10	30	3	4	2	12	7	6	22	32
Average # vertices	17.93	284.32	29.87	39.05	32.63	19.77	13.00	429.61	508.50	74.49
Max # edges	33	14,267	119	1,049	149	1,249	1,467	4,071	4,783	40,119
Min # edges	10	63	2	5	1	26	12	4	21	60
Average # edges	19.79	715.66	32.30	72.81	62.14	96.53	65.93	497.75	594.87	2,457.34
# labels	7	82	37	3	-	-	-	-	-	-
# attributes	-	-	-	-	18	-	-	-	-	-
# graphs	188	1,178	4,110	1,113	600	1,000	1,500	2,000	4,999	5,000
# classes	2	2	2	2	6	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 (1/2)

	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)

Graph Classification - Real World Datasets (2/2)

	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)

Outline

Summary

- Message Passing Neural Networks successful but not very intuitive
- Proposed Random Walk Graph Neural Network, a novel neural network model, which employs a random walk kernel to learn *graph features* that contribute to interpretable/intuitive graph representations
- Also proposed an efficient computation scheme to reduce its time and space complexity
- Achieved high levels of performance on standard graph classification datasets

Thank you!