Lancaster University

Introducing Graph Neural Networks

Pascal Welke

30.4.2025

Topics Today

Graph Representation Learning

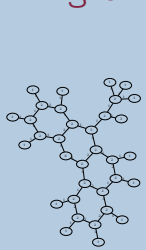
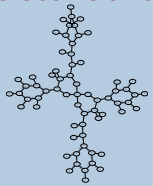
The Weisfeiler Leman (WL) Algorithm

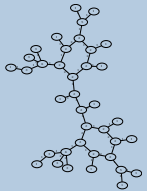
(Message Passing) Graph Neural Networks

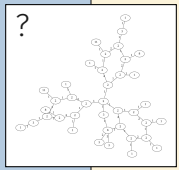
- Graph Neural Networks for Transactional Graph Learning
- The Expressive Power of Graph Neural Networks

Extensions to Message Passing

Learning on structured data

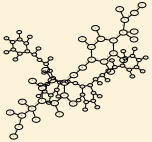
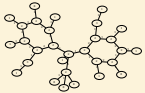
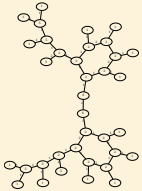







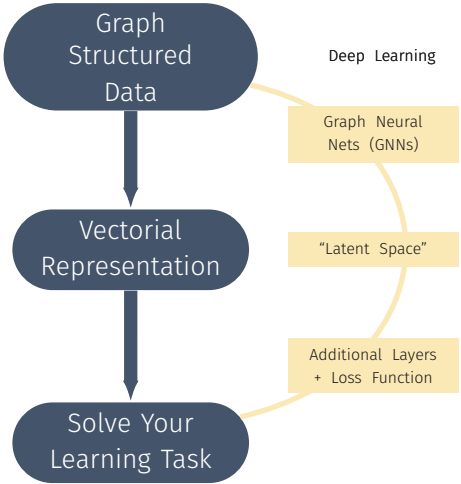
Hu et al (2020) Morris et al (2020) Dwivedi et al (2022)



Graph Representation Learning

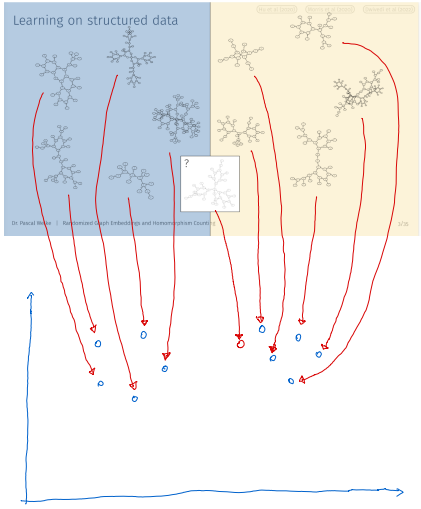
Graph representation learning



The goal

Vectorial graph representations that are adaptable to given data

In particular, to the target labels on training data



The problem with vectorial graph representations

We want our graph representation function ϕ to be

- permutation-invariant

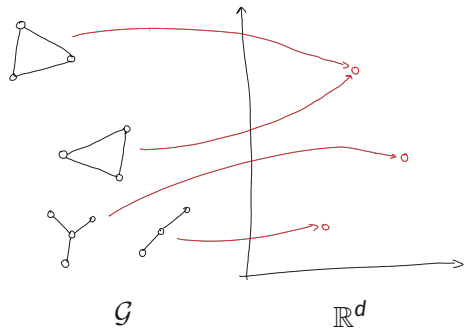
  for all isomorphic graphs

  $$G \simeq H : \ \phi(G) = \phi(H)$$

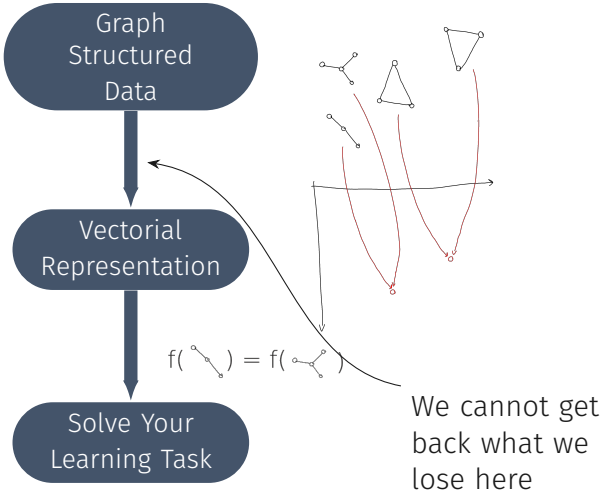
- complete

  for all non-isomorphic graphs

  $$G \not\simeq H : \ \phi(G) \neq \phi(H)$$



Why do we care?



- Unfortunately computing any permutation invariant and complete embedding (or kernel) is as hard as deciding graph isomorphism
- **Typical solution**: drop completeness for efficiency
  - most practical graph kernels, GNNs, Weisfeiler Leman test, …

Isomorphism-Hardness

Theorem

Computing a permutation invariant and complete representation for some graph class $\mathcal{G}$ is Isomorphism-hard.

- We don't know the complexity of Graph Isomorphism
- No polynomial time algorithm is known
- It is not known to be NP-hard

Implications

- Does this mean we cannot *learn* a complete representation for the class of all graphs?
- Well, supervised learning from a fixed finite training dataset can be seen as *constant time preprocessing*, i.e. it takes $O(1)$ time.
- Thus, if we could learn a *complete* representation c that is *computable in polynomial time* for any $G \in \mathcal{G}$ then this would imply a polynomial time algorithm for the isomorphism problem on $\mathcal{G}$
- Still, this could work with the right tools

What about Graph Neural Networks?

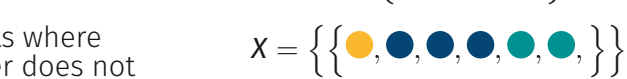
- GNNs promise to learn “suitable” graph representations for “any” learning task
- Can we use them to learn complete representations for the class of all graphs?

The Weisfeiler Leman (WL) Algorithm

A Quick Note on Multisets

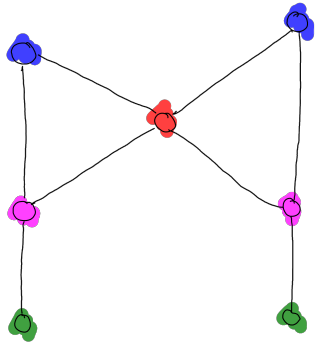
- *Multisets* are collections of objects where multiplicity *does* matter, and order does not matter
  - A multiset X of objects from $\mathcal{X}$ contains each object in $\mathcal{X}$ a nonnegative number of times
  - Hence it makes sense to write $\mathbb{N}^{\mathcal{X}}$ to denote the set of all multisets of objects from $\mathcal{X}$
  - We can represent a multiset X of objects from $\mathcal{X}$ as vector $h_{\mathcal{X}}(X) \in \mathbb{R}^{|\mathcal{X}|}$

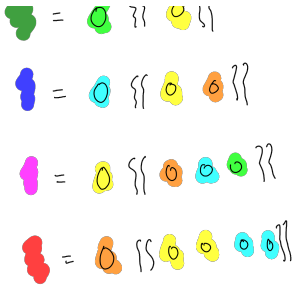
$\mathcal{X} =$

$X =$ 

$h_{\mathcal{X}}(X) =$ 

WL Label Propagation Scheme (1)





$k = 2$

WL Label Propagation Scheme (2)

Let $\mathcal{G}$ be a graph class, $G \in \mathcal{G}$, and $v \in V(G)$.

Then the *Weisfeiler Leman label* of v in iteration $k > 0$ is defined as

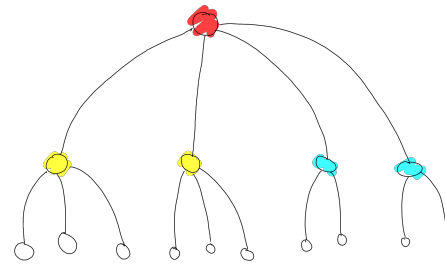
$$r_k^{WL}(v) = \text{hash}_{k-1}\left(r_{k-1}^{WL}(v), \left\{\left\{r_{k-1}^{WL}(w) \mid w \in N(v)\right\}\right\}\right)$$

Where

- $r_0^{WL} := f_V : \dot{\bigcup}_{G \in \mathcal{G}} V(G) \to \mathcal{X}'$ gives the "original" vertex labels
- $\text{hash}_{k-1} : \mathcal{X}_{k-1} \times \mathbb{N}^{\mathcal{X}_{k-1}} \to \mathcal{X}_k$ is an injective function (also called a perfect hash function)

What Does This Do?

- Why would assigning new colors to vertices help us in any way?
- Well, the new colors encode neighborhood information
- They, to a certain extend, remember the k-hop neighborhood of a vertex in iteration k



Result of Weisfeiler and Leman

Theorem

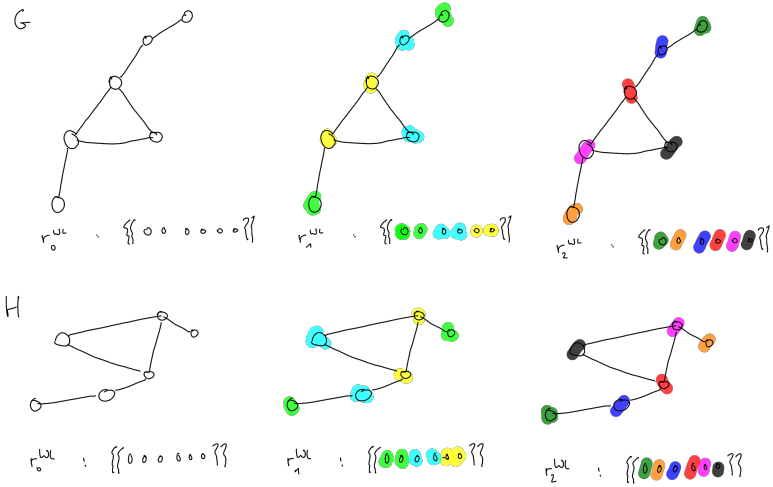
Let $G_1, G_2 \in \mathcal{G}$ be two graphs. If G_1 and G_2 are isomorphic (respecting f_V), then

$$\left\{\left\{r_k^{WL}(v) \mid v \in V(G_1)\right\}\right\} = \left\{\left\{r_k^{WL}(v) \mid v \in V(G_2)\right\}\right\}$$

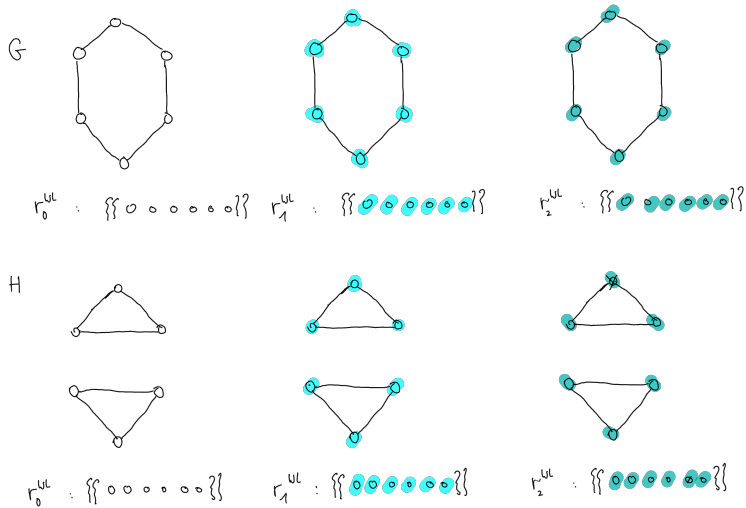
for all $k \geq 0$.

Weisfeiler and Leman (1968)

Isomorphic Graphs



Nonisomorphic Graphs



Tractable Incomplete Representations

- We now know that Weisfeiler Leman representations are incomplete on the class of all graphs
- There are “very few” nonisomorphic graphs with identical Weisfeiler Leman representations Cai et al (1992)

The WL Algorithm and Graph Learning

- A cumulative version of WL combined with SVMs is typically a strong baseline for learning on graphs
- Usually, $k = 5$ even suffices
- The WL kernel is easy to implement and fast to compute

(Message Passing) Graph Neural Networks

Basic GNN

$$r_{k+1}(v) = \mathrm{upd}_k\left(r_k(v),\ \mathrm{agg}_k\left(\{\{r_k(w) \mid w \in N(v)\}\}\right)\right)$$

In its oldest form, a *graph neural network layer in iteration* k is defined for all $v \in V(G)$ as follows: Scarselli et al (2009)

$$r_{k+1}(v) = \sigma\left(W_k^{\text{self}} r_k(v) + W_k^{\text{neigh}} \sum_{w \in N(v)} r_k(w) + b_k\right)$$

- $W_k^{\text{self}}, W_k^{\text{neigh}} \in \mathbb{R}^{d_k \times d_{k+1}}$ are trainable parameter matrices
- $\sigma : \mathbb{R}^{d_k+1} \to \mathbb{R}^{d_k+1}$ is an element-wise nonlinear function, e.g. tanh, ReLU, sigmoid
- $b \in \mathbb{R}^{d_{k+1}}$ are trainable bias parameters

Stay Tuned for W and b

- Let's assume for now, that we know how to choose $\boldsymbol{W}_k^{\text{self}}$, $\boldsymbol{W}_k^{\text{neigh}}$, and $\boldsymbol{b} \in \mathbb{R}^{d_{k+1}}$
- We will discuss how to learn these beasts in the next section
- For now, in neural network lingo, we are only interested in the forward pass
- (in message passing lingo, we are interested in different definitions of the agg and upd functions)

Definition

We will use the notation θ_k for all trainable weights in iteration k of GNN. We will interpret this as a flattened vector, i.e. on the previous slide $\theta_k \in \mathbb{R}^{2d_k d_{k+1} + d_{k+1}}$

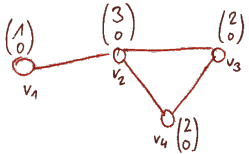
Reformulation in terms of agg and upd

$$\mathrm{agg}_k(\{\{r_k(w) \mid w \in N(v)\}\}) = \sum_{w \in N(v)} r_k(w)$$

$$\mathrm{upd}_k = W_k^{\mathrm{self}} r_k(v) + W_k^{\mathrm{neigh}} \, \mathrm{agg}_k(\cdot) + b_{k+1}$$

- Note that the aggregator is dead simple (it sums up messages)
- The interesting stuff happens in the update function
- This ensures permutation invariance of agg

Example



$\sigma = id$

$W_i^{self} = \begin{pmatrix}1&1\\0&1\end{pmatrix}$

$W_i^{neigh} = \begin{pmatrix}1&1\\1&0\end{pmatrix}$

$b_i = \begin{pmatrix}0\\0\end{pmatrix}$

$$r_1(v_1) = \begin{pmatrix}1&1\\0&1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} + \begin{pmatrix}1&1\\1&0\end{pmatrix}\left(\begin{pmatrix}3\\0\end{pmatrix}\right) + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix} + \begin{pmatrix}3\\3\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}4\\3\end{pmatrix}$$

$$r_1(v_2) = \begin{pmatrix}1&1\\0&1\end{pmatrix}\begin{pmatrix}3\\0\end{pmatrix} + \begin{pmatrix}1&1\\1&0\end{pmatrix}\left(\begin{pmatrix}1\\0\end{pmatrix} + \begin{pmatrix}2\\0\end{pmatrix} + \begin{pmatrix}2\\0\end{pmatrix}\right) + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}3\\0\end{pmatrix} + \begin{pmatrix}5\\5\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}8\\5\end{pmatrix}$$

$$r_1(v_3) = r_1(v_4) = \begin{pmatrix}1&1\\0&1\end{pmatrix}\begin{pmatrix}2\\0\end{pmatrix} + \begin{pmatrix}1&1\\1&0\end{pmatrix}\left(\begin{pmatrix}2\\0\end{pmatrix} + \begin{pmatrix}3\\0\end{pmatrix}\right) + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}2\\0\end{pmatrix} + \begin{pmatrix}5\\5\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}7\\5\end{pmatrix}$$

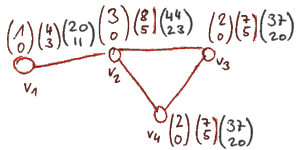
$$r_2(v_1) = \begin{pmatrix}1&1\\0&1\end{pmatrix}\begin{pmatrix}4\\3\end{pmatrix} + \begin{pmatrix}1&1\\1&0\end{pmatrix}\left(\begin{pmatrix}8\\5\end{pmatrix}\right) + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}7\\3\end{pmatrix} + \begin{pmatrix}13\\8\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}20\\11\end{pmatrix}$$

$$r_2(v_2) = \begin{pmatrix}1&1\\0&1\end{pmatrix}\begin{pmatrix}8\\5\end{pmatrix} + \begin{pmatrix}1&1\\1&0\end{pmatrix}\left(\begin{pmatrix}4\\3\end{pmatrix} + \begin{pmatrix}7\\5\end{pmatrix} + \begin{pmatrix}7\\5\end{pmatrix}\right) + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}13\\5\end{pmatrix} + \begin{pmatrix}31\\18\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}44\\23\end{pmatrix}$$

$$r_2(v_3) = r_2(v_4) = \begin{pmatrix}1&1\\0&1\end{pmatrix}\begin{pmatrix}7\\5\end{pmatrix} + \begin{pmatrix}1&1\\1&0\end{pmatrix}\left(\begin{pmatrix}7\\5\end{pmatrix} + \begin{pmatrix}8\\5\end{pmatrix}\right) + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}12\\5\end{pmatrix} + \begin{pmatrix}25\\15\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}37\\20\end{pmatrix}$$

Observations

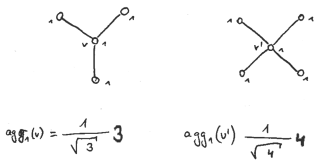
- Note that the numbers in our example tended to grow large rather quickly
- In particular, one might expect that the representations of high degree nodes will contain large values, with a large difference to the values of representations of low degree nodes
- this might lead to numerical instabilities and difficulties in optimization of the loss function



Variants

- We can use the mean aggregator instead of the sum aggregator
- this keeps the values in the computation of high degree vertices at bay
- it should also limit the influence of individual nodes in the training
- another suggestion is to use the *symmetric normalization* aggregator

$$\text{agg}_k(\cdot) = \sum_{w \in N(v)} \frac{r_k(w)}{\sqrt{|N(v)|\,|N(w)|}}$$



Graph Convolutional Networks (GCNs)

- A very popular variant of GNNs is the *Graph Convolutional Neural Network*
  (Kipf and Welling (2016))
- It uses symmetric normalization
- It does not distinguish between representations of v and its neighbors

$$r_{k+1}(v) = \text{reLU}\left(W_k \sum_{w \in N(v) \cup \{v\}} \frac{r_k(w)}{\sqrt{|N(v)|\,|N(w)|}} + b_k\right)$$

- again, $W \in \mathbb{R}^{d_k \times d_{k+1}}$ and $b \in \mathbb{R}^{d_{k+1}}$ are trainable parameters

Possible Issues with GCNs

- GCNs likely lose some (important) information
  - by construction, GCNs can never assign a different weight to the former representation of v
  - hence GCNs cannot distinguish between v and its neighbor
- The symmetric normalization (partially) inhibits to learn from the degree of v

Graph Isomorphism Networks (GINs)

- Another variant adds a multiple layer trainable neural network in each message passing step (Xu et al (2019))
- Let NN_{θ_k} be a (multilayer) neural network (with trainable weights and element-wise non-linearities)
- Let $\epsilon_k \in \mathbb{R}$ be a trainable weight

$$r_{k+1}(v) = NN_{\theta_k}\left((1+\epsilon_k)r_k(v) + \sum_{w \in N(v)} r_k(w) + b_k\right)$$

Possible Issues with GINs

- GINs give you a lot of freedom to choose:
- How many layers should your MLP have?
- How many hidden units per hidden layer?
- Which non-linearity should be chosen?
- All these choices will be hyperparameters in learning...
  - the authors suggest to choose a small MLP
  - with one or two hidden layers
  - 16 or 32 hidden units per layer (for low-dimensional initial node representations)
  - reLU activation functions

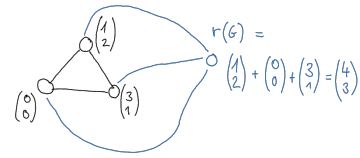
(Message Passing) Graph Neural Networks | Graph Neural Networks for Transactional Graph Learning

GNNs for Transactional Graph Learning

- We have now discussed different variants of neural networks that can be used as update and aggregator functions in the message passing framework
- How can we train these neural networks (weights) in a graph classification setting?
- ...well, as usual with
  - (stochastic) gradient descent,
  - back-propagation,
  - and a differentiable loss function

Graph Pooling

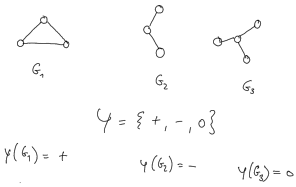
- The step from individual node representations to a graph representation is called *pooling*
- Given vertex representations $\{\{r_k(v) \mid v \in V(G)\}\}$ we can, for example, use
  - sum pooling: $r(G) := \sum_{v \in V(G)} r_k(v)$
  - mean pooling: $r(G) := \mathit{mean}\,(\{\{r_k(v) \mid v \in V(G)\}\})$
  - max pooling: $r(G) := \max(\{\{r_k(v) \mid v \in V(G)\}\})$



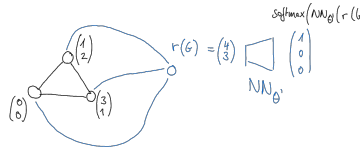
Graph Classification with Neural Networks

- Neural Networks for classification tasks are often trained with the *negative log likelihood* loss function of a *softmax classification*

$$L_{G,y}(h_\theta) = \sum_{G \in D} -\log\left(\vec{y}(G)^T \operatorname{softmax}(r(G))\right)$$



- $\vec{y}(G) \in \mathbb{1}^d$ is a one-hot vector encoding of the class of $G \in \mathcal{G}$,
  - for G with $y(G) = c_i$, we set $\vec{y}(G) = \vec{e}_i$, i.e., the ith unit vector
- $r(G) \in \mathbb{R}^d$ is the last hidden representation of G



One Epoch of Training

Given Training set $D \subseteq \mathcal{G}$, $y : D \to C$, learning rate γ, number of message passing steps k, initial vertex representation function r_0

Output Updated weights $\theta_0, \ldots, \theta_{k-1}, \theta_{graph}, \theta'$

1: **for** $G \in D$ **do**
2: Run k iterations of your graph neural network message passing for all $v \in V(G)$
3: Aggregate the vertex representations of all vertices in G and update the result to obtain $r(G)$
4: Apply $NN_{\theta'}$ to the resulting representation $r(G)$
5: Compute $L_y(h_{k,\theta_0,\ldots,\theta_{k-1},\theta_{graph},\theta'})$
6: Set $(\theta_0, \ldots, \theta_{k-1}, \theta') = (\theta_0, \ldots, \theta_{k-1}, \theta_{graph}, \theta') - \gamma \nabla L_y(h_{k,\theta_0,\ldots,\theta_{k-1},\theta_{graph},\theta'})$

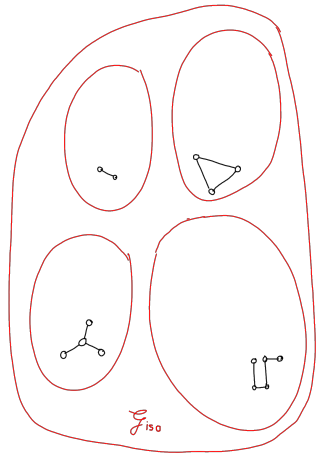
(Message Passing) Graph Neural Networks | The Expressive Power of Graph Neural Networks

Expressivity (1)

- Isomorphism defines an equivalence relation on any graph class $\mathcal{G}$
- We can consider the quotient space $\mathcal{G}_{iso}$ that contains exactly one representative graph G_X for each subset $X \subseteq \mathcal{G}$ of pairwise isomorphic graphs in $\mathcal{G}$
- Our main assumption means that our target function $f : \mathcal{G} \to \mathcal{Y}$ that is invariant under isomorphism induces *a function*

$$f_{iso} : \mathcal{G}_{iso} \to \mathcal{Y}$$

$$f_{iso}(G_X) = f(G_X)$$



Expressivity (2)

- Complete functions on $\mathcal{G}$ induce *injective* functions on $\mathcal{G}_{iso}$
- Importantly, injective functions on $\mathcal{G}_{iso}$ are the most *expressive* functions in the following sense:

Definition

Let $f, f' : \mathcal{G} \to \mathcal{Y}$ be two functions that are invariant under isomorphism. f' is *less expressive* than f, denoted $f' \leq_x f$ if

$$f(G) = f(H) \;\Rightarrow\; f'(G) = f'(H)$$

- f, f' are *equally expressive* if $f \leq_x f'$ and $f' \leq_x f$
- f' is *strictly less expressive* than f, written $f' <_x f$ if $f' \leq_x f$ and not $f \leq_x f'$

GNN and WL Representations are Equally Expressive (1)

Theorem

Let $\mathcal{G}$ be the class of all graphs and let $R_k : \mathcal{G} \to \mathbb{R}^d$ be a representation of $\mathcal{G}$ computed by a message passing graph neural network r_k with k iterations and some final graph aggregation step $\mathsf{agg}'(\{\{r_k(v) \mid v \in V(G)\}\})$. *Then*

$$R_k \leq_x R_k^{WL}$$

GNN and WL Representations are Equally Expressive (2)

Theorem

Let $\mathcal{G}$ be the class of all graphs and let $R_k : \mathcal{G} \to \mathbb{R}^d$ be a representation of $\mathcal{G}$ computed by a message passing graph neural network r_k with k iterations and some final graph aggregation step $\mathsf{agg}'(\{\{r_k(v) \mid v \in V(G)\}\})$. *Then*

$$R_k^{WL} \leq_x R_k$$

if the following conditions hold:

1. *r_k aggregates and updates node features with injective functions* agg *and* upd *in*
$$r_{k+1}(v) = \mathsf{upd}_k\left(r_k(v),\ \mathsf{agg}_k\left(\{\{r_k(w) \mid w \in N(v)\}\}\right)\right)$$
2. *R_k's graph-level aggregation step, which operates on $\{\{r_k(v) \mid v \in V(G)\}\}$, is injective.*

The GIN Theorem

Lemma

Assume $\mathcal{X}$ is countable. Then there exists a function $f : \mathcal{X} \to \mathbb{R}^d$ such that for infinitely many choices of ϵ,

$$h(c, X) = (1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x)$$

is unique for each pair (c, X), where $c \in \mathcal{X}$ and $X \subseteq \mathcal{X}$ is a multiset of bounded size.
Moreover, any function g over such pairs can be decomposed as

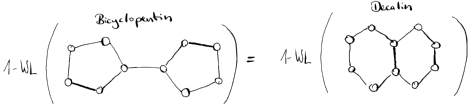
$$g(c, X) = \phi\left((1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x)\right)$$

for some function ϕ.

Extensions to Message Passing

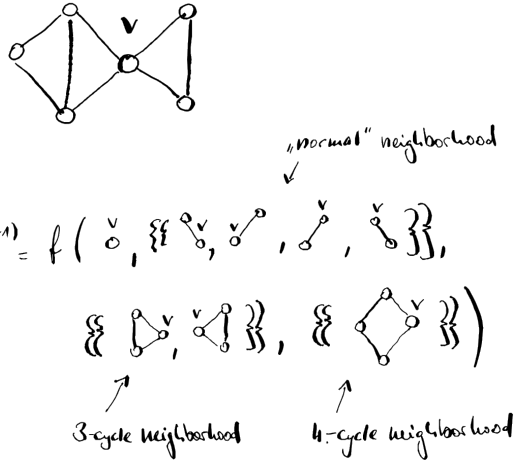
At a glance

- Property prediction for small molecules is one main application area of GNNs
- Number and type of cycles in molecules is important
- But



- Extension of the Message Passing Paradigm over generalized neighborhoods

A glimpse at one such Architecture



- Generalized message passing over multiple sets of local "neighborhoods"
- Cycles can be enumerated quickly on many sparse graphs
  Horváth et al (2004)
- Cycle representations can be computed with GINs

A complete representation for cycles:

Lancaster University

Chapter 1 | Appendix

References

Jin-yi Cai, Martin Fürer, Neil Immerman (1992) An optimal lower bound on the number of variables for graph identifications. Comb 12(4):389–410, DOI 10.1007/BF01305232

Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, Dominique Beaini (2022) Long range graph benchmark. In: Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track, URL https://openreview.net/forum?id=in7XC5RcjEn

Kurt Hornik, Maxwell B Stinchcombe, Halbert White (1989) Multilayer feedforward networks are universal approximators. Neural Networks 2(5):359–366, DOI 10.1016/0893-6080(89)90020-8

Tamás Horváth, Thomas Gärtner, Stefan Wrobel (2004) Cyclic pattern kernels for predictive graph mining. In: Won Kim, Ron Kohavi, Johannes Gehrke, William DuMouchel (eds) Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004, ACM, pp 158–167, DOI 10.1145/1014052.1014072

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, Jure Leskovec (2020) Open graph benchmark: Datasets for machine learning on graphs. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, URL https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html

Thomas N Kipf, Max Welling (2016) Semi-supervised classification with graph convolutional networks. CoRR abs/1609.02907, URL http://arxiv.org/abs/1609.02907, 1609.02907

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, Marion Neumann (2020) Tudataset: A collection of benchmark datasets for learning with graphs. CoRR abs/2007.08663, URL https://arxiv.org/abs/2007.08663, 2007.08663

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, Gabriele Monfardini (2009) The graph neural network model. IEEE Trans Neural Networks 20(1):61–80, DOI 10.1109/TNN.2008.2005605

Boris Weisfeiler, Andrei Leman (1968) A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsia 2(9)

Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka (2019) How powerful are graph neural networks? In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, URL https://openreview.net/forum?id=ryGs6iA5Km