

The Power of Graph Learning

Floris Geerts (University of Antwerp, Belgium)

Vertex level

Suboraph **<u>become</u>** Subgraph level

Edge-level Edge/link level

Examples

- Vertex classification: categorise online user/items, 2002 location amino acids (protein folding, alpha fold)
- Link prediction: knowledge graph completion, recommender systems, drug side effect discovery
- Graph classification: molecule property, drug discovery
- Subgraph tasks: traffic prediction

Images: Machine Learning on Graphs, Stanford course Jure Leskovec

Jure Leskovec, Stanford CS224W: Machine Learning with Graphs 4

Particle Networks Image credit: <u>Wikipedia</u> Image credit: Pinterest Image credit: Polymen Image credit: <u>visitlondon.com</u>

Food Webs Underground Networks

Event Graphs Computer Networks

Disease Pathways

2/16/2023

Why learning on graphs?

Economic Networks Communication Networks

Image credit: Missoula Current News

Citation Networks

Networks of Neurons

Image credit: The Conversation

Internet

Knowledge Graphs

Molecules

Image credit: MDPI

Regulatory Networks

Graphs are everywhere!

Images: Machine Learning on Graphs, Stanford course Jure Leskovec

Image credit: <u>Science</u> Image credit: Lumen Learning

Graph learning methods are thus widely applicable

Jure Leskovec, Stanford CS224W: Machine Learning with Graphs 4

Particle Networks Image credit: <u>Wikipedia</u> Image credit: Pinterest Image credit: Pinterest Image credit: visitlondon.com

Food Webs Underground Networks

Event Graphs Computer Networks

Disease Pathways

2/16/2023

Networks of Neurons Image credit: The Conversation

Why learning on graphs?

Economic Networks Communication Networks

Image credit: Missoula Current News

Citation Networks

Internet

Knowledge Graphs Image credit: Maximilian Nickel et al

Molecules

Image credit: MDPI

Regulatory Networks

Graphs are everywhere!

Images: Machine Learning on Graphs, Stanford course Jure Leskovec

Graph learning methods are thus widely applicable

How is learning typical done?

Embedding-based graph learning

Embedding-based graph learning \mathcal{P} AB Embedding method \mathbb{R}^d … $\mathscr{G} = \text{all graphs}$ $\mathbb{Y} = \text{output space}$

Embedding-based graph learning The world of classical ML \mathcal{P} AB Embedding method $\sqrt{2}$ ℝ*d* … $\mathscr{G} = \text{all graphs}$ $\mathscr{V} = \text{output space}$

Embeddings

Graph embedding: *ξ* : → $Vertex\ embedding: ξ : Θ → (V → V)$ p -Vertex embedding: $\xi : \mathcal{G} \to (\mathcal{V}^p \to \mathbb{Y})$

 $\mathcal{G} = \text{all graphs}$ $\mathcal{V} =$ all vertices $\mathbb{Y} =$ output space

Graph embeddings

Graph embedding: *ξ* : →

Graph classification/regression

Vertex embeddings

$Vertex\ embedding: ξ : Θ → (V → V)$

Vertex classification/regression. For example, prediction of subject of papers.

 $paper_1 \rightarrow \text{math}$ $paper, \rightarrow computer science$ $\ddot{\hspace{1.5cm}}$ $\ddot{\hspace{1.5$ paper_n biology *ξ*(*G*, *v*)

p -Vertex embedding: ξ: *'θ* → (γ^ρ → M)

p-Vertex embeddings

(Adam, Eve) \rightarrow (Trump, Biden)

For example, 2-vertex embeddings: link prediction

…

ξ(*G*, *v*, *w*) ↦ link ↦ no link

What makes graphs special?

…

Support vector machines

…

What makes graphs special?

 $V = \mathbb{R}^{10}$

…

Invariant embeddings

We need embeddings to be graph invariants

Isomorphic inputs should give the same result

Invariant embeddings

We need embeddings to be graph invariants Isomorphic inputs should give the same result

for all *G*, all isomorphisms π , and $\mathbf{v} \in V^p$

Invariance is typically achieved by composing invariant building blocks to build embeddings

-
-

 $G^p : \xi(G, \mathbf{v}) = \xi(\pi(G), \pi(\mathbf{v}))$ p -vertex embedding $\xi : \mathcal{G} \to (\mathcal{V}^p \to \mathbb{Y}) : (G, v) \mapsto \xi(G, v)$ is invariant if

Graph learning: Invariant embeddings

$\mathscr{G} = \text{all graphs}$ $\mathscr{V} = \text{output space}$

ℝ*d*

…

The world of classical ML

Graph learning: Invariant embeddings

The world of Invariant!

$\mathscr{G} = \text{all graphs}$ $\mathbb{Y} = \text{output space}$

Embedding method

ℝ*d*

…

Graph learning: Invariant embeddings

The world of Invariant!

Embedding method

$\mathscr{G} = \text{all graphs}$ $\mathscr{V} = \text{output space}$

ℝ*d*

…

Hypothesis class ℋ

Graph learning: ERM

Given training set $\mathcal T$ and hypothesis class $\mathcal H$ of invariant embedding methods $\mathcal{T} := \{(G_1, v_1, y_1), ..., (G_\ell, v_\ell, y_\ell)\} \subseteq \mathcal{G} \times \mathcal{V}^p \times$

Empirical risk minimisation: Find embedding ξ in $\mathcal X$ which minimises empirical loss on training set \mathcal{T} :

 $\int_{i=1}^{i}$ $|\text{oss}(\xi(G_i, \mathbf{v}_i), y_i)|$

Image: Christopher Morris

Image: Christopher Morris

Era of Deep Graph learning

k-GNNs

ℋ GraphSage GCN GIN Dropout GNN Graphormer GATs Id-aware GNN CWN GNN as Kernel MPNNs MPNN+ SGNs **GatedGCNs** Walk GNNs Reconstruction GNNs Ordered subgraph Networks

k-IGNs k-FGNNs

"Deep" hypothesis classes

2-IGN GIN PPGN ChebNet 2-IGN GIN CayleyNet *δ* − *k*−GNNs randomMPNN Simplicial MPNNs k-GNNs k-LGNNs Nested GNNs units (GRU) of Chung et al. (2014):

 $h_i^{\ell+1} = f_{\text{G-CNN}}\left(h_i^{\ell} \, , \, \{h_j^{\ell} : j \to i\} \,\right)$

where \mathcal{M} is the Hadamard point-wise multiplication operator. This model was used for N by Li et al. (2016) and also in quantum chemistry by Gilmer et al. (2017) for fast organic molecule

properties estimation, for which standard techniques (DFT) require expensive computational time. Tree-Structure Communication Communication Communication of the authors extending the original LSTM model of the o Hochreiter & Schmidhuber (1997) to a tree-graph structure: iteration, we do not need MLPs before summation if input features are one-hot encodings as their is injective. We can make ϵ a learnable parameter or a fixed sc resentations as

and *^C*G-GRU(*h^t*

*j*2*C*(*i*)

 G is the contribution Networks of Li et al. (2016). In this work, the authors use the gated recurrent records use the gated recurrent \mathcal{L}_max p_{recon} securirence of subtrees is sparse across different graphs or the are noisy p_{recon} section 4. Consider the matrix $\mathbf{X} \in \mathbb{R}^{n \times 2a}$ defined by

believe this mechanism to be important for graphs, as they will be able to learn what edges are

 μ tion of Eq. (4) does not have an analytical solution, Li et al. re scheme: $T_{i_1,i_2,i}$ multiplier multiplier multipliers for the neighbor aggregation, we develop a theory of "deep" of "deep" of "deep" develop a theory of "deep" develop a theory of "deep" develop a theory of "deep" develop a the tion of Eq. (4) does not have an analytical solution, Li et al. states that sum aggregators can represent injective, in fact, *universal* functions over multisets.

 $h_i^{t+1} = C_{\text{G-GRU}}(h_i^t, \bar{h}_i^t), \quad h_i^{t=0} = x_i \ \ \forall i,$ where $\bar{h}_i^t = \sum$ $j \rightarrow i$ h_j^t Lemma 5. *Assume ^X is countable. There exists a function ^f* : *^X* ! ^R*ⁿ so that ^h*(*X*) = ^P $h_i^{t+1} = \mathcal{C}_{\text{G-GRU}}(h_i^t, \bar{h}_i^t), \quad h_i^{t=0} = x_i \ \forall i,$ *decomposed as g* (*X*) = v_i = $C_{\text{G-RU}}(n_i, n_i), n_i$ *^x*2*^X ^f*(*x*) \mathcal{L}^{n_j} multisets. An important distinction between deep multisets and sets is that certain popular injective se

$$
h_i = f_{\text{G-GRU}}(x_i, \{h_j : j \to i\}) = C_{\text{G-GRU}}(x_i, \sum_{j \to i} h_j)
$$
the form
Now def
[b]). We

a strong assumption. Besides, a large number of iterations can be computations of its can be computations can be computational expensive. It i

ⁱ = *f*`

$$
z_i^{t+1} = \sigma(U_z h_i^t + V_z \bar{h}_i^t) \n r_i^{t+1} = \sigma(U_r h_i^t + V_r \bar{h}_i^t) \n \tilde{h}_i^{t+1} = \tanh(U_h (h_i^t \odot r_i^{t+1}) + V_h \bar{h}_i^t) \n h_i^{t+1} = (1 - z_i^{t+1}) \odot h_i^t + z_i^{t+1} \odot \tilde{h}_i^{t+1},
$$

In other words, the vector *h*`+1

Molecular Graph Convolutions, Kearnes et al. (2016). A traditions the convolution of the second convolutions,
Convolutions, Convolutions, Kearnes et al. (2016).

Consider a p-dimensional simplicial complex with Sⁿ n- $X_{j,:} = (\mathbf{B}_{j,i_2,:}, \mathbf{B}_{i_1,j,:}), \quad j \in [n].$ (7)

3 MODELS STATES

The product over *n* in (10) reflects the fact that the features over simplices of different dimensions do not interact. The $G_{\rm N}$ is recovered as the special case of σ is recovered as the special case of σ

Here, it is also worth discussing an important benefit of GNNs beyond distinguishing different graphs,

edges and node features (Yanardag & Vishwanathan, 2015).

4.1 GRAPH ISOMORPHISM NETWORK (GIN)

r₁U, a
itrica $M_n \in \mathbb{R}^{S_n \times S_n}, U_n \in \mathbb{R}^{S_n \times S_{n-1}}$, and $O_n \in \mathbb{R}^{S_n \times S_{n+1}}$
and write it in complex. These could be the Hodge Laplacian matrix L_n
and the corresponding boundary matrices B_n^{\top}, B_{n+1} , or one complex. These could be the Hodge Laplacian matrix L_n
and the corresponding boundary matrices B^T , B and or one where ψ is an entry-wise activation ($s \mapsto \max\{0, s\}$ for ReLU), $W_n \in \mathbb{R}^{d_n \times m_n}$ are trainable weight matrices and are some choice of adjacency matrices for the simplicial of their variants (e.g. normalised).

It is convenient to write the entire layer output in standard

Recurse and approximating over a we form. Using Roth's lemma and concatenating over *n* we (β_i, γ_i) can write (11) as (details in Appendix B)

Hypothesis classes: how do they look like? The iterative scheme is guaranteed to converge as long as the mapping is contractive, which can be communication between multiple agents to solve multiple tasks like traffic control. Syntactic Graph Convolutional Networks of Marchestra Convolution (2017). The authorized convolution of Marches posed the following transfer function that is, capturing similarity of graph structures. Note that node feature vectors in the WL test are \sim essentially one-hot encodings and thus cannot capture the similarity between subtrees. In contrast, a GNN satisfying the criteria in Theorem 3 generalizes the WL test by *learning to embed* the subtrees to low-dimensional space. This enables GNN space. learn to map similar graph structures to similar embeddings and capture dependencies between graph structures. Capturing structures is shown to be helpful for the node labels is shown to be helpful for generalization *n* $\frac{1}{2}$ $\frac{1}{2}$ IN THE THE FEATURE feature dimension by *mⁿ* = *m*, *n* = 0*,...,p*. We consider an MPSN layer with linear message functions, such as a series of the sum agency of the sum and sum and sum agency \sqrt{g} sum of the messages followed by a ReLU activation. For each dimension *n*, the output feature matrix *H*out *ⁿ* equals: the multiset *{{*(**B***j,i*2*,*:*,* **B***ⁱ*1*,j,*:) *| j* 2 [*n*]*}}*. As before, we use the multiset representation introduced in

*^x*2*^X ^f*(*x*)

functions, such as the mean aggregator, are not injective multiset functions. With the mechanism for

will satisfy the injectiveness condition (a) in Theorem 3. Our next condition (a) in Theorem 3. Our next corollary provides a simple and α

i^{$\binom{1}{2}$ is equal to} modeling universal multiset functions in Lemma 5 as a building block, we can conceive aggregations in Lemma 5 a schemes that can represent universal functions over a node and the multiset of its neighbors, and thus, and thus

maximally powerful GNNs, which being simple. While being simple, which being simple.

4.2 GRAPH-LEVEL READOUT OF GINAL AND CONTRACTOR

some function '*.*

Implementation details. We implemented the GNN model as described in Section 6 (see Equation 6) using the Tensor Flow flow framework (Abadi et al., 2016). We used the Tensor Flow for the Used three in

We can use multi-layer perceptrons (MLPs) to model and learn *f* and ' in Corollary 6, thanks to the universal approximation theorem (Hornik et al., 1989; Hornik, 1991). In practice, we model *ⁱ* = *f*` ϕ $\left(\frac{1}{n} + \frac{1}{n} + \frac{1}{n} + \frac{1}{n} + \frac{1}{n+1} + \frac{1}{n+1} + \frac{1}{n+1} + \frac{1}{n+1} \right)$, $\left(\frac{1}{n} \right)$ $n_1 n_1 n_2 n_1 n_1 n_1 n_1 n_1 n_1$ $\frac{1}{\sqrt{2}}$ $\psi\left(M_nH_n^{\text{in}}W_n+U_nH_{n-1}^{\text{in}}W_{n-1}+O_nH_{n+1}^{\text{in}}W_{n+1}\right),$ $\label{eq:4.1} \Big(M_n H_n^{\rm in} W_n + U_n H_{n-1}^{\rm in} W_{n-1} + O_n H_{n+1}^{\rm in} W_{n+1}$ \overline{a} *,* (11)

(MPS), the features on simplicity of different different dimensions of different dimensions of different differe

^f(*k*+1) '(*k*) with one MLP, because MLPs can represent the composition of functions. In the first

$$
h_v^{(k)} = \text{MLP}^{(k)}\left(\left(1 + \epsilon^{(k)}\right) \cdot h_v^{(k-1)} + \sum\nolimits_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right).
$$

may exist many other powerful GNNs. GIN is one such examp

Note the second substantial by GIN can be directly used for the direction of the second for the second for the α

posed the following transfer function: (without having to account for all subsets of size 3).

$$
h_i^{\ell+1} = f_{\text{S-GCN}}^{\ell} \left(\{ h_j^{\ell} : j \to i \} \right) = \text{ReLU} \left(\sum_{j \to i} \eta_{ij} \odot V^{\ell} h_j^{\ell} \right)
$$

sedge gates, and are computed by:

 $\eta_{ij} = \sigma \left(A^{\ell} h^{\ell}_i + B^{\ell} h^{\ell}_j \right)$ $n_{ij} = \sigma (A^{\ell} h^{\ell}_{i} + B^{\ell} h^{\ell}_{i}).$ time complexity can be improved to *O*(*n*²*.x*) due to more advanced matrix multiplication algorithms. tion *R* used. Note one could also learn the could als

Theorem 16 (Number of linear regions for an SCNN layer).

$$
\hat{h}_i^{\ell,t} = \sum_{j \to i} h_j^{\ell,t},
$$
\n
$$
\hat{p} = R(\lbrace h_v^T \mid v \in G \rbrace).
$$
\n
$$
\hat{p} = R(\lbrace h_v^T \mid v \in G \rbrace).
$$
\n
$$
\hat{p} = \sum_{i \to i} h_i^{\ell,t} = \sum_{j \to i} h_j^{\ell,t},
$$
\n
$$
\hat{p}^{\ell,t+1} = \sigma(U_i^{\ell} x_i^{\ell} + V_i^{\ell} \bar{h}_i^{\ell,t})
$$
\n
$$
\hat{p}^{\ell,t+1} = \sigma(U_i^{\ell} x_i^{\ell} + V_i^{\ell} \bar{h}_i^{\ell,t})
$$
\n
$$
\hat{p}^{\ell,t+1} = \tanh(U_i^{\ell} x_i^{\ell} + V_i^{\ell} \bar{h}_i^{\ell,t})
$$
\n
$$
\hat{p}^{\ell,t+1} = \tanh(U_i^{\ell} x_i^{\ell} + V_i^{\ell} \bar{h}_i^{\ell,t})
$$
\n
$$
\hat{p}^{\ell,t+1} = \sigma(U_j^{\ell} x_i^{\ell} + V_j^{\ell} h_j^{\ell,t})
$$
\n
$$
\hat{p}^{\ell,t+1} = \sigma(U_j^{\ell} x_i^{\ell} + V_j^{\ell} h_j^{\ell,t})
$$
\n
$$
\hat{p}^{\ell,t+1} = \hat{p}^{\ell,t+1} \odot \hat{p}^{\ell,t+1} + \sum_{j \to i} f_{ij}^{\ell,t+1} \odot e_j^{\ell,t+1}
$$
\n
$$
\hat{p}^{\ell,t+1} = \sigma_i^{\ell,t+1} \odot \tanh(e_i^{\ell,t+1})
$$
\n
$$
\hat{p}^{\ell,t+1} = \sigma_i^{\ell,t+1} \odot \tanh(e_i^{\ell,t+1})
$$
\nand initial conditions:
$$
h_i^{\ell,t=0} = \hat{p} = \hat{p} = 0, \forall i, \ell
$$

*^mnSn*¹ *i*

$$
W = \begin{bmatrix} W_0^\top \otimes M_0 & W_1^\top \otimes O_0 \\ W_0^\top \otimes U_1 & W_1^\top \otimes M_1 & W_2^\top \otimes O_1 \\ W_1^\top \otimes U_2 & W_2^\top \otimes M_2 & W_3^\top \otimes O_2 \\ & & & \ddots \end{bmatrix} . \tag{13}
$$

We study the number of linear regions of linear regions of the function (12) and the function (12) and the function (12) and

with recurrent formula is the general case of graphs, we proceed as Scarsen et al. and iterative process to solve Eq. (10): At layer ℓ , for $t = 0, 1, ..., T$ the SCNN layer as a standard layer (*Ux*) with weight *ⁿ* ⌦ *^Mn*) ² ^R(*mnSn*)⇥(*Sndn*) and input *n* 2 *n* 2 *n* 3 *n* input space R*^N* into two regions separated by a hyperplane $\frac{1}{2}$ of the consider to consider the constant constant of $\frac{1}{2}$ of $\frac{1}{2}$ or $\$ get at each node in the graph, and where is a graph, $\frac{1}{2}$

$$
H^{\text{out}} = \psi(WH^{\text{in}}),\tag{12}
$$

 $\sum_{n=0}^{\infty}$ $\sum_{n=0}^{\in$ graphs and multiple layers: where $H^{\text{in}} = \text{vec}([H_0^{\text{in}}|H_1^{\text{in}}|\cdots|H_p^{\text{in}}]$ $\begin{array}{l} \hbox{where}\;\; H^{\rm in} \;=\; {\rm vec}([H_0^{\rm in}|H_1^{\rm in}|\cdots |H_p^{\rm in}])\;\in\; \mathbb{R}^N,\;N\;=\ \sum_{n=0}^p S_nd_n,H^{\rm out} ={\rm vec}([H_0^{\rm out}|H_1^{\rm out}|\cdots |H_p^{\rm out}])\in\mathbb{R}^M, \end{array}$ $\overline{M} = \sum_{n=0}^{p} S_n m$, and where $H^{\text{in}} = \text{vec}([H_0^{\text{in}}|H_1^{\text{in}}|\cdots|H_p^{\text{in}}]) \in \mathbb{R}^N$, I
 $\sum_{n=0}^p S_n d_n$, $H^{\text{out}} = \text{vec}([H_0^{\text{out}}|H_1^{\text{out}}|\cdots|H_n^{\text{out}}]) \in$

$$
m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})
$$

$$
h_v^{t+1} = U_t(h_v^t, m_v^{t+1})
$$

aph using some readout function R according to the refer to the h α β β β β \overline{u} *x* \overline{u} \overline{v} \overline{u} layer, the weight matrix has a specific structure. A standard matrix has a specific structure. A structure in dard dense layer with *Sndⁿ* inputs and *mnSⁿ* ReLUs with $\frac{1}{2}$ according to biases computes functions with $\frac{1}{2}$ $N(x)$ denotes the noighbors of v , i . *j,i*2*,...,i^k* **k**₁, *i*₁ *i*_{*k*} *v*₁ denotes R and using some readout function R according aph using some readout function *R* according to the expression of the expression of all the expression of the exp the sum, $N(v)$ denotes the neighbors of v in **Gr. The readout phase computes a feature vector for the readout phase computes a feature vector**

hence **W***ⁱ*1*,i*2*,*: = *u*(*X*)*,* where *X* is defined in Equation 7. To get an implementation with the model in Equation 6 we need to replace ⌧1*,* ⌧² with MLPs. We use the universal approximation theorem to

that end (details are in the supplementary material).

$$
\mathbf{W}_{i_1,i_2,l} := (\mathbf{Z}_{:,,:,l} \cdot \mathbf{Y}_{:,:,l})_{i_1,i_2} = \sum_{j=1}^n \mathbf{Z}_{i_1,j,l} \mathbf{Y}_{j,i_2,l} = \sum_{j=1}^n \mathbf{B}_{j,i_2,:}^{\beta_l} \mathbf{B}_{i_1,j,:}^{\gamma_l} = \sum_{j=1}^n (\mathbf{B}_{j,i_2,:}, \mathbf{B}_{i_1,j,:})^{(\beta_l, \gamma_l)},
$$
can v

)*ⁱ* =

SCRIPT SIMPLICIAL COMPLEX COMPLEX PROTECTIVE COMPLEX COMPLEX NETWORKS (SCNNESS COMPLEX SCHNESS COMPLEX COMPLEX
SIMPLICIAL COMPLEX SIMPLICIAL COMPLEX COMPLEX COMPLEX COMPLEX SUR LA COMPLEX COMPLEX SURVEY COMPLEX COMPLEX CO

^j=1 **^A**¹

7 Experiments

Consider the multi-index set $\{\alpha \mid \alpha \in [n]^{2a}, |\alpha| \leq n\}$ of cardinality $b = \binom{n+2a-1}{2a-1}$), and write it in the form $\{(\beta_l, \gamma_l) | \beta, \gamma \in [n]^a, |\beta_l| + |\gamma_l| \leq n, l \in b\}.$

 $T \mid l \in [0]$, and $\tau_2(x) = (x^n \mid l \in [0])$, and $\tau_2(x) = (x^n \mid l \in [0])$ of their variants (e.g. normalised).
 (B)_{$i_1, i_2, l = (\mathbf{B}_{i_1, i_2, l})^{\beta_i}$; similarly,} Now define polynomial maps $\tau_1, \tau_2 : \mathbb{R}^a \to \mathbb{R}^b$ by $\tau_1(x) = (x^{\beta_l} | l \in [b])$, and $\tau_2(x) = (x^{\gamma_l} | l \in [b])$ *n*=0 2 *i*_l). We apply τ_1 to the features of **B**, namely $Y_{i_1,i_2,l} := \tau_1(\mathbf{B})_{i_1,i_2,l} = (\mathbf{B}_{i_1,i_2,l})^{\beta_l}$; similarly, $\mathbf{Z}_{i_1,i_2} := \tau_2(\mathbf{B})_{i_1,i_2}$, $\mathbf{B}_{i_2,i_2} := \tau_1(\mathbf{B})_{i_2,i_2}$ $\mathbf{Z}_{i_1,i_2,l} := \tau_2(\mathbf{B})_{i_1,i_2,l} = (\mathbf{B}_{i_1,i_2,:})^{\gamma_l}$. Now,

*^B*1*, B*2*, B*3, where in each block *^Bⁱ* : ^R*ⁿ*2⇥*^a* ! ^R*ⁿ*2⇥*^b* we took *^m*3(*x*) = *^x* to be the identity (i.e.,

updated based on messages *m^t*+1

^v according to

The message functions *Mt*, vertex update functions *Ut*, and

an MPNN by introducing hidden states for all edges in the

and 2. Of the existing MPN is the existing MPN of the existing MPN and the existing MPN α and α also the existing α

$$
\hat{y} = R(\{b
$$

R = X

states *h^T*

for *T* = 1.

Our goal is to compute an output tensor $W \in \mathbb{R}^{n^2 \times b}$, where $W_{i_1, i_2, :} = u(X)$. *represented by a Relation by a Relatio*

as the vanilla graph $C_{\rm{C}}$ such that graph neural network to learn the value of α $y = R(\lbrace h_v^{\dagger} \mid v \in G \rbrace).$ message passing networks. First, it has lower space complexity (see details below). This allows us to the lower space complexity (see details below). This allows us to the lower space complexity (see details below). This $\hat{y} = R({h_v^T \mid v \in G})$. version, hindering the internation, from a problem a problem a problem of view our model is view our model is

arguably simpler to implement as it only consists of fully connected in the matrix multiplication of fully consists of fully connected layers and matrix multiplication of fully connected layers and matrix multiplication of

How to compare different classes? How to compare such embedding classes theoretically? How to bring order to the chaos?

How to compare different classes?

 \cdot How to compare such embedding classes theoretically?

How to bring order to the chaos?

1. See graph embedding methods as queries in some query language

> 2. Analyse expressive power of query language

3. Transfer understanding back to graph learning world

How to compare different classes?

 \cdot How to compare such embedding classes theoretically?

How to bring order to the chaos?

2. Analyse expressive power of query language

1. See graph embedding methods as queries in some query language

3. Transfer understanding back to graph learning world

What kind of language?

> Expressive power?

Graph Embedding Language

Graph Embedding Language (GEL)

Most methods are specified in terms of linear algebra computations interleaved with non-linear function applications

Crucial component is multiplication with adjacency matrix which corresponds to neighbourhood aggregation

Desired language needs function application and aggregation

Graph Neural Networks 101 Non-linear activation function (ReLU, sign, sigmoid, …) *σ* $\mathbf{F}_G^{(t)} \in \mathbb{R}^{n \times d}$ denotes embedding of vertices in graph G 0.1 31 8 4.03 5 0.03 9.7 −1 −3 118 −63 0.204 *v*₁ v_2^1 v_3^2

Weight matrices $W_1^{(t)} \in \mathbb{R}^{d \times d}$ and $W_2^{(t)} \in \mathbb{R}^{d \times d}$ and bias vector $\mathbf{b} \in \mathbb{R}^{1 \times d}$

 $\mathbf{A}_{0} \mathbf{F}_{G}^{(t-1)} \mathbf{W}_{2}^{(t)} + \mathbf{B}^{(t)}$ Aggregation over neighbours Adjacency matrix ∈ ℝ*n*×*^d*

Ine embedding of vertex labels

$$
\mathbf{F}_G^{(0)} \leftarrow \text{Initial hot-c}
$$
\n
$$
\mathbf{F}_G^{(t)} := \sigma \left(\mathbf{F}_G^{(t-1)} \mathbf{W}_1^{(t)} + \cdots \right)
$$

Matrix form

GNN 101: Graph embedding • Weight matrix $W \in \mathbb{R}^{d \times d}$ and and bias vector $\mathbf{b} \in \mathbb{R}^{1 \times d}$

and biases $\Theta = W_1^{(1)}, ..., W_1^{(L)}, W_2^{(1)}, ..., W_2^{((L)}, W, \mathbf{b}^{(1)}, ..., \mathbf{b}^{(L)}, \mathbf{b})$

 $\mathbf{F}_G := \sigma \mid \sum$

Aggregation over all vertices $\mathbf{F}_G^{(L)}\mathbf{W} + \mathbf{b} \in \mathbb{R}^{1 \times d}$

Hypothesis class \mathcal{H} consists of $\xi_{\Theta}: G \mapsto \mathbf{F}_G$ parametrised by weights

v∈*VG*

Empirical Risk Minimisation: Find best parameters Θ

Graph Embedding Language (GEL)

 $\xi_{\varphi}: \mathcal{G} \to (\mathcal{V}^{\ell} \to \mathbb{R}^d)$)

GEL expression

Higher order embedding

Hella, Libkin, Nurmonen, Wong: *Logics with Aggregates.* (2001) Abo Khamis, Ngo, Pichler, Suciu, Wang: *Convergence of Datalog over (Pre-) Semiring..* (2022) G. and Reutter: *Expressiveness and approximation properties of graph neural networks.* (2022)

 $\varphi(\mathbf{x})$ of dimension *d* and free variables $\mathbf{x} = \{x_1, ..., x_e\}$

Syntax

Semantics

A simplified version of a query languages with aggregates studied in database theory and it resembles Datalog[∘]

 $\xi_{\varphi}(G, x_i/v, x_j/w) := \{$ 1 $(v, w) \in E_G$ 0 otherwise $\xi_{\varphi}(G, x_i/v) := j$ th feature of *v* $\xi_{\varphi}(G, x_i/v, x_j/w) := \{$ $1 \quad v = w$ 0 otherwise

Atomic GEL expressions

Atomic expressions Semantics

Label: $\varphi(x_i) := \text{Lab}_j(x_i)$ of dim 1 and free var x_i Edge: $\varphi(x_i, x_j) := E(x_i, x_j)$ of dim1, free vars x_i, x_j Equality: $\varphi(x_i, x_j) := 1[x_i = x_j]$ of dim 1, free vars x_i, x_j

Assigning vertex *v* to *xi*

Function application: Syntax

Let $\varphi_1(\mathbf{x_1}),...,\varphi_\ell(\mathbf{x_1})$ be GEL expressions of $\dim d_1,...,d_\ell$ and free vars $\mathbf{x_1},...,\mathbf{x_\ell}$ $\mathop{\rm Let}\nolimits F: \mathbb{R}^{d_1+\cdots+d_\ell} \to \mathbb{R}^d$ be a function. Then,

is a GEL expression of dim *d* and free vars $\mathbf{x} = \mathbf{x}_1 \cup \dots \cup \mathbf{x}_e$

-
- $\varphi(\mathbf{x}) = F(\varphi_1, ..., \varphi_{\ell})$
	-

GEL: Function Application
∈ ℝ*^d* \mathbb{R}^{d_1} \mathbb{R}^{d_ℓ}

Function application: Syntax

Let $\varphi_1(\mathbf{x_1}),...,\varphi_\ell(\mathbf{x_1})$ be GEL expressions of $\dim d_1,...,d_\ell$ and free vars $\mathbf{x_1},...,\mathbf{x_\ell}$ $\mathop{\rm Let}\nolimits F: \mathbb{R}^{d_1+\cdots+d_\ell} \to \mathbb{R}^d$ be a function. Then,

is a GEL expression of dim *d* and free vars $\mathbf{x} = \mathbf{x}_1 \boldsymbol{\mu} \cdots \boldsymbol{\nu} \mathbf{x}_{\ell}$

GEL: Function Application

Semantics

 $\mathcal{E}_{\varphi}(G, \mathbf{x}/\mathbf{v}) := F\left(\mathcal{E}_{\varphi_1}(G, \mathbf{x}_1/\mathbf{v}_1), \ldots, \mathcal{E}_{\varphi_\ell}(G, \mathbf{x}_\ell/\mathbf{v}_\ell)\right)$ ∩ ∩

 $\varphi(\mathbf{x}) = F(\varphi_1, ..., \varphi_{\ell})$

Linear algebra Activation functions Anything you want…

 $\xi_{\varphi}(G, \mathbf{x}/\mathbf{v}) := \Theta\left(\left\{\left\{\xi_{\varphi_1}(G, \mathbf{x}/\mathbf{v}, \mathbf{y}/\mathbf{w}) \mid \right\}, \mathbf{w} \in V_G^{\mathbf{y}}\right\}\right)$, $\mathbf{w} \in V_G^{\vert \mathbf{y} \vert}$ *G*

GEL: Aggregation

Aggregation: Syntax

be a function mapping bags of vectors in \mathbb{R}^{d_1} to a vector in \mathbb{R}^{d} . Then,

is a GEL expression of dim d and free vars **x**

Semantics

Let $\varphi_1(\mathbf{x}, \mathbf{y})$ and $\varphi_2(\mathbf{x}, \mathbf{y})$ be GEL expressions of $\dim d_1$ and d_2 and free vars \mathbf{x}, \mathbf{y} . Let Θ

$\varphi(\mathbf{x}) = \text{agg}^{\Theta}[\varphi_1 | \varphi_2]$

∈

 \mathbb{R}^{d_1}

GEL: Aggregation

Aggregation: Syntax

Let $\varphi_1(\mathbf{x}, \mathbf{y})$ and $\varphi_2(\mathbf{x}, \mathbf{y})$ be GEL expressions of $\dim d_1$ and d_2 and free vars \mathbf{x}, \mathbf{y} . Let Θ be a function mapping bags of vectors in \mathbb{R}^{d_1} to a vector in \mathbb{R}^{d} . Then,

Semantics

$\xi_{\varphi}(G, \mathbf{x}/\mathbf{v}) := \Theta\left(\left\{\left\{\xi_{\varphi_1}(G, \mathbf{x}/\mathbf{v}, \mathbf{y}/\mathbf{w}) \mid \xi_{\varphi_2}(G, \mathbf{x}/\mathbf{v}, \mathbf{y}/\mathbf{w}) \neq \mathbf{0}, \mathbf{w} \in V_G^{\mathbf{v}}\right\}\right\}\right)$, $\mathbf{w} \in V_G^{\vert \mathbf{y} \vert}$ *G* **guard** $\xi_{\varphi_2}(G, \mathbf{x}/\mathbf{v}, \mathbf{y}/\mathbf{w}) \neq 0$

∈

 \mathbb{R}^{d_1}

 $\varphi(\mathbf{x}) = \text{agg}^{\Theta}[\varphi_1 | \varphi_2]$

is a GEL expression of dim d and free vars **x**

GEL: Aggregation example

⋅ = shorthand for product function application

GEL: Aggregation example

⋅ = shorthand for product function application

What does this compute? 6 x Triangle count

GEL: Aggregation example

$\varphi = \text{aggsum}_{x,y,z} [1[y = y] | E(x, y) \cdot E(y, z) \cdot E(x, z) \cdot 1[x \neq y] \cdot 1[x \neq z] \cdot 1[y \neq z]$

⋅ = shorthand for product function application

What does this compute? 6 x Triangle count

Let us see a more elaborate example

Message Passing Neural Networks

 $\text{We define } \varphi^{(0)}(x_1) := \mathbf{1}[x_1 = x_1]$ Then for $t > 0$, we get

For readout layer, we get

 $\varphi := \text{agg}^\Theta_{x_i}$ *x*1 $\left[\varphi^{(L)}\right]$

Gilmer, Schoenholz, Riley, Vinyals, Dahl.: *Neural message passing for quantum chemistry.* (2017)

 $\varphi^{(t)}(x_1) := \textsf{Upd}^{(t)}\left(\varphi^{(t-1)}(x_1), \textsf{agg}^{\Theta^{(t)}}_{x_2}\left[\varphi^{(t-1)}(x_2) \,|\, E(x_1,x_2)\right]\right)$

 (x_1) $\left[1[x_1 = x_1]\right]$

This encompasses the GNNs 101

Fragments of GEL GEL_k(Ω , Θ): k variable fragment of GEL with functions in Ω and

aggregations in Θ

 $GGEL₂(Ω, Θ): 2 variable fragment GEL with edge guarded$ $\arg \arg \min \{ \varphi(x) = \arg \varphi[\varphi_1 | E(x, y)] \}$

> MPNNs including readout phase fit in $GEL_2(\Omega, \Theta)$ MPNNs without readout phase fit in GGEL(Ω, Θ)

 $GGEL₂(Ω, Θ): 2 variable fragment GEL with edge guarded$ aggregation only

Fragments of GEL GEL_k(Ω , Θ): k variable fragment of GEL with functions in Ω and

aggregations in Θ

ℋ

G., Reutter: *Expressiveness and approximation properties of graph neural networks.* (2022)

 $k+1-IGNs$ randomA k-GNNs k-FGNNs

k-GNNs

GEL₂ GEL₃ GEL*k*

Simplicial MPNNs k-LGNNs

ChebNet 2-IGN GIN Graphormer PPGN CWN Dropout GNN CayleyNet GATs Id-aware GNN GraphSage *δ* − *k*−GNNs MPNNs MPNN+ GCN GIN SGNs GatedGCNs Walk GNNs Reconstruction GNNs Ordered subgraph Networks GNN as Kernel Nested GNNs fit in one of those fragments! CayleyNet Suppletation NET GEL2 GEL2

Graph Convolutional Networks Use $D^{-1/2}(I + A)D^{-1/2}$ as propagation matrix

 $\varphi(x_1) := F(\text{aggsum}[1[x_2 = x_2] | E(x_1, x)])$ with $F : \mathbb{R} \to \mathbb{R} : x \mapsto$ 1 $1 + x$

 $\psi(x_1, x_2) := \varphi(x_1)(1[x_1 = x_2] + E(x_1, x_2))\varphi(x_2)$ as the "adjacency" matrix in in the MPNN expressions we have seen before.

1

 $1 + \deg_G(v)$

GCN: Kipf and Welling: *Semi-supervised classification with graph convolutional networks* (2017)

Hence, $\xi_{\phi}(G, v) =$ <u>and we can use</u>

$GCN\in GGL_2(\Omega,\Theta)$

Simplified GCNs

Uses path information $A^pF^{(0)}$ in a single layer. For $p = 3$ and for $\varphi^{(0)}(x_1)$ initial feature: (x_1)

 $\psi(x_1) :=$ $\mathbb{E}[x]_{x_2} \text{argsum} \left[\mathbb{E}[x]_{x_1} \right] \text{argmin} \left[\mathbb{E}[x]_{x_2} \right]$

Wu et al. :*Simplifying Graph Convolutional Networks* (2019)

 $E(x_1, x_2) |E(x_1, x_2)| |E(x_2, x_1)| |E(x_1, x_2)|$ l

$SGNs \in GGEL_2(\Omega,\Theta)$

k-Folklore GNNs (k-FGNs)

 $\xi^{(t)}(x_1, ..., x_k) := \text{MLP}_1^{(t)}\left(\text{aggsum}_y\right)$ *k* ∏ $i=1$

k-vertex embedding Global aggregation

Maron et al.: *Provably powerful graph networks* (2019) W. Azizian and M. Lelarge. *Characterizing the expressive power of invariant and equivariant graph neural networks* (2021)

 $\binom{t}{2} (\xi^{(t-1)}(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_k)))$

Subgraph count GNNs

- Use count of subgraphs to augment MPNNs
	- homomorphism count hom(P^r , G^v) for rooted motif P,
	- subgraph iso count $\mathsf{sub}(P^r, G^v)$ for rooted motif P
- If motif has tree width k then hom(P^r , G^v) can be computed using k+1 variables.
- For example, $(G, v) \mapsto \text{hom}(\bigwedge, G^v)$ can be expressed as
	- $\varphi(x_1) := \text{aggsum}_{x_2}^{\text{sum}} E(x_1, x_2) E(x_1, x_3) E(x_2, x_3) (1[x_1 = x_1] 1[x_1 = x_2])$ $(1[x_1 = x_1] - 1[x_1 = x_3])(1[x_1 = x_1] - 1[x_2 = x_3])$

$Tree width k \mapsto \text{GEL}_{k+1}(\Omega, \Theta)$

Bouritsas et al.: *Improving graph neural network expressivity via subgraph isomorphism counting* (2020) Barceló et al.: *Graph neural networks with local graph parameters.* (2021)

Bevilacqua et al: *Equivariant subgraph aggregation network* (2022) Cotta et al.: *Reconstruction for powerful graph representations* (2021) Bevilacqua et al.: *Understanding and extending subgraph GNNs by rethinking their symmetries* (2022) Huang et al.: *Boosting the cycle counting power of graph neural networks with I2-GNNs* (2022) Papp et al.: DropGNN: Random dropouts increase the expressiveness of graph neural networks. (2021) Qian et al.: *Ordered subgraph aggregation networks.* (2022) You et al.: *Identity-aware graph neural networks*. (2021) Zhang and P. Li. *Nested graph neural networks* (2021) Zhao et al.: *From stars to subgraphs: Uplifting any GNN with local structure awareness* (2022)

 $\varphi^{(t)}(x_1, x_2, x_3) := \mathsf{Upd}^{(t)}\Big(\varphi^{(t-1)}(x_1, x_2, x_3), \mathsf{agg}^\Theta_{x_4}$ *x*4 $[\varphi^{(t-1)}(x_1, x_2, x_4) | E(x_3, x_4)]$

4 variables \mapsto GEL₄(Ω , Θ)

Bevilacqua et al: *Equivariant subgraph aggregation network* (2022) Cotta et al.: *Reconstruction for powerful graph representations* (2021) Bevilacqua et al.: *Understanding and extending subgraph GNNs by rethinking their symmetries* (2022) Huang et al.: *Boosting the cycle counting power of graph neural networks with I2-GNNs* (2022) Papp et al.: DropGNN: Random dropouts increase the expressiveness of graph neural networks. (2021) Qian et al.: *Ordered subgraph aggregation networks.* (2022) You et al.: *Identity-aware graph neural networks*. (2021) Zhang and P. Li. *Nested graph neural networks* (2021) Zhao et al.: *From stars to subgraphs: Uplifting any GNN with local structure awareness* (2022)

Takeaway message #1: Classification in terms of number of variables

G., Reutter: *Expressiveness and approximation properties of graph neural networks.* (2022)

ℋ

Graphormer

PPGN

CWN

 $k+1-IGNs$ k-GNNs k-FGNNs

rando Simplicial MPNNs k-LGNNs

ChebNet 2-IGN GIN CayleyNet GATs Id-aware GNN *δ* − *k*−GNNs Walk GNNs

Dropout GNN MPNNs MPNN+

GraphSage GCN GIN

k-GNNs

SGNs

GatedGCNs

Reconstruction GNNs

Ordered subgraph Networks

GNN as Kernel Nested GNNs

How to compare different classes?

★ How to compare such embedding classes theoretically?

How to bring order to the chaos?

1. See graph embedding methods as queries in some query language

> 2. Analyse expressive power of query language

3. Transfer understanding back to graph learning world

Which language?

How to compare different classes?

★ How to compare such embedding classes theoretically?

How to bring order to the chaos?

1. See graph embedding methods as queries in some query language

> 2. Analyse expressive power of query language

3. Transfer understanding back to graph learning world

GEL GGEL *k* 2

Which language?

Expressive power

NESSA RASAN NYANG POASIA POASIA NYANG ARAWA NYANG MASANA ARAWAN NYANG MASAN NYANG ALANG NYANG MASANG NYANG NYANG

Distinguishing power

Which inputs can be separated/distinguished by embeddings in \mathcal{H} ?

Captured by the following equivalence relation on $\mathcal{G} \times \mathcal{V}^p$:

$\rho(\mathcal{H}) := \{ (G, v, H, w) \mid \forall \xi \in \mathcal{H} : \xi(G, v) = \xi(H, w) \}$

Distinguishing power

Which inputs can be separated/distinguished by embeddings in \mathcal{H} ?

Captured by the following equivalence relation on $\mathcal{G} \times \mathcal{V}^p$: $\rho(\mathcal{H}) := \{ (G, v, H, w) \mid \forall \xi \in \mathcal{H} : \xi(G, v) = \xi(H, w) \}$

Strongest power: $\mathcal H$ powerful enough to detect non-isomorphic graphs: $\rho(\mathcal{H})$ only contains isomorphic pairs

Weakest power: H cannot differentiate any two graphs: ρ (H) contains all pairs of graphs.

Distinguishing power

Allows for comparing different classes of embeddings methods

 ρ (methods₁) \subseteq ρ (methods₂)

methods, is more powerful than methods methods₂ is bounded by methods₁ in power 1 12 mois poweriai man memod? 2 12 DOUTIULU DY HILUTOUS

 ρ (methods₁) = ρ (methods₂)

Both methods are as powerful Allows for comparing embedding methods with algorithms, logic, …

on graphs

Expressive power in ML community

Focus has been on characterising the distinguishing power of classes $\mathscr H$ of embedding methods.

Hopefully, characterisations of $\rho(\mathcal{H})$ shed light on what graph properties a learning method in H can detect/use.

Logic

 $\left(E(y, x) \wedge L_a(x) \right)$

binary edge predicate unary label predicate

each with at least two neighbours labeled "a"

First-order logic with k variables and counting quantifiers (C_k) .

$$
k=2 \qquad \varphi(x) = \exists^{\leq 5} y \left(E(x, y) \wedge E(x, y) \right)
$$

Given graph *G*, vertex $v \in V_G$ satisfies φ : It has at most 5 neighbours

Guarded fragment GC₂ of only existential quantification for the form $\exists^{\geq n} y(E(x, y) \land \varphi(y))$ 2 O1 \mathcal{L}_2

Logic $We can consider $\rho(C_k)$ and $\rho(GC_2)$$ $\rho(C_k) := \{ (G, \mathbf{v}, H, \mathbf{w}) \mid \forall \varphi \in C_k : (G, \mathbf{v}) \models \varphi \Longleftrightarrow (H, \mathbf{w}) \models \varphi) \}$

The distinguishing power of these logics is well understood

Cannot distinguish d-regular graphs Cannot count cycles (triangles) Only tree information

Arvind et al.: *On the power of color refinement* (2015) Images: Wolfram MathWorld, Christopher Morris

Expressive power of GEL: Main result

Theorem (G. and Reutter 2022)

activation functions) and Θ contains summation

Theorem (Xu et al. 2019, Morris et al. 2019, G. and Reutter 2022)

Xu, Hu, Leskovec, Jegelka: *How powerful are graph neural networks?* (2019) Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, Grohe: *Weisfeiler and Leman go neural: Higher-order graph neural networks.* (2019 Hella, Libkin, Nurmonen, Wong: *Logics with Aggregates.* (2001) Cai, Fürer, Immerman: *An optimal lower bound on the number of variables for graph identification*. (1992) G., Reutter: Expressiveness and approximation properties of graph neural networks. (2022) M. Grohe: *The logic of graph neural networks.* (2021)

 $\rho(GGL(\Omega,\Theta)) = \rho(GC_2)$

 $\rho(\text{GEL}_k(\Omega, \Theta)) = \rho(C_k)$

Lower bounds: Ω contains linear combinations, concatenation, product (or

The following results follow from standard analysis of aggregate query languages: all real number arithmetic can be eliminated.

Can we train a GNN 101 which embeds G differently from H?

$\rho(GNN101) = \rho(C_2)$ Theorem (Morris et al. 2019)

Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, Grohe: *Weisfeiler and Leman go neural: Higher-order graph neural networks.* (2019

Can we train a GNN 101 which embeds G differently from H?

$\rho(GNN101) = \rho(C_2)$ Theorem (Morris et al. 2019)

Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, Grohe: *Weisfeiler and Leman go neural: Higher-order graph neural networks.* (2019

NO!

GNN 101

Can we train a GNN 101 which embeds G differently from H?

$\rho(GNN101) = \rho(C_2)$ Theorem (Morris et al. 2019)

G and H are known to be indistinguishable by C,

 \Rightarrow $(G, H) \in \rho(C_2) = \rho(GNN101)$

Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, Grohe: *Weisfeiler and Leman go neural: Higher-order graph neural networks.* (2019

NO!

GNN 101s, MPNNs are pretty weak

GNN 101

Can we train a GNN101 such that P embeds differently from NP?

$\rho(\text{GNN101}) = \rho(\text{C}_2)$ Theorem (Morris et al. 2019)

Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, Grohe: *Weisfeiler and Leman go neural: Higher-order graph neural networks.* (2019

GNN 101

Can we train a GNN101 such that P embeds differently from NP?

$\rho(\overline{GNN101}) = \rho(\overline{C_2})$ Theorem (Morris et al. 2019)

Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, Grohe: *Weisfeiler and Leman go neural: Higher-order graph neural networks.* (2019

YES!

Can we train a GNN101 such that P embeds differently from NP?

$\rho(GNN101) = \rho(C_2)$ Theorem (Morris et al. 2019)

P satisfies $\exists^{-1} x \exists^{-1} y E(x, y)$ but NP does not ⇒ $(P, NP) \notin \rho(C_2) \Rightarrow (P, NP) \notin \rho(GNN101)$ single degree one node

Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, Grohe: *Weisfeiler and Leman go neural: Higher-order graph neural networks.* (2019

GNN 101

YES!

Consequences

If embedding method M can be cast in $GEL_k(\Omega, \Theta)$ then $\rho(C_k) \subseteq \rho(M)$

If embedding method M can also encode formulas in C_k then $\rho(C_k) \supseteq \rho(M)$

2-IGN ChebNet Graphormer PPGN CWN Dropout GNN CayleyNet GATs Id-aware GNN GCN GIN GraphSage $k+1-IGNs$ *δ* − *k*−GNNs k-GNNs MPNNs MPNN+ rando Simplicial MPNNs k-FGNNs k-LGNNs SGNs GatedGCNs Walk GNNs Reconstruction GNNs Ordered subgraph Networks GNN as Kernel Nested GNNs GGEL $GEL₂$ GEL₃ GEL*k*

 $\rho(MPNNs) = \rho(C_2)$

Other - more insightful - characterisations?

What else can we say?

 $\rho(MPNNs) = \rho(C_2)$

Other - more insightful - characterisations?

What else can we say?

homomorphism counts

Homomorphisms Let $P = (V_P, E_P, L_P)$ and $G = (V_G, E_G, L_G)$ be graphs.

A function $h: V_p \to V_G$ is a homomorphism if it is edge preserving (v, w) ∈ E_p ⇒ $(h(v), h(w))$ ∈ E_G and label preserving.

$Define $Hom(P, G) := \{ all homomorphisms from P to G\}$$

$Define $Hom(P, G) := \{ all homomorphisms from P to G\}$$

$Define $Hom(P, G) := \{ all homomorphisms from P to G\}$$

$Define $Hom(P, G) := \{ all homomorphisms from P to G\}$$

Graph isomorphisms and homomorphisms

Two graph G and H are isomorphic if and only if for all graphs *P* $hom(P, G) = hom(P, H)$ Theorem (Lovász 1967)

Graph isomorphisms and homomorphisms

Two graph G and H are isomorphic if and only if for all graphs *P* $hom(P, G) = hom(P, H)$ Theorem (Lovász 1967)

Homomorphism count information is very insightful and a complete invariant.

Graph isomorphisms and homomorphisms

Two graph G and H are isomorphic if and only if for all graphs *P* $hom(P, G) = hom(P, H)$ Theorem (Lovász 1967)

Homomorphism count information is very insightful and a complete invariant.

What if we do not consider all graphs *P*?

Homomorphism count vectors

\cdot We can consider *ρ*(\mathcal{P}) for a set of graphs \mathcal{P} :

$\rho(\mathscr{P}) := \{ (G, H) \mid \forall P \in \mathscr{P} : \text{hom}(P, G) = \text{hom}(P, H) \}$

Theorem (Lovász 1967)

Two graph G and H are isomorphic if and only if $f(G, H) \in \rho(\mathcal{P})$ for \mathcal{P} the set of all graphs.

Theorem (Dell et al. 2019, Dvorák 2010)

if and only if $Hom(T, G) = hom(T, H)$ for all *trees* T *if and only if* $f(G,H) \in \rho(\mathcal{T})$ for \mathcal{T} the set of all trees. $(G, H) \in \rho(C_2)$

Z. Dvoräk: *On recognizing graphs by numbers of homomorphisms.* (2010) Dell, Grohe, Rattan: *Lovász meets Weisfeiler and Leman.* (2018) Cai, Fürer, Immerman: *An optimal lower bound on the number of variables for graph identification*. (1992) M. Grohe: *The logic of graph neural networks.* (2021)

Important class of MPNNs can only detect tree-based information

Theorem (Dell et al. 2019, Dvorák 2010)

if and only if $Hom(T, G) = hom(T, H)$ for all *trees* T *if and only if* $f(G,H) \in \rho(\mathcal{T})$ for \mathcal{T} the set of all trees. $(G, H) \in \rho(C_2)$

All embedding methods in GEL₂(Ω, Θ) can only distinguish graphs based on tree **information!**

Z. Dvoräk: *On recognizing graphs by numbers of homomorphisms.* (2010) Dell, Grohe, Rattan: *Lovász meets Weisfeiler and Leman.* (2018) Cai, Fürer, Immerman: *An optimal lower bound on the number of variables for graph identification*. (1992) M. Grohe: *The logic of graph neural networks.* (2021)

Important class of MPNNs can only detect tree-based information

if and only if $f(P, G) = \text{hom}(P, H)$ for all *graphs P of treewidth* k *if and only if* $f(G,H) \in \rho(\mathcal{T}_k)$ for \mathcal{T}_k the set of all graphs of tree width k $(G, H) \in \rho(C_{k+1})$

Z. Dvoräk: *On recognizing graphs by numbers of homomorphisms.* (2010) Dell, Grohe, Rattan: *Lovász meets Weisfeiler and Leman.* (2018) Cai, Fürer, Immerman: *An optimal lower bound on the number of variables for graph identification*. (1992) M. Grohe: *The logic of graph neural networks.* (2021)

Theorem (Dell et al. 2019, Dvorák 2010)

if and only if $f(P, G) = \text{hom}(P, H)$ for all *graphs P of treewidth* k *if and only if* $f(G,H) \in \rho(\mathcal{T}_k)$ for \mathcal{T}_k the set of all graphs of tree width k $(G, H) \in \rho(C_{k+1})$

Z. Dvoräk: *On recognizing graphs by numbers of homomorphisms.* (2010) Dell, Grohe, Rattan: *Lovász meets Weisfeiler and Leman.* (2018) Cai, Fürer, Immerman: *An optimal lower bound on the number of variables for graph identification*. (1992) M. Grohe: *The logic of graph neural networks.* (2021)

All embedding methods in $GEL_k(\Omega, \Theta)$ can only distinguish graphs based on **treewidth** $k - 1$ pattern information!

Theorem (Dell et al. 2019, Dvorák 2010)

A k-tree is a graph that can be obtained starting from a (k+1)-clique and then iteratively adding a vertex connected to a k-clique

 $k=2$

A k-tree is a graph that can be obtained starting from a (k+1)-clique and then iteratively adding a vertex connected to a k-clique

\triangle A k-tree is a graph that can be obtained starting from a $(k+1)$ -clique and then iteratively adding a vertex connected to a k-clique

\triangle A k-tree is a graph that can be obtained starting from a $(k+1)$ -clique and then iteratively adding a vertex connected to a k-clique

\triangle A k-tree is a graph that can be obtained starting from a $(k+1)$ -clique and then iteratively adding a vertex connected to a k-clique

A k-tree is a graph that can be obtained starting from a (k+1)-clique and then iteratively adding a vertex connected to a k-clique

A partial k-tree is a subgraph of a k-tree

A k-tree is a graph that can be obtained starting from a $(k+1)$ -clique and then iteratively adding a vertex connected to a k-clique

A partial k-tree is a subgraph of a k-tree

Treewidth of a graph is smallest k such that the graph is a partial k-tree

and then iteratively adding a vertex connected to a k-clique

A partial k-tree is a subgraph of a k-tree

 \cdot Trees=Treewidth 1

A k-tree is a graph that can be obtained starting from a $(k+1)$ -clique

Treewidth of a graph is smallest k such that the graph is a partial k-tree

More connections

To combinatorial graph algorithms color refinement and higherdimensional Weisfeiler-Leman graph isomorphism tests.

To linear algebraic congruences between adjacency matrices and systems of equations.

To distance measures on graphs and metric equivalences.

Z. Dvoräk: *On recognizing graphs by numbers of homomorphisms.* (2010) Dell, Grohe, Rattan: *Lovász meets Weisfeiler and Leman.* (2018) Cai, Fürer, Immerman: *An optimal lower bound on the number of variables for graph identification*. (1992) M. Grohe: *The logic of graph neural networks.* (2021)

Takeaway message #2: Classification in terms of logic, homomorphism counts, ...

G., Reutter: Expressiveness and approximation properties of graph neural networks. (2022)

ℋ

Graphormer

PPGN

CWN

 $k+1-IGNs$ rando k-GNNs k-FGNNs

ChebNet 2-IGN GIN GATs Id-aware GNN *δ* − *k*−GNNs Walk GNNs

Dropout GNN MPNNs MPNN+

GraphSage GCN GIN

k-GNNs

CayleyNet Simplicial MPNNs k-LGNNs

SGNs

GatedGCNs

Reconstruction GNNs

Ordered subgraph Networks

GNN as Kernel Nested GNNs

Expressive power

Which inputs can be separated/distinguished by embeddings in $\mathcal X.$

Which embeddings can be approximated by embeddings in \mathcal{H} ?

\mathcal{H} = class of embedding methods

Approximation properties

Equip the set of graphs $\mathcal G$ with a topology and assume that $\mathcal H$ consists of continuous graph embeddings from $\mathcal G$ to R.

Let & ⊆ S be a compact set of graphs.

Azizian, Lelarge: *Characterizing the expressive power of invariant and equivariant graph neural networks* (2021) G., Reutter: *Expressiveness and approximation properties of graph neural networks* (2022)

Approximation properties

Equip the set of graphs $\mathcal G$ with a topology and assume that $\mathcal H$ consists of continuous graph embeddings from $\mathcal G$ to R.

Let & ⊆ S be a compact set of graphs.

If $\mathcal X$ is closed under linear combinations and product, then $\mathscr H$ can approximate any continuous function Ξ : $\mathscr{C} \to \mathbb{R}$ satisfying $\rho(\mathcal{H}) \subseteq \rho(\{\Xi\})$. Theorem (Azizian & Lelarge 2021, G. and Reutter 2022)

Azizian, Lelarge: *Characterizing the expressive power of invariant and equivariant graph neural networks* (2021) G., Reutter: *Expressiveness and approximation properties of graph neural networks* (2022)

Can be generalised to embeddings with output space ℝ*^d*

MPNNs: Approximation

On compact set of graphs, MPNNs can approximate any continuous graph

$\text{embedding } \Xi : \mathscr{C} \to \mathbb{R} \text{ satisfying } \rho(C_2) \subseteq \rho(\{\Xi\})$ Theorem

$(G, H) \in \rho(MPNN) \Rightarrow$ connected components Cannot approximate graph functions based on - 3-cliques

Azizian, Lelarge: *Characterizing the expressive power of invariant and equivariant graph neural networks* (2021) G., Reutter: *Expressiveness and approximation properties of graph neural networks* (2022)

Intricate relation between distinguishing power and approximation properties

-

Expressive power

Which inputs can be separated/distinguished by embeddings in $\mathcal X.$ Which embeddings can be approximated by embeddings in \mathcal{H} ? What is the VC dimension of \mathcal{H} ?

\mathcal{H} = class of embedding methods

VC dimension

We define the VC dimension of \mathcal{H} on $\mathcal{G}' \subseteq \mathcal{G}$ as

 \mathcal{H} := max{*s* | ∃ G_1 , ..., G_s in \mathcal{G}' which can be shattered by \mathcal{H} }

Theorem (Morris et al. 2023)

for some hypothesis classes also equality holds. $\mathscr{C}(\mathscr{H}) \leq |\mathscr{G}'|_{\rho(\mathscr{H})}|$

Morris, G.,Tönshoff, Grohe; *WL meet VC* (2023).

A set of graphs $G_1, ..., G_s$ can be shattered by $\mathcal H$ if for any boolean $\text{vector } \tau \in \{0,1\}^s$, there is a $\xi_{\tau} \in \mathcal{H}$ such that $\xi_{\tau}(G_i) = \tau_i$ for all $i = 1,...,s$

Equivalence classes induced by *ρ*(ℋ)

Expressive power

Which embeddings can be approximated by embeddings in \mathcal{H} ? What is the VC dimension of \mathcal{H} ? Which embeddings can be expressed by embeddings in \mathcal{H} ?

\mathcal{H} = class of embedding methods

Which inputs can be separated/distinguished by embeddings in $\mathcal X.$

Which unary *C*₂ formulas can MPNNs express?

\Rightarrow Not all: $\varphi(x) := L_b(x) \wedge \exists y L_r(y)$

I am blue and there exist a red vertex somewhere…

$component₁$

 \bigcirc

component₂

Which unary *C*₂ formulas can MPNNs express?

\Rightarrow Not all: $\varphi(x) := L_b(x) \wedge \exists y L_r(y)$

Cannot be reached by neighborhood aggregation

I am blue and there exist a red vertex somewhere…

$componient₁$

component₂

Which unary *C*₂ formulas can MPNNs express?

Let $\varphi(x)$ be a unary C_2 formula. Then, $\varphi(x)$ is equivalent to a $_2$ formula *if and only if* $\varphi(x)$ *is expressible by the class of* MPNNs. Theorem (Barceló et al. 2020)

$\exists \xi \in \text{MPNN}: \forall G \in \mathcal{G}, \forall v \in V_G : (G, v) \models \varphi \Leftrightarrow \xi(G, v) = 1$

Barceló, Kostylev, Monet, Pérez, Reutter, Silva: *The logical expressiveness of graph neural networks* (2020) Barceló, Kostylev, Monet, Pérez, Reutter, Silva: The Expressive Power of Graph Neural Networks as a Query Language. (2020)

MPNNs+

Every unary C_2 formula $\varphi(x)$ is expressible by the class of MPNNs+ Theorem (Barceló et al. 2020)

Barceló, Kostylev, Monet, Pérez, Reutter, Silva: *The logical expressiveness of graph neural networks* (2020) Barceló, Kostylev, Monet, Pérez, Reutter, Silva: The Expressive Power of Graph Neural Networks as a Query Language. (2020)

Of course, there are queries beyond C_2 which MPNNs can express

Allow for aggregation over all vertices not only edge-guarded
If a unary query Q is computable by a GNN with rational weights and piecewise linear activation functions, then Q is definable in the guarded Descriptive complexity of GNNs Theorem (Grohe 2023)

fragment of $FO₂ + C$

Extends to general GNNs with real weights and more complex activation functions \Rightarrow approximate with GNNs as in theorem

M. Grohe. *The Descriptive Complexity of Graph Neural Networks* (2023)

Different from Two sorted logic, numerical predicates etc. 2

If a unary query Q is computable by a GNN with rational weights and piecewise linear activation functions, then Q is definable in the guarded Descriptive complexity of GNNs Theorem (Grohe 2023)

M. Grohe. *The Descriptive Complexity of Graph Neural Networks* (2023)

fragment of $FO₂ + C$

Boolean functions computable by non-uniform polynomial-size bounded-depth family of circuits with threshold gates

Different from Two sorted logic, numerical predicates etc. 2

Extends to general GNNs with real weights and more complex activation functions \Rightarrow approximate with GNNs as in theorem Situates queries expressible by GNNs in (non-uniform) TC⁰

If a unary query Q is computable by a GNN with rational weights and piecewise linear activation functions, then Q is definable in the guarded Descriptive complexity of GNNs Theorem (Grohe 2023)

fragment of $FO₂ + C$

Extends to general GNNs with real weights and more complex activation functions \Rightarrow approximate with GNNs as in theorem \cdot Situates queries expressible by GNNs in (non-uniform) TC⁰

M. Grohe. *The Descriptive Complexity of Graph Neural Networks* (2023)

Boolean functions computable by non-uniform polynomial-size bounded-depth family of circuits with threshold gates

Different from Two sorted logic, numerical predicates etc. 2

Converse holds, with random vertex features.

Takeaway message #3: Classification along different dimensions of expressibility, not only distinguishabilty

G., Reutter: *Expressiveness and approximation properties of graph neural networks.* (2022)

ℋ

Graphormer

PPGN

 $k+1-IGNs$ rando k-GNNs k-FGNNs

ChebNet 2-IGN GIN CWN Dropout GNN GATs Id-aware GNN *δ* − *k*−GNNs Walk GNNs

GraphSage GCN GIN

k-GNNs

CayleyNet Simplicial MPNNs k-LGNNs

MPNNs MPNN+

SGNs GatedGCNs

Reconstruction GNNs

Ordered subgraph Networks

GNN as Kernel Nested GNNs

How to compare different classes?

How to compare such embedding classes theoretically?

How to bring order to the chaos?

2. Analyse expressive power of query language

1. See graph embedding methods as queries in some query langu Distinguishability, approximation, generalisation,

3. Transfer understanding back to graph learning world

uniform and non-uniform expressiveness

How to compare different classes?

 \cdot How to compare such embedding classes theoretically?

How to bring order to the chaos?

2. Analyse expressive power of query language

1. See graph embedding methods as queries in some query langu Distinguishability,

3. Transfer understanding back to graph learning world

GEL

approximation, generalisation, uniform and non-uniform expressiveness

Conclusion

MPNNs: Efficient, most widely used but not expressive (C_2) Methods matching C_k for $k > 2$ require tensors making them inefficient 2

Ongoing efforts to boost power but preserve efficiency

MPNNs: Efficient, most widely used but not expressive (C_2) 2

Methods matching C_k for $k > 2$ require tensors making them inefficient

• Ongoing efforts to boost power but preserve efficiency

Barceló et al.: *Graph neural networks with local graph parameters.* (2021) Bouritsas et al.: *Improving graph neural network expressivity via subgraph isomorphism counting* (2020)

Feature augmentation

Kreuzer et al.: *Rethinking graph transformers by spectral attention* (2021) Ying et al.: *Do transformers really perform bad for graph representation* (2021) Lim et al.: *Sign and Basis Invariant Networks for Spectral Graph Representation Learning* (2022) Zhang et al.: Rethinking the expressive power of gnns via graph biconnectivity (2023)]²

Precompute hom/iso counts

Random features

Dasoulas et al.: *Coloring graph neural networks for node disambiguation* (2020) Sato et al.: *Random features strengthen graph neural networks* (2021). Abboud et al. : The surprising power of graph neural networks with random node initialization. (2021)

Spectral/Global properties

MPNNs: Efficient, most widely used but not expressive (C_2) 2

Methods matching C_k for $k > 2$ require tensors making them inefficient

• Ongoing efforts to boost power but preserve efficiency

Feature augmentation

Precompute hom/iso counts

Barceló et al.: *Graph neural networks with local graph* Bouritsas et al.: *Improving graph neural network expressivity via* raph isomorphism counting (2020)

Running graph learning
Dasoulas et al.: *Coloring gra*ph Sato et al.: *Random featur* Sato et al.: *Random featur* **power of a powerful condition of a condition of a condition of graph** $\sum_{i=1}^{n}$ Kreuzer et al.: *Rethinking graph* Random features Spectral/Glo Analysis of expressive power (logic, hom count,...)

Ying et al.: *Do transformers really p* Lim et al.: *Sign and Basis Invariant Netwo* Zhang et al.: Rethinking the expressive power of gnns via

MPNNs: Efficient, most widely used but not expressive (C_2) 2

Methods matching C_k for $k > 2$ require tensors making them inefficient

Ongoing efforts to boost power but preserve efficiency

Subgraph GNNs

Bevilacqua et al: *Equivariant subgraph aggregation network* (2022) Cotta et al.: *Reconstruction for powerful graph representations* (2021) Bevilacqua et al.: *Understanding and extending subgraph GNNs by rethinking their symmetries* (2022) Huang et al.: *Boosting the cycle counting power of graph neural networks with I2-GNNs* (2022) Papp et al.: *DropGNN: Random dropouts increase the expressiveness of graph neural networks.* (2021) Qian et al.: *Ordered subgraph aggregation networks.* (2022) You et al.: *Identity-aware graph neural networks*. (2021) Zhang and P. Li. *Nested graph neural networks* (2021) Zhao et al.: *From stars to subgraphs: Uplifting any GNN with local structure awareness* (2022)

Feature augmentation

Precompute hom/iso counts

Barceló et al.: *Graph neural networks with local graph* Bouritsas et al.: *Improving graph neural network expressivity via* raph *isomorphism counting* (2020)

Running graph learning
Dasoulas et al.: *Coloring gra*ph Sato et al.: *Random featur* Sato et al.: *Random featur* **power of a powerful condition of a condition of a condition of graph** $\sum_{i=1}^{n}$ Kreuzer et al.: *Rethinking graph* Random features Spectral/Glo Analysis of expressive power (logic, hom count,...)

Ying et al.: *Do transformers really p* Lim et al.: *Sign and Basis Invariant Netwo* Zhang et al.: Rethinking the expressive power of gnns via

MPNNs: Efficient, most widely used but not expressive (C_2) 2

Methods matching C_k for $k > 2$ require tensors making them inefficient

Ongoing efforts to boost power but preserve efficiency

Subgraph GNNs

Bevilacqua et al: Equivariant subgraph aggregation network ²²) Cotta et al.: *Reconstruction for powerful graph representations* Bevilacqua et al.: *Understanding and extending subgraph GN_N rethinking their symmetries* (2022) Huang et al.: *Boosting the cycle counting power of*

Papp et al.: DropGNN: Random drop-Qian et al.: Ordered subgraph You et al.: *Identity-aware* Zhang and P. Li. Nest

Feature augmentation

Running graph learning
Dasoulas et al.: *Coloring gra*ph Sato et al.: *Random featur* Sato et al.: *Random featur* **power of a powerful condition of a condition of a condition of graph** $\sum_{i=1}^{n}$ Kreuzer et al.: *Rethinking graph* Random features Spectral/Glo Analysis of expressive power (logic, hom count,…)

Ying et al.: *Do transformers really p* Lim et al.: *Sign and Basis Invariant Netwo* Zhang et al.: Rethinking the expressive power of gnns via Zhao et al.: *From sta*¹ **to subgraphs: 100 mg/mn/ many views.** t Running graph learning method on many views, then aggregate. Analysis of expressive power

Precompute hom/iso counts

Barceló et al.: *Graph neural networks with local graph* Bouritsas et al.: *Improving graph neural network expressivity via* raph *isomorphism counting* (2020)

Number of variables depends on GEL skills, are there better notions?

Analysis does not always explain experiments. Is a more fine grained analysis possible, perhaps taking learning process into account?

- Specialized homomorphism count characterizations, more fine grained than logic?
-
- Relational embedding methods.
- * Recurrent GNNs are closely related to fixpoint computations. Relationship to query languages with recursion?

Recipe for upper bounding architectures

3.Then the *power* of your architecture is **bounded** $\mathbf{b}\mathbf{v}$ \mathbf{C}_k

1. **Take you GNN** architecture and **write it in GEL**, but using *a minimal number of variables* of variables.

2.Call this *number* of variables **k**.