Lecture 9. EN.553.744 Data Science for Large-Scale Graphs

Prof. Luana Ruiz

Today:

- GATs
- Expressivity in graph-level tasks
- Graph isomorphism
- Weisfeiler-Leman test

▶ A final, non-convolutional GNN: graph attention (GAT)

$$[X_\ell]_i := \sigma\Big(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\,[X_{\ell-1}]_j\, H_\ell\Big)$$

where α_{ij} are the graph attention coefficients, computed as:

$$\begin{cases} \alpha_{ij} = \text{softmax}_{\mathcal{N}(i)}(e_{ij}) \\[2ex] e_{ij} = \dfrac{\overbrace{\rho}^{\text{nonlinearity}}\big(\vec{a}\,(x_i H' \overbrace{\|}^{\text{concatenation}} \underbrace{x_j H'}_{\in \mathbb{R}^d})\big)}{\sum_{j'\in\mathcal{N}(i)} \rho\big(\vec{a}\,(x_i H' \| x_{j'} H')\big)} \end{cases}$$

- $a \in \mathbb{R}^{1\times 2d}$
- ρ is typically the Leaky ReLU $\rho(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x < 0 \end{cases}$

The learnable parameters are: $\vec{a}$, H_ℓ for $1 \le \ell \le L$, H'

GAT layer $[X_\ell]_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} [X_{\ell-1}]_j H_\ell\right)$ thought of as a conv. layer where S, the GSO, is learned (i.e. $S_{ij} = \alpha_{ij}$)

↳ This add't'l flexibility increases the capacity of the architecture — observed empirically — but comes at cost of more expensive forward pass ⟶ especially for large & dense graphs since need to compute $|\mathcal{E}|$ coeffs. α_{ij}

→ all available / implemented in PyTorch geometric

▶ The "readout" in graph-level tasks

In node-level tasks (e.g. source localization, community detection, citation networks, etc.), both the input & output are graph signals $X \in \mathbb{R}^{n \times d_0}$, $Y \in \mathbb{R}^{n \times d_L}$. so the map $\Phi_{\mathcal{H}}$ is composed strictly of GNN layers

↳ ensures a parametrization that is independent of graph size n

In graph-level tasks, the output does not need to be a graph signal, it could be $y \in \mathbb{R}$, $y \in \{0, 1, \dots, C\}$ or $y \in \mathbb{R}^d$

How to map GNN layers to such outputs?

↳ readout layers

•) Option 1: a fully connected layer

→ L GNN layers of the form:

$$X_\ell = \sigma\left(\sum_{k=0}^{K-1} S^k X_{\ell-1} H_{\ell,k}\right)$$

→ followed by: $Y = \rho\left(C \cdot \underbrace{\mathrm{vec}(X_L)}_{\in \mathbb{R}^{nd_L}}\right)$ 1-layer perceptron

$C \in \mathbb{R}^{d \times nd_L}$

↳ $\mathrm{vec}(\cdot)$ vectorizes $\mathbb{R}^{u \times v}$ matrices → $\mathbb{R}^{uv}$ vectors

↳ ρ can be identity or ReLU, softmax etc...

Downsides of a fully connected readout layer:

- adds $n d_L d$ learning params → grows with graph size
- not permutation invariant Ex.: verify
- not transferable across graphs

- Option 2: aggregation

→ L GNN layers of the form:

$$X_\ell = \sigma\left(\sum_{k=0}^{K-1} S^k X_{\ell-1} H_{\ell,k}\right)$$

→ followed by: $y = \text{aggr}(X_L C)$ $\quad C \in \mathbb{R}^{d_L \times d}$

$$\text{aggr} : \mathbb{R}^{n \times d} \to \mathbb{R}^{1 \times d} \quad (\mathbb{R}^d)$$

(node-level aggregation)

Typical aggregation functions: mean, sum, max
(min, median, etc...)

↳ parametrization independent of n
↳ permutation invariant
↳ transferable across graphs

▶ Graph isomorphism problem

Let G and G' be two graphs. A graph isomorphism between G and G' is a bijection
$$f : V(G) \to V(G')$$
s.t. $\forall\, i,j \in V(G)$,

$$\boxed{(i,j) \in \mathcal{E}(G) \iff (f(i), f(j)) \in \mathcal{E}(G')}$$

↳ Can we train a GNN to detect if graphs are isomorphic? I.e., to produce identical outputs for graphs in the same equivalence class (i.e., which are isomorphic) and $\neq$ outputs for non-isomorphic graphs?

→ Consider two graphs G & G' with $L = V\Lambda V^T$ and $L' = V'\Lambda' V'^T$

→ Assume they do not have node features (but we can impute them)

→ Consider the 1-layer GNN (linear):

$$y = \sum_{k=0}^{K-1} h_k L^k x \qquad (*)$$

→ Suppose $\exists \lambda_j$ s.t. $\lambda_j \neq \lambda_i' \quad \forall i \qquad (**)$

How can we use GNN $(*)$ to solve the graph isomorphism problem for this pair?

•) Pick x white, i.e., s.t. $\mathbb{E}([\hat{x}_i]) = 1 \quad \forall i$

•) Set $h_k = \begin{cases} 1 & \text{if } k = 1 \\ 0 & \text{o.w.} \end{cases}$

Then: $\mathbb{E}(\hat{y}) = \mathbb{E}(V^T y)$

$$= \mathbb{E}(V^T \cancel{h_1} L x)$$

$$= \mathbb{E}(\cancel{V^T} \cancel{V} \Lambda V^T x)$$

$$= \mathbb{E}(\Lambda \hat{x}) = \Lambda \mathbb{1}$$

$$\Rightarrow \quad \mathbb{E}[\hat{y}] = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \vdots \\ \lambda_n \end{bmatrix}$$

" GNNs are more powerful than we think" (Kanatsoulis et al.)

for implementation in node domain

If (**) is satisfied, it suffices to compare:

sum, or aggregation in the spectral domain

$$\mathbb{E}(\mathbb{1}^T \hat{y}) = \sum_i \lambda_i \quad \text{with}$$

$$\mathbb{E}(\mathbb{1}^T \hat{y}') = \sum_i \lambda_i'$$

$\neq$

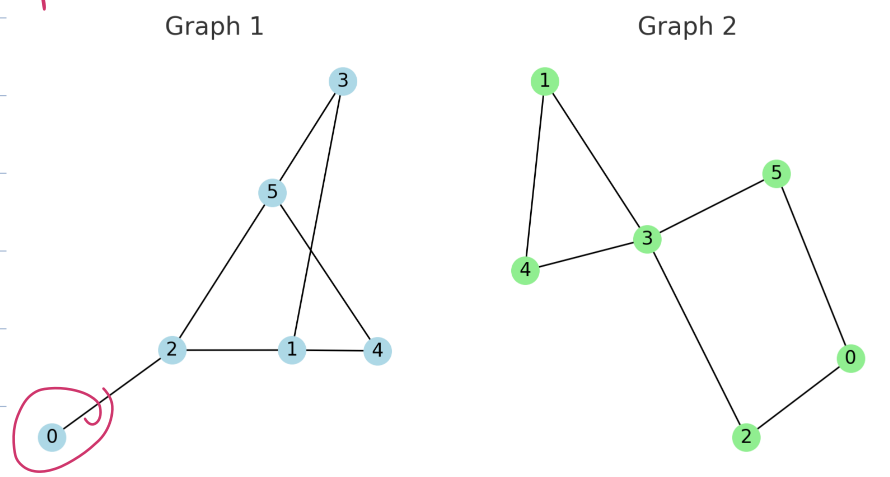
to verify $G \not\cong G'$

Hence: As long as graphs' eigenvalues are $\neq$, convolutional GNNs can distinguish them.

But what if the eigenvalues coincide?

There are many non-isomorphic graphs sharing the same eigenvalues

Example :



clearly non isomorphic

but their Laplacian eigenvalues are the same :

$$\lambda_1 = 0; \ \lambda_2 = 0.764; \ \lambda_3 = 2; \ \lambda_4 = \lambda_5 = 3; \ \lambda_6 = 5.236$$

Exercise : verify !

Though our eigenvalue-based heuristic didn't work for this and other examples, a simple heuristic that works is counting degrees

For G_1, $\{1, 3, 3, 2, 2, 3\}$

For G_2, $\{2, 2, 2, 4, 2, 2\}$

This is the idea behind the (1-) Weisfeiler-Leman test

The Weisfeiler-Leman graph isomorphism test

The WL test consists of running the "color refinement algorithm" for two graphs G & G'. If the "colors"/representations produced by WL are $\neq$, then G & G' are non-isomorphic. O.w., we don't know.

→ Given a graph $G = (\mathcal{V}, \mathcal{E})$ with node features X, do:

$c_i(0) = f(\cdot, \{x_i\}) \quad \forall i \in \mathcal{V}$

repeat:

hash function

multisets

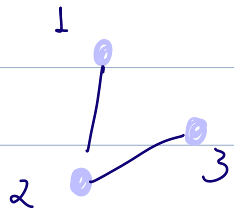
$c_i(t) = f(c_i(t-1), \{\{c_j(t-1), j \in \mathcal{N}(i)\}\}) \quad \forall i \in \mathcal{V}$

until colors cease to change (say at $t = T$)

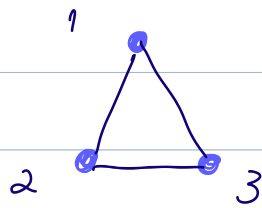
return $\{\{c_i(T) \; \forall i \in \mathcal{V}\}\}$

Example:

G_1



G_2



Color refinement for G_1:

0) $c_1(0) = c_2(0) = c_3(0) = f(0, 1) = {\color{green}\bullet}$

1) $c_1(1) = f({\color{green}\bullet}, \{{\color{green}\bullet}\}) = {\color{green}\bullet}$

$c_3(1) = f({\color{green}\bullet}, \{{\color{green}\bullet}\}) = {\color{green}\bullet}$

$c_2(1) = f({\color{green}\bullet}, \{{\color{green}\bullet}, {\color{green}\bullet}\}) = {\color{red}\bullet}$

2) $c_1(2) = f({\color{green}\bullet}, \{{\color{red}\bullet}\}) = {\color{green}\bullet}$

$c_3(2) = f({\color{green}\bullet}, \{{\color{red}\bullet}\}) = {\color{green}\bullet}$

$c_2(2) = f({\color{red}\bullet}, \{{\color{green}\bullet}, {\color{green}\bullet}\}) = {\color{red}\bullet}$

return $\{\{{\color{green}\bullet}, {\color{red}\bullet}, {\color{green}\bullet}\}\}$

Color refinement for G_2:

0) $c_1(0) = c_2(0) = c_3(0) = f(0, 1) = {\color{green}\bullet}$

1) $c_1(1) = c_2(1) = c_3(1) = f({\color{green}\bullet}, \{\{{\color{green}\bullet}, {\color{green}\bullet}\}\}) = {\color{green}\bullet}$

return $\{\{\bullet, \bullet, \bullet\}\}$

Hence non-isomorphic !