Thứ
Ngày . . No.

Personal Note #2: Graph Neural Network.

→ Graph Neural Networks work with Graph-like data.

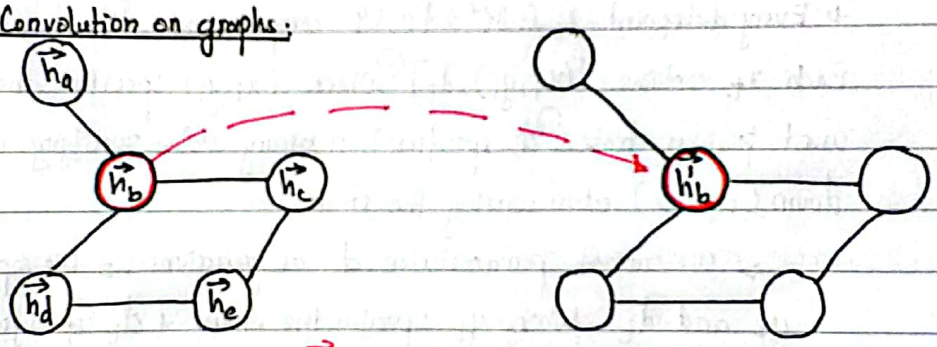
⊛ Graph convolution.

→ Graph can be seen as strict generalisation of images

→ CNN leverages convolutional operations to extract the spatial regularity in images.

→ Would be nice if we could generalize it to operate on arbitrary graphs!

*) Convolution on graphs:



$$\vec{h}'_i = g(\vec{h}_a, \vec{h}_b, \ldots, \vec{h}_e).$$

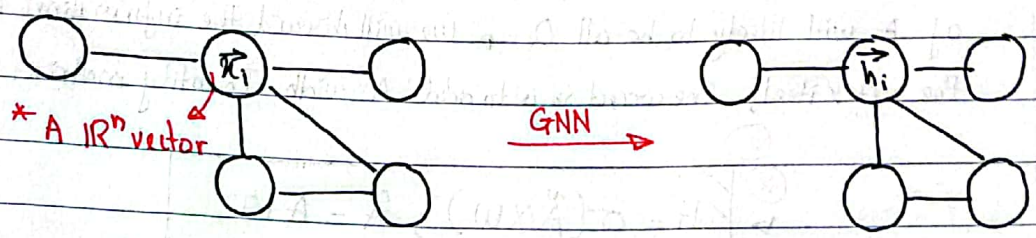
→ Ideally $g(\cdot)$ should be some function of the entire set of neighbors to particular nodes.

→ ~~Challenges with graph convolutions~~. Desired properties for graph conv:
- Computational efficiency ($\sim O(V+E)$)
- Fixed number of params (independent of inputs).
- Localisation (acts on local neighborhood of nodes).
- Weighted neighbors importance.
- Inductivity.

→ However, arbitrary graphs are much harder to deal with.

Thứ
Ngày . . No.

⊛ What to do with GNN outputs.



Inputs (X, A) Latents (H, A)

⊛
1. Node classification: $\vec{Z}_i = f(\vec{h}_i)$ ⊛
2. Graph classification: $\vec{Z}_G = f(\sum_i \vec{h}_i)$
3. Link classification: $\vec{Z}_{ij} = f(\vec{h}_i, \vec{h}_j, \vec{e}_{ij})$

⊛ Simple graph update rule:

↳ Let's assume our graph is unweighted and undirected:

→ $A_{ij} = A_{ji} = \begin{cases} 1, \text{if } x_i \& x_j \text{ are connected.} \\ 0, \text{otherwise.} \end{cases}$

↳ We can aggregate the neighborhood by multiplying with adjacency matrix:

① → $H = \sigma(AXW)$ →
- X: Input nodes.
- A: Adj matrix
- W: Learnable weights.

→ • $X \in \mathbb{R}^{N \times d}$ where $\begin{cases} N = \#\text{ of nodes in the graph.} \\ d = \#\text{ of dimensions in the feature vector/node.} \end{cases}$

• $A \in \mathbb{R}^{N \times N}$ where node's order in A is constitent with node's order in X.

• $W \in \mathbb{R}^{d \times d'}$ where d' = latent's dimensions.

Thứ
Ngày . . No.

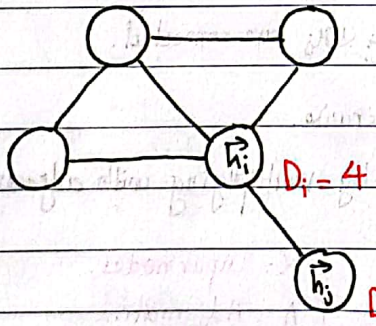
(*) ↳ However, there is one issue in (1), Unless enforced, the diagonal of A will likely to be all 0s → we will discard the information on the vertex itself. One correction is to add A with Identity matrix.

(2) → $H = \sigma(\tilde{A}XW)\ ;\ \tilde{A} = A + I$

↳ The rule can be re-written node-wise as:

(3) $\vec{h_i} = \sigma\left[\sum_{j \in N_i} W\vec{h_j}\right]\ ;\ N_i = i^{th}$ node neighborhood

(*) ↳ Another problem is that the neighborhood size for every node is in consistent → The output scale might differ a lot.



→ a simple remedy is to take the mean output at each node by dividing the output by the neighborhood size.

→ We can normalize the adj matrix with the deg matrix D.

(4) → $H = \sigma(\tilde{D}^{-1}\tilde{A}XW)\ ;\ \tilde{D} = \text{deg mat of } \tilde{A}.$

(5) → $\vec{h_i} = \sigma\left[\frac{1}{|N_i|}\sum_{j \in N_i} W\vec{h_j}\right]$

Thứ
Ngày

No.

⊛ Other aggregation rules:

↳ Symmetric pooling: (4) $H = \sigma\left(\tilde{D}^{-\frac{1}{2}} A \tilde{D}^{-\frac{1}{2}} X W\right)$

→ Node-wise: (7) $\vec{h}_i = \sigma\left[\sum_{j \in N_i} \frac{1}{\sqrt{|N_i||N_j|}} W \vec{h}_j\right]$

→ Simple and powerful → Most cited GNN paper!

⊛ Message Passing Neural Network.

↳ Let $\vec{m}_{ij}$ be the message sent across $i \to j$, computed using a message function:

(8) $\vec{m}_{ij} = f_e(\vec{h}_i, \vec{h}_j, \vec{e}_{ij})$

* Edge features.

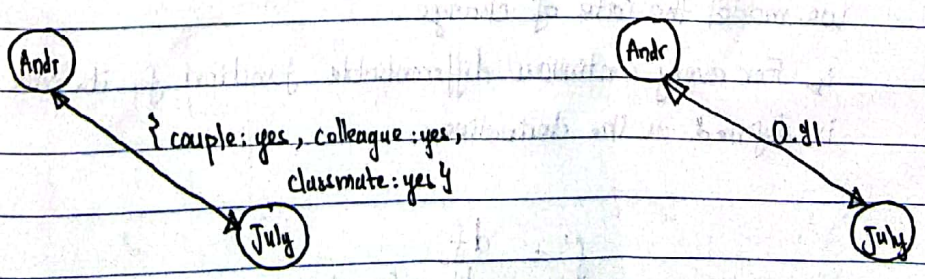
↳ A node's update rule is a function of it's previous state and the aggregation of all the messages. This is called a readout function.

(9) $\vec{h}_i' = f_v\left(\vec{h}_i, \sum_{j \in N_i} \vec{m}_{ij}\right)$

↳ Both $f_e(\cdot)$ and $f_v(\cdot)$ are small MLPs.

*) Examples of edge features:

↳ Edge features are just like node features except it contains information about objects' relationship/connection, not the object it self. E.g.



① Connection with bunch of attributes.

② Connection with weights (E.g. how close 2 people are)