MESSEGE-PASSING : Information is propagated between nodes of a graph along its edges.

A GNN layer can be thought of as a message-passing setup where nodes update their embeddings by aggregating messages from their direct neighbours.

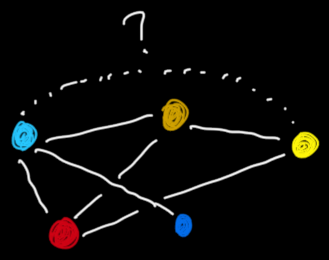
Various GNN models differ in how nodes aggregate the messages from their neighbours.

(I) GCN (Kipf and Welling) Element-wise mean of the embeddings of the neighbours.

(II) GraphSAGE (Hamilton et al) Element-wise Maximum of the embeddings of the nbrs.

(III) Graph Attention Network (Velickovic et al) (GAT)
weighted sum of the embedding of the neighbours where the weights for each neighbour are computed via an attention b/w the current node and the neighbour.

often require to model the interaction b/w two non-connected nodes.



↓

MP-GNNs achieve this by using multiple layers.

⇓

OVER-SQUASHING Message-passing fails to propagate information effectively over the graph.

↓

Growing amount of information is squashed into fixed-sized vectors.

from distant nodes

⇒ MP-GNNs are not adept at capturing long-range dependencies.

FURTHERMORE

OVER-SMOOTHING Embeddings of the nodes become indistinguishable with increasing the depth of the network.

↓

GNN Layers

Receptive field of a node grows exponentially with the radius of the neighborhood.

Traffic Analogy (Due to Dominique Beaini)

Over-squashing → Stuck in traffic

Over-smoothing → Lost in traffic

⇒ Also relates to the expressivity of GNNs. (in terms of WL-test)

GRAPH TRANSFORMERS attempt to alleviate these problems of MP-GNNs.

↓

Instead of passing messages b/w neighbours, the idea is to directly model the dependence b/w any two nodes in the graph.

As opposed to the local attention over neighbours in GAT, graph transformers seek to apply global attention over the whole graph.

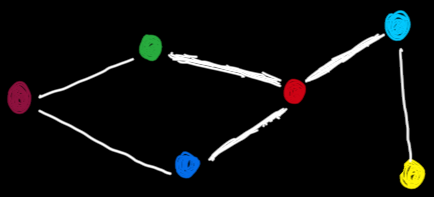
⇒ Both connected and non-connected nodes can contribute to the node embedding.

Problem: Standard transformer with global attention is oblivious to the graph structure.

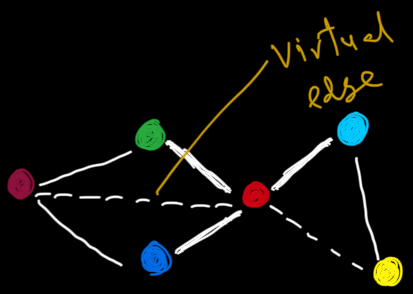
⇓

Challenge is how to incorporate the structural information of the graph.

Different Graph transformer models differ in how they leverage graph structure along with the global attention.



MP-GNN

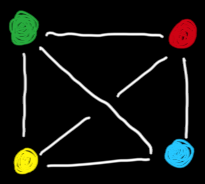


Graph Transformer

NEXT:

- Spectral Attn Network
- GraphiT
- Graphormer
- SignNet and BasisNet
- Learnable structural and positional Encoding.

TRANSFORMERS (NLP) treat a sentence as a fully-connected graph.

This is a sentence $\longleftrightarrow$ 

Each word is connected to every other word.

Positional Encoding is chosen to account for the relative positions of the words within a sentence.

Transformer Input is Word Embedding $\oplus$ Positional Encoding

How to extend the transformers to arbitrary graphs?

↓

use Positional Encoding (PE) for nodes.

Challenging to design Positional Encoding for graphs since there is NO canonical ordering on graphs.

Infact, GNNs are designed to learn node embeddings that are invariant to the node positions.

(Permutation Invariance)

DESIDERATA for PE :

(I) PE should be chosen such that the nearby nodes have similar PEs and the nodes that are far apart have dissimilar PEs.

(II) PE of a node should be unique.

↓

But graph structure imposes symmetries.

E.G. if nodes i and j are structurally symmetric (interchangable) then their PEs are also interchangable.

$i \leftrightarrow j \implies PE(i) \leftrightarrow PE(j)$

⇒ PE is arbitrary upto symmetries of the graph.

↳ NEED to design architectures that can deal with these ambiguities.

Refined (II) The amount of ambiguities in PE should be as little as possible.

Simplest possible choice for PE is to assign an ordering to the nodes. $\forall v \in V$, $PE(v) = Index(v)$

However, there are $N!$ possible orderings. $N = |V|$ Number of Nodes

$\Downarrow$

During training the orderings are uniformally sampled from $N!$ possible choices such that the network is independent of the choice of the ordering.

(Murphy et al 2019)

Proposal : use graph Laplacian eigenvectors as PE.

(Dwivedi et al 2021)

- Provide an embedding of the nodes into the Euclidean space.
- A meaningful coordinate system
- Cognizant of the global graph structure.
- Encode distance aware information
  - Nearby nodes have similar PEs
  - Far-apart nodes have different PEs

⇨ Laplacian eigenvectors can be interpreted as a GENERALIZATION of the PE of the NLP transformer.

Sentence graph Laplacian = Discrete Line

↓ eigenvectors

Sine and Cosine

$$\forall v \in V, \quad PE(v) = (\underbrace{\lambda_1, \lambda_2, \cdots, \lambda_k}) \in \mathbb{R}^k$$

k-smallest non-trivial eigenvectors of the Laplacian.

$$\Delta = I - D^{-1/2} A D^{-1/2} \quad \underline{\text{EIGEN DECOMPOSITION}} \quad U^T \Lambda U$$

Δ: Laplacian; D: Degree matrix; A: Adjacency matrix; Λ: eigenvalues; U: eigenvec.

Q: Are these Positional Encodings unique?

NO

Laplacian eigenvectors are defined upto sign. ($\lambda \pm 1$ Symmetry.)

⇓

Network Needs to deal with this ambiguity.

During training the sign of the eigenvectors are randomly flipped so as to have the model invariant to sign.

[Dwivedi et al 2021]

$\Downarrow$

Possible number of flips 2^k.

- Laplacian PE reduces the sample space of the node-ordering PE.

$\downarrow$

Invariant to node-reparametrization.
But defined upto the sign.

Choose $k << N \implies 2^k << N!$

Laplacian PE has been used in various Graph Transformer architectures.

An added benefit is that Laplacian eigenvectors can be pre-computed.

Laplacian Positional Encoding

PE in the NLP transformer (Vaswani et al) → Sine and Cosine fns

↓

CAN'T be defined for arbitrary graphs since there is no notion of position along an axis. (time-steps for words in a sentence)

Extension of Sinusoidal fns to arbitrary graphs

⇓

Eigenvectors of the graph Laplacian.

- In Euclidean space, Laplacian eigenfunctions are Sine/Cosine. with squared frequencies as eigenvalues.

Dwivedi et al proposed using k Laplacian eigenvectors as PE. (fixed #)

(A graph G with N nodes can have at most N linearly independent eigenvectors)

Now the graphs in a dataset can have varying number of nodes.

⇓

Bottleneck when there is a large disparity in the number of nodes across graphs. i.e. the smallest graphs have significantly fewer nodes than the largest graphs.

⇓

A small fraction of the eigenvectors will be used for large graphs.

⇓

Loss of information!

- This motivates the need for PE of dimension k which DOES NOT depend on the number of eigenvectors in the graph

↓

Spectral Attention Network
(Kreuzer et al NeurIPS 2021)

↓

Utilize full spectrum of the Laplacian spectrum to Learn the position of each node in the graph.

[LEARNABLE POSITIONAL ENCODING]
(LPE)

Idea:
Can use the Laplacian eigenvectors as initialization of the trainable PE.

(1) Calculate Laplacian eigenvectors of the graph.
initial static PE

(11) use a suitable architecture to process the LapPE to "learn" PE for the nodes.
- Can be a Neural Network
- Can be learnt End-to-End with the Graph transformer

OR Can be an independent unsupervised or Self-Supervised Component.

⇨ Kreuzer et al proposed a transformer type architecture to process initial Laplacian PE into learnable PE which are independt of the number of Laplacian eigenvectors.

Transformers are most effective on Sequential data (with PE)

⇓

Meta Question: How to think of Laplacian eigenvectors as Sequences ?

i.e. What is the appropriate PE for Laplacian Eigenvectors.

Fourier transform maps a function from space (or time) domain to spatial (or temporal) frequency domain.

In Euclidean space, for a fn

$$\hat{f}(\xi) = \int_{\mathbb{R}^n} f(x)\, \underbrace{e^{-2\pi i \xi \cdot x}}\, dx$$

↓ Euler's formula

$$\cos(2\pi \xi x) - i \sin(2\pi \xi x)$$

For graphs, the Fourier transform of a fn is defined as

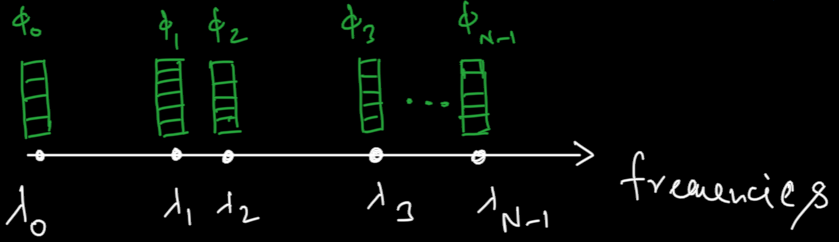
$$F(f)(\lambda_i) = \langle f, \phi_i \rangle$$

↓ eigenvalue of the Laplacian

↓ Eigenvector of the Laplacian corresponding to λ_i.

⇒ Laplacian eigenvectors are best viewed as vectors positioned on the axis of eigenvalues.

(Rather than Columns of a matrix)

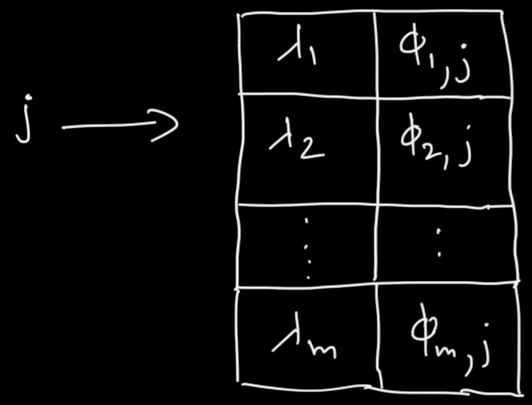


☞ We can think of Laplacian eigenvectors as sequences with the corresponding eigenvalues as time-steps (Non-Uniform)

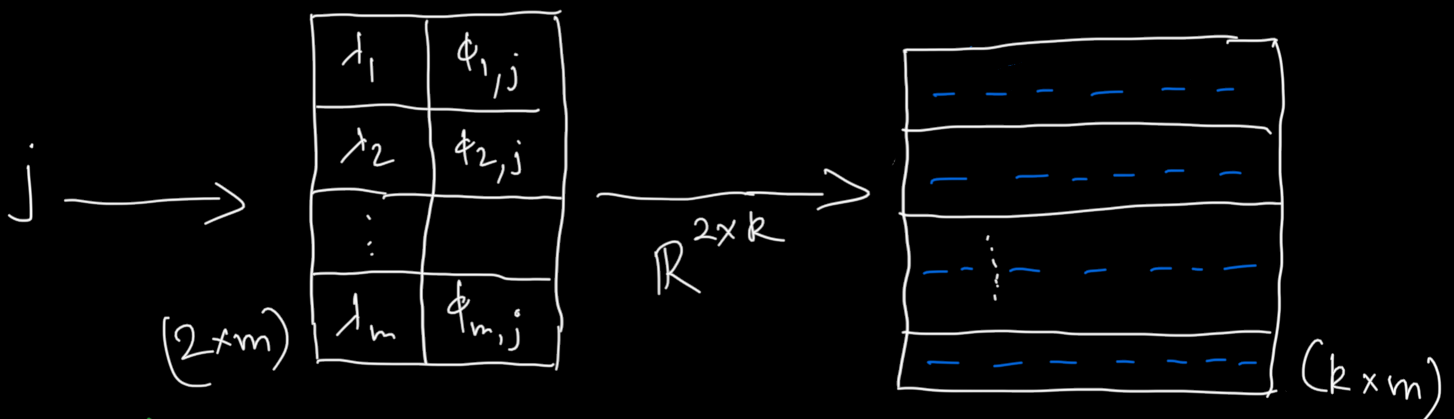
LPE transformer a la Kreuzer et al.

(I) Create Embedding matrix for each node j by concatinating the m-smallest eigenvalues with their corresponding eigenvector. $(2 \times m)$

- m is a hyperparameter similar to length of sentences.
- For graphs where $m >$ #nodes, a padding is added.



(II) A Linear projection layer is applied to map the initial embeddings to k-dimensional Embeddings. (k is a hyperparam)



(III) A transformer encoder then computes Self attention over the sequence of length m and hidden dimension k.

(IV) Sum-pooling along the m-axis produces an embedding of fixed k-dimension.

$\Downarrow$

Learned Positional Embedding of dimension k.

Inspired by Physics

(Pseudo) Inverse of the Laplacian represents electrostatic potential of a charge.

Green's function

Let $\Delta = D^{-1/2} L D^{-1/2}$ be the normalized, Symmetric Laplacian.

$(\hat{\lambda}_i, \hat{\phi}_i)$ and $(\hat{\lambda}_i, \hat{\phi}_i)$

↓ ↓

e'values and e'vectors of Δ.

$$\Delta^{-1} := G(j_1, j_2) = d_{j_1}^{1/2} d_{j_2}^{-1/2} \underset{i>0}{\leq} \frac{(\hat{\phi}_{i,j_1} \hat{\phi}_{i,j_2})^2}{\hat{\lambda}_i}$$

↓

electric potential b/w nodes j_1 & j_2.

$\Rightarrow$ Product of the Laplacian eigenvectors is proportional to the electrostatic interaction between 2 nodes.

RECALL: We can build node positional Encodings (PE) using the eigenvectors of the graph Laplacian.

Let's build Graph Transformers

Let $G = (V, E)$ be a graph.

Nodes, Edges

$\forall$ node $i \in V$, $\alpha_i \in \mathbb{R}^{d_n}$ Node features

$\forall$ edge $(i,j) \in E$, $\beta_{ij} \in \mathbb{R}^{d_e}$ Edge features

$\forall$ node $i \in V$, $\lambda_i \in \mathbb{R}^k$ Node positional Encoding.

The first step in the GT architecture is the linear projections of the node feats, edge feats, and node PEs.

$$\hat{h}_i^0 = A^0 \alpha_i + a^0, \quad A^0 \in \mathbb{R}^{d \times d_n}, \; a^0 \in \mathbb{R}^d$$

$$e_{ij}^0 = B^0 \beta_{ij} + b^0, \quad B^0 \in \mathbb{R}^{d \times d_e}, \; b^0 \in \mathbb{R}^d$$

$$\lambda_i^0 = C^0 \lambda_i + c^0, \quad C^0 \in \mathbb{R}^{d \times k}, \; c^0 \in \mathbb{R}^d$$

Input Node Embedding $h_i^0 = \hat{h}_i^0 + \lambda_i^0$

(Same as NLP transformers)

$(l+1)$-th Layer of the GT (Dwivedi et al)

$\forall$ node i

$$\hat{h}_i^{l+1} = O_h^l \Big\|_{k=1}^{H} \Big(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,l} \, V^{k,l} h_j^l \Big)$$

- $H \rightarrow$ #(Heads)
- $O_h^l \in \mathbb{R}^{d \times d}$
- $\|_{k=1}^{H}$: Concat
- $\sum_{j \in \mathcal{N}_i}$: Sum over Neighbours
- $w_{ij}^{k,l} = \mathrm{Softmax}_j \left(\dfrac{Q^{k,l} h_i^l \cdot K^{k,l} h_j^l}{\sqrt{d_k}} \right)$

$h_i^l \longrightarrow Q^{k,l} h_i^l$ QUERY

$h_j^l \longrightarrow K^{k,l} h_j^l$ KEY

$h_j^l \longrightarrow V^{k,l} h_j^l$ VALUE

$Q^{k,l}, K^{k,l}, V^{k,l} \} \in \mathbb{R}^{d_k \times d}$

$\Rightarrow$ Query (current node) attends to the Values (Neighbours) based on its correlations with the corresponding Key.

This is direct application of the Scaled-dot Product Attn of the transformer to graphs. $\mathrm{Softmax}\left(\dfrac{K Q^T}{\sqrt{d}}\right) V$

Graph Attn Network (GAT) used additive attention (Velickovic et al)

$$w_{ij} = \underset{j}{\text{Softmax}}\left(\text{MLP}(Wh_i \oplus Wh_j)\right)$$

⇒ The attention outputs across heads are concatenated and then linearly projected to obtain the $(l+1)$-th layer node Embedding.

The node embeddings pass through Norm and feed-fwd layers with residual connections before fed to the task-specific Network.

$$\hat{h}_i^{l+1} = \text{Norm}\left(h_i^l + \hat{h}_i^{l+1}\right)$$

(skip-connection)

$$\hat{\hat{h}}_i^{l+1} = W_2^l \, \text{ReLU}\left(W_1^l \hat{h}_i^{l+1}\right)$$

(Projection)

$$h_i^{l+1} = \text{Norm}\left(\hat{h}_i^{l+1} + \hat{\hat{h}}_i^{l+1}\right)$$

$$\text{output} = \text{MLP}\left(h_i^L\right)$$ ← embedding after L-layers.

GT Layer with Edge features

Motivated by the better utilization of rich information present in several datasets in the edge features, Dwivedi et al proposed a modification of the GT layer in the presence of edge features.

$$\hat{w}_{ij}^{k,l} = \left(\frac{Q^{k,l} h_i^l \cdot K^{k,l} h_j^l}{\sqrt{d_k}} \right) \cdot E^{k,l} e_{ij}^l$$

e_{ij}^l → edge features

$E^{k,l} \in \mathbb{R}^{d_k \times d}$

$$w_{ij}^{k,l} = \text{softmax}_j \left(\hat{w}_{ij}^{k,l} \right)$$

$$h_i^{l+1} = O_h^l \Big\|_{k=1}^{H} \left(\sum_{j \in N_i} w_{ij}^{k,l} \, v^{k,l} h_j^l \right)$$

Edge features can also be updated as

$$e_{ij}^{l+1} = O_e^l \Big\|_{k=1}^{H} \left(\hat{w}_{ij}^{k,l} \right)$$

Think of $\hat{w}_{ij}^{k,l} = Q^{k,l} h_i^l \cdot K^{k,l} h_j^l$ as the intermediate attention score quantifying the information about the edge (i,j).

When additional edge info in the form of edge features is available, $\hat{w}_{ij}^{k,l}$ is modified as —

$$\hat{w}_{ij}^{k,l} = \hat{w}_{ij}^{k,l} * E^{k,l} e_{ij}^l$$

Graph Transformer a la Dwivedi et al

- Local attention over the neighbourhood of each node i.e. the attn depends on the neighbourhood connectivity
- The node PE is given by the eigenvectors of the graph Laplacian.
- Architecture can be extended to include edge features.

↓

- Still local attention
- LapPE can result in bottlenecks.

↓

Spectral Attention Network :

- Global full-graph Attention
- Learnable Positional Encodings (LPE)

↓

Full-graph Attn while preserving ~~the~~ local connectivity of the nodes in the graph.

Use two sets of attention mechanisms

- one for nodes connected by real edges
- one for nodes connected by virtual edges (Any pair of nodes)

Similar to the GT architecture, the $(l+1)$-th layer of SAN is defined as

$$\hat{h}_i^{l+1} = O_h^l \Big\Vert_{k=1}^{H} \Big(\sum_{j \in V} \omega_{ij}^{k,l} V^{k,l} h_j^l \Big)$$

↳ over all Nodes

$$\text{where } \omega_{ij}^{k,l} = \begin{cases} \frac{1}{1+\gamma} \operatorname{softmax}\left(\hat{\omega}_{ij}^{k,l}\right) & \text{if } (i,j) \in \mathcal{E} \\ \frac{\gamma}{1+\gamma} \operatorname{softmax}\left(\hat{\omega}_{ij}^{k,l}\right) & \text{if } (i,j) \notin \mathcal{E} \end{cases}$$

$\gamma \in \mathbb{R}^+$ is a hyperparameter that controls the amount of bias towards the full-graph attention.

SAN uses the edge feature enhanced version of the GT attention as follows —

$$\hat{\omega}_{ij}^{k,l} = \begin{cases} \dfrac{Q^{1,k,l} h_i^l \cdot K^{1,k,l} h_j^l \cdot E^{1,k,l} e_{ij}}{\sqrt{d_k}} & \text{if } (i,j) \in \mathcal{E} \\ \dfrac{Q^{2,k,l} h_i^l \cdot K^{2,k,l} h_j^l \cdot E^{2,k,l} e_{ij}}{\sqrt{d_k}} & \text{if } (i,j) \notin \mathcal{E} \end{cases}$$

Similar to GT, we have Norm and Residual Connections.

$$\hat{\hat{h}}_i^{l+1} = \text{Norm}\left(\hat{h}_i^{l+1} + h_i^l\right)$$

$$\hat{\hat{h}}_i^{l+1} = W_2^l \, \text{ReLU}\left(W_1^l \, \hat{\hat{h}}_i^{l+1}\right)$$

$$h^{l+1} = \text{Norm}\left(\hat{\hat{h}}_i^{l+1} + \hat{\hat{h}}_i^{l+1}\right)$$

Leveraging eigenvectors of the graph Laplacian for building node positional encodings (PE) has shown promise to craft expressive graph Transformer architectures.

(ref Dwivedi et al Graph Transformer
Kreuzer et al Spectral Attn Network)

⇓

Limitation is that the Laplacian eigenvectors are defined upto the sign.

i.e. if v is an eigenvector, then so is $-v$

↓

Need to have Sign-invariant PE.

Prior works dealt with the sign ambiguity by randomly flipping the sign of the eigenvectors during training.

⇓

SignNet a new neural architecture that is invariant to sign flips.

(Lim et al)

More generally, for any basis Symmetry i.e. infinitely many choices for basis.

↓

BasisNet – Invariant to basis Symmetry.

Idea: Process each of the eigenspaces individually, and then aggregate. → Spanned by the diff. choices of eigenvectors.

For Sign Symmetry, if v is an eigenvector, then so is $-v$

↓

Process v and $-v$ separately & then aggregate

Neural Networks are employed for processing eigenvectors & then for aggregation.

Let $\{v_1, v_2, \cdots, v_k\}$ be the Laplacian eigenvectors.

$\{-v_1, -v_2, \cdots, -v_k\}$ are also the Laplacian eigenvectors.

SignNet $\quad \phi(\{v_1, v_2, \cdots, v_k\}) \;,\; \phi(\{-v_1, -v_2, \cdots, -v_k\})$

$\{\phi(v_1), \phi(v_2), \cdots, \phi(v_k)\} \quad \{\phi(-v_1), \phi(-v_2), \cdots, \phi(-v_k)\}$

$$\Rightarrow f(v_1, v_2, \cdots, v_k) = \rho\left((\phi(v_1) + \phi(-v_1)), (\phi(v_2) + \phi(-v_2)), \cdots, (\phi(v_k) - \phi(-v_k))\right)$$

$\rho, \phi \longrightarrow$ Neural Networks e.g. MLP, Transformers, DeepSets, GNN etc.

Justification for this design (in 1d)

⇨ A continuous fn $h: \mathbb{R}^n \to \mathbb{R}^s$ is sign-invariant if and only if

$$h(x) = \phi(x) + \phi(-x)$$

for some continuous fn $\phi: \mathbb{R}^n \to \mathbb{R}^s$.

Further, h is permutation-equivariant if & only if $\exists\ \phi$ as above and ϕ is permutation equivariant.

$$P f(x) = f(Px)$$

$P \in \mathbb{R}^{n \times n}$ permutation matrix

⇨ Without details, BasisNet can be defined as —

$$f(v_1, v_2, \cdots, v_k) = \beta\left(\left[\phi_{d_i}(v_i v_i^T) \right]_{i=1,\cdots,k} \right)$$

ϕ_{d_i} → Order 2 invariant GNN

β → MLP, DeepSets, Transformer.

Graphormer

Ying et al (NeurIPS 2021) address the question of efficacy of the Transformer architecture in graph representation learning.

Recall, the key to adaptability of the transformer architecture to graphs is proper incorporation of structural information of graphs.

Self-attention operates over a pair of entities (nodes) and encodes the semantic similarity b/w them.

↓

Agnostic to the relational information.

Two Key Ideas :

(1) Different nodes may have different importance e.g. celebrities tend to have more influence.

||

Encode node centrality explicitly into the model

↓

Add learnable vector encoding to each node which is dependent on the degree of the node.

for each node $v_i \in V$,

$$h_i^{(0)} = x_i + z^-_{deg^-(v_i)} + z^+_{deg^+(v_i)}$$

where $z^-, z^+ \in \mathbb{R}^d$ are learnable embedding vectors specified by the indegree (deg^-) and outdegree (deg^+) of the node.

(11) Different node pairs have different structural properties owing to the relational info.

⇓

Encode relational properties of node-pairs into the model.

↓

Add learnable encoding for each node-pair which is dependent on the relational structure

Spatial Encoding $\phi: V \times V \rightarrow \mathbb{R}$

$\phi(v_i, v_j)$ defines the connectivity b/w the nodes i and j.

$$\phi(v_i, v_j) := \begin{cases} \text{Shortest Path Distance b/w } v_i \text{ and } v_j & \text{if connected} \\ -1, & \text{otherwise.} \end{cases}$$

(III) Various graphs come with edge features which carry important structural information (e.g. in molecular graphs, atom pairs may have features describing properties of the chemical bonds b/w them).

Encode edge features into the model.

↓

Add learnable edge embeddings which depend on the edge features as well as the graph structure.

$\forall$ pair of nodes (v_i, v_j)

Let $SP_{ij} = (e_1, e_2, \cdots, e_N)$ be the shortest path b/w them.

and w_n^E be the learnable vector encoding of the edge e_n.

then $$c_{ij} = \frac{1}{N} \sum_{n=1}^{N} x_{e_n} (w_n^E)^T$$

- $\frac{1}{N}$: Average along the shortest path.
- x_{e_n}: original edge feature
- $(w_n^E)^T$: Dot-product of the given and learnable encoding.

Finally, the Self-attention of the Graphormer is defined as -

$$A_{ij} = \underbrace{\frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}}}_{\text{Self-attn of the Standard transformer}} + \underbrace{b(\phi(v_i, v_j))}_{\text{bias-term (fn of } \phi(v_i, v_j))} + \underbrace{c_{ij}}_{\text{edge feats bias-term}}$$