Graph Neural Networks

Sahar Almahfouz Nasser

Department of Electrical Engineering

Indian Insitute of Technology Bombay

The Tutorial's Agenda:

- Why graphs?
- Types of tasks
- Components of a graph
- Embedding
- Batching
- Most popular GNNs
- GNN vs. CNN
- GNN vs. Transformer
- Graph machine learning tools

Why Graphs?

With graphs we can describe and analyse the relations/interactions between entities

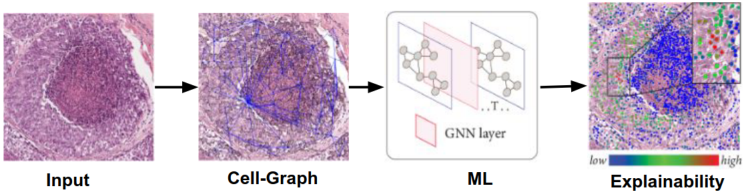
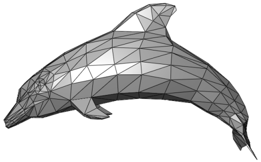


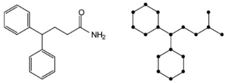
Figure 1: Overview of GNNs functionalities [JPA+21]

Many Types of Data Are Graphs

New frontiers beyond classic neural networks that only learn on images and sequences.


(a) 3D Shapes

(b) Social Networks



(c) Molecules

Discovering Connections Between entities

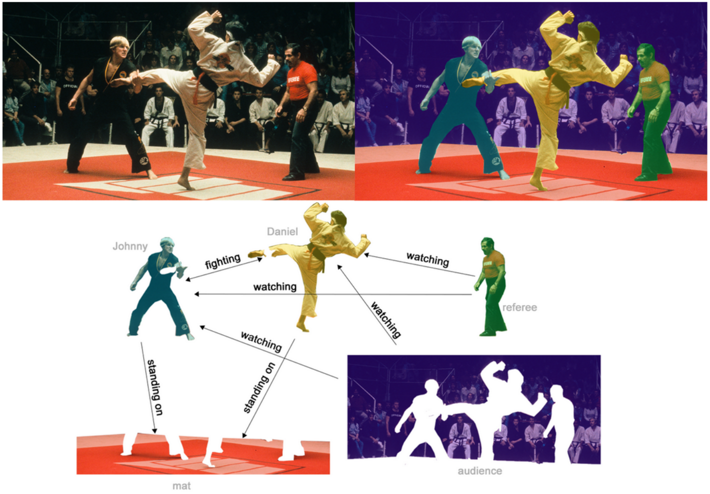


Figure 3: Edge-level inference is in image scene understanding

The Hottest Subfield In ML

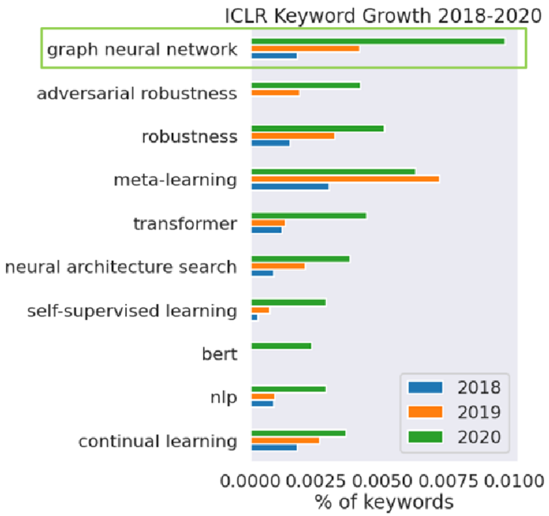


Figure 4: From CS224W

ML Lifecycle

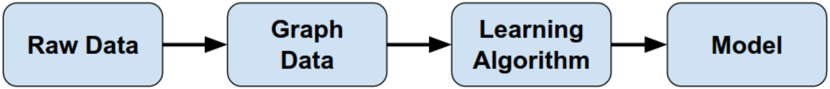


Figure 5: ML lifecycle

Different Types of Tasks

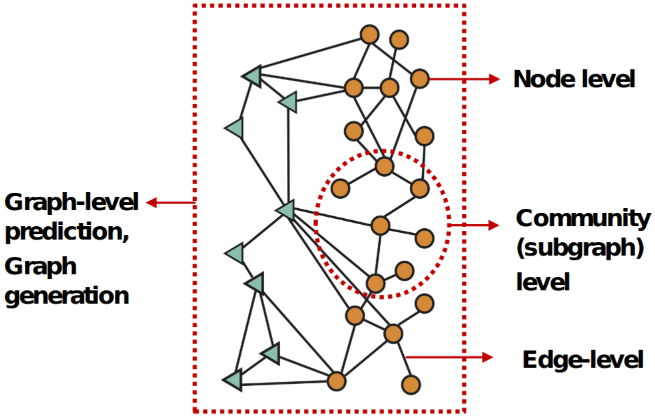

Node level
Graph-level prediction, Graph generation
Community (subgraph) level
Edge-level

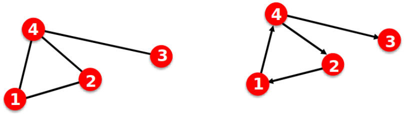
Figure 6: Different Types of Tasks

Different Types of Tasks

- Node classification: Predict a property of a node. Example: Categorize online users / items
- Link prediction: Social Networks
- Graph classification: Categorize different graphs. Example: Molecule property prediction
- Clustering: Detect if nodes form a community. Example: Social circle detection
- Other tasks:
  - Graph generation: Drug discovery
  - Graph evolution: Physical simulation

Components of a Network

- ***Objects*** : Nodes or Vertices $\boldsymbol{N}$
- ***Interactions*** : Links or Edges $\boldsymbol{E}$
- ***System*** : Network or Graph $\boldsymbol{G(N, E)}$



$\boldsymbol{A_{ij} = 1}$ if there is a link from node ***i*** to node ***j***

$\boldsymbol{A_{ij} = 0}$ otherwise

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Figure 7: Adjacency Matrix

Node Embedding

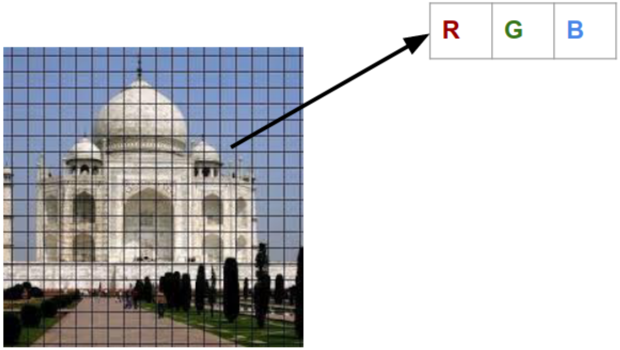


Figure 8: Node Embedding Pixel Embedding

Node Embedding

- Shallow encoder: embedding lookup
- Deep encoders (GNNs)

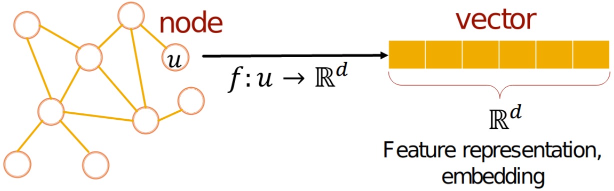


Figure 9: Node Embedding

GNN Layer

- GNNs adopt a "graph-in, graph-out"
- Progressively transform the embeddings (node, edge, global-context), without changing the connectivity of the input graph

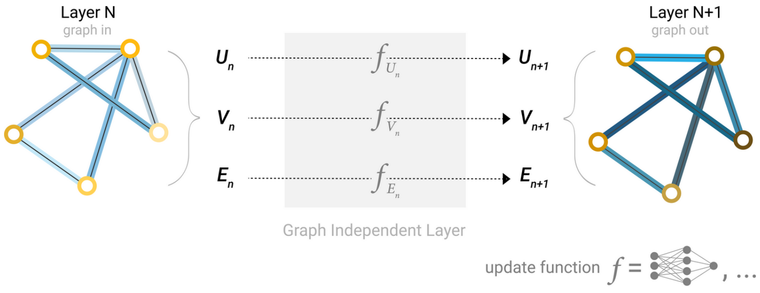


Figure 10: GNN layer

GNN Predictions by Pooling Information

- Binary predictions on nodes
- For each node embedding, apply a linear classifier

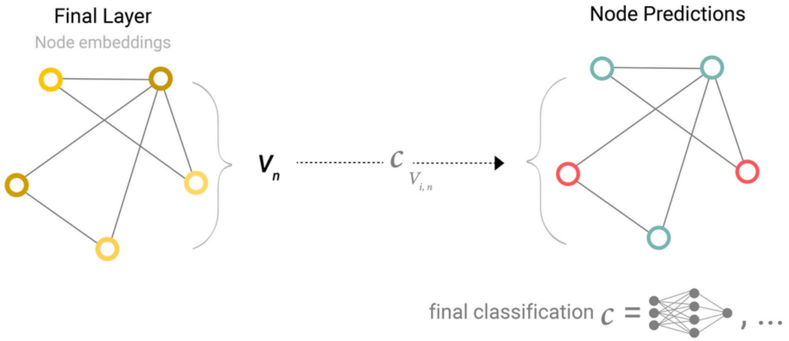


Figure 11: Node Classification

GNN Predictions by Pooling Information

- When information in the graph stored in edges, but no information in nodes, but still need to make predictions on nodes
- Pooling proceeds in two steps:
  - For each item to be pooled, gather each of their embeddings and concatenate them into a matrix
  - The gathered embeddings are then aggregated, usually via a sum operation

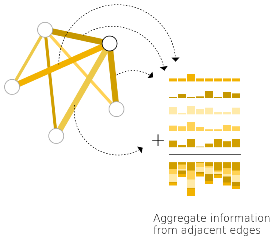


Figure 12: Pooling Information

Global Prediction

- Task: predict a binary global property
- Similar to Global Average Pooling layers in CNNs

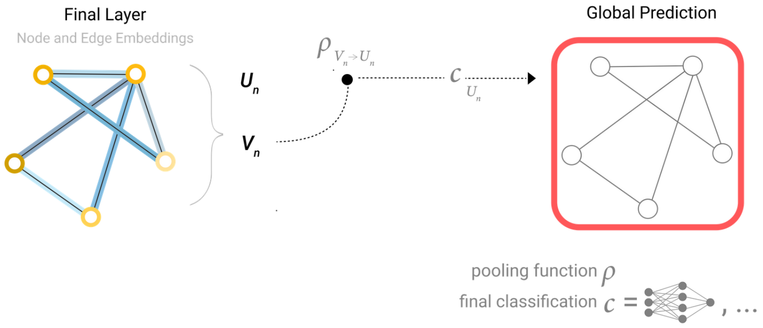


Figure 13: Global Prediction

Classification Task

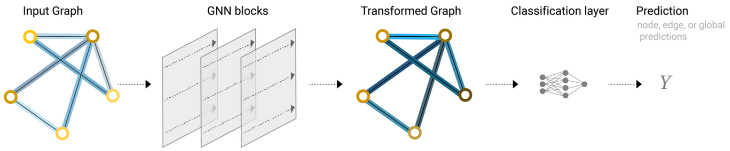


Figure 14: An end-to-end prediction task with a GNN model

Message Passing

- For each node in the graph, gather all the neighboring node embeddings
- Aggregate all messages via an aggregate function (like sum)
- All pooled messages are passed through an update function, usually a learned neural network
- message passing can occur between either nodes or edges

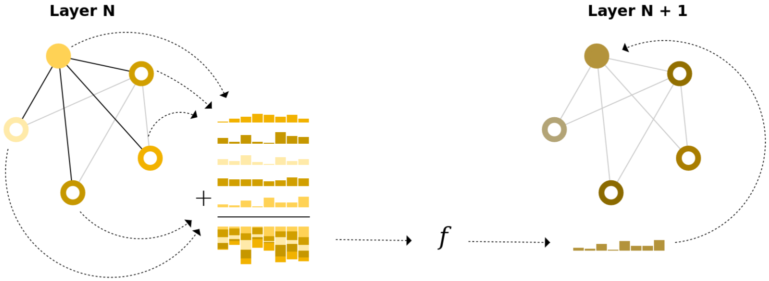


Figure 15: Message Passing

Aggregation Functions

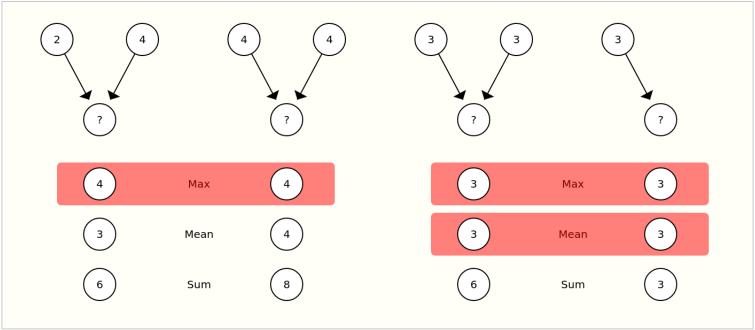


Figure 16: Aggregation Functions

Combining Edge and Node Information

- For each node in the graph, gather all the neighboring
- The node and edge information stored in a graph are not necessarily the same size or shape
- One way to combine the information is by a linear mapping from the space of edges to the space of nodes, and vice versa

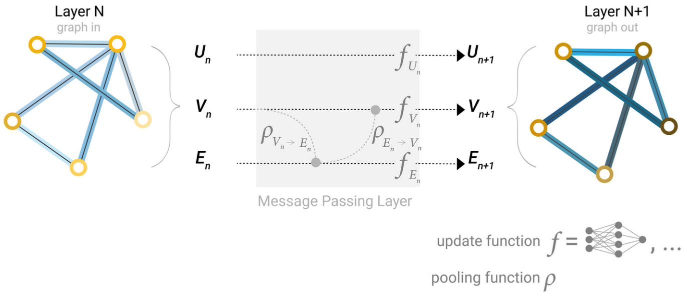


Figure 17: Combining Information

GNN playground

See GNN playground at Distill.pub

Some empirical GNN design lessons

- A higher number of parameters does correlate with higher performance
- models with higher dimensionality tend to have better mean and lower bound performance
- The mean performance tends to increase with the number of layers
- The lower bound for performance decreases with larger number of layers
- The sum aggregation function has a very slight improvement on the mean performance compared to min and max functions

Sampling Graphs and Batching in GNNs

- Due to the variability in the number of nodes and edges adjacent to each other, we cannot have a constant batch size
- Idea: create subgraphs that preserve essential properties of the larger graph(e.g. citation networks)

Most popular GNNs (Deep Graph Encoders)

- Graph Convolutional Networks (GCN)
- Graph Attention Networks (GAT)
- Graph Sample and Aggregate (GraphSAGE)
- Graph Isomorphism Network (GIN)

Graph Convolutional Networks

Classical Convolution: O(n) by parallelization

$$h^{l+1} = w^l * h^l$$
$$h_i^{l+1} = w^l * h_i^l$$
$$h_i^{l+1} = \sum_{j \in N_i} < [w_j^l], [h_{ij}^l] >$$

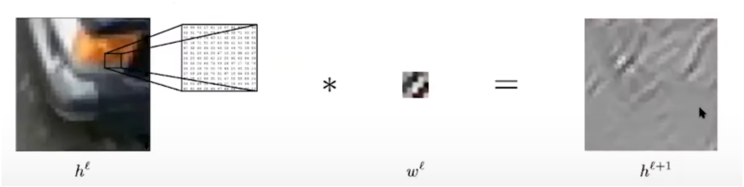
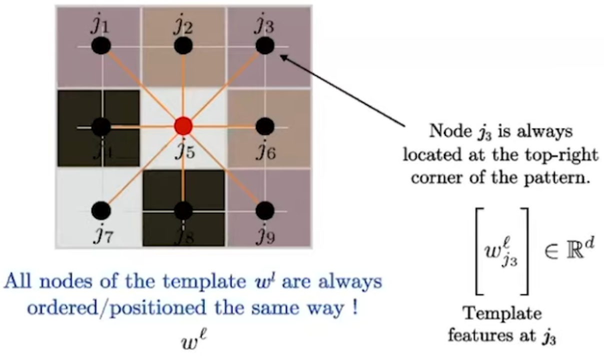


Figure 18: Classical Convolution

Graph Convolutional Network



j_1
j_2
j_3
j_4
j_5
j_6
j_7
j_8
j_9
Node j_3 is always located at the top-right corner of the pattern.
$\begin{bmatrix} w^{\ell}_{j_3} \end{bmatrix} \in \mathbb{R}^d$
Template features at j_3
All nodes of the template w^l are always ordered/positioned the same way !
w^{ℓ}

Figure 19: Classical Convolution

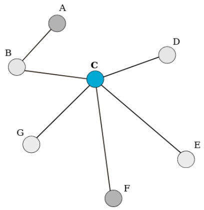
Polynomial Filters on Graphs

The Graph Laplacian

$$D_v = \sum_u A_{vu}$$

D: degree matrix, A: adjacency matrix.

The graph Laplacian $L = D - A$ is the square $n \times n$ matrix



Input Graph G

$$\begin{array}{c|ccccccc} & A & B & \mathbf{C} & D & E & F & G \\ \hline A & 1 & -1 & & & & & \\ B & -1 & 2 & -1 & & & & \\ \mathbf{C} & & -1 & 5 & -1 & -1 & -1 & -1 \\ D & & & -1 & 1 & & & \\ E & & & -1 & & 1 & & \\ F & & & -1 & & & 1 & \\ G & & & -1 & & & & 1 \end{array}$$

Laplacian L of G

Figure 20: The Laplacian of the undirected graph G

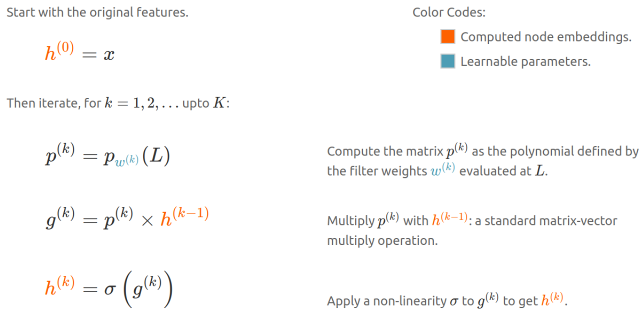
Polynomial Filters on Graphs

$$p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + \ldots + w_d L^d = \sum_{i=0}^{d} w_i L^i$$

where $w = [w_0, w_1, ..., w_d]$ and p$_w(L) isan($n$\times n$ matrix)
These polynomials can be thought of as the equivalent of ‘filters’ in CNNs
See an example Distill

Polynomial Filters on Graphs

Let’s consider a GCN which consists of K different polynomial filter layers



Start with the original features.

$$h^{(0)} = x$$

Color Codes:

🟧 Computed node embeddings.

🟦 Learnable parameters.

Then iterate, for $k = 1, 2, \ldots$ upto K:

$$p^{(k)} = p_{w^{(k)}}(L)$$

Compute the matrix $p^{(k)}$ as the polynomial defined by the filter weights $w^{(k)}$ evaluated at L.

$$g^{(k)} = p^{(k)} \times h^{(k-1)}$$

Multiply $p^{(k)}$ with $h^{(k-1)}$: a standard matrix-vector multiply operation.

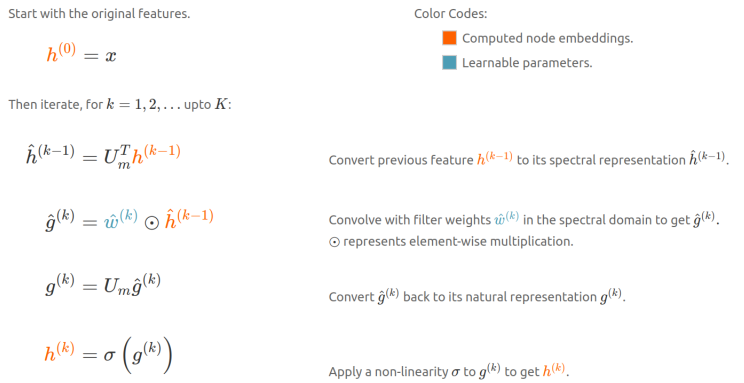
$$h^{(k)} = \sigma\left(g^{(k)}\right)$$

Apply a non-linearity σ to $g^{(k)}$ to get $h^{(k)}$.

Figure 21: Embedding Computation

These networks reuse the same filter weights across different nodes

Spectral Convolutions



Start with the original features.

$$h^{(0)} = x$$

Color Codes:

- 🟧 Computed node embeddings.
- 🟦 Learnable parameters.

Then iterate, for $k = 1, 2, \ldots$ upto K:

$$\hat{h}^{(k-1)} = U_m^T h^{(k-1)}$$

Convert previous feature $h^{(k-1)}$ to its spectral representation $\hat{h}^{(k-1)}$.

$$\hat{g}^{(k)} = \hat{w}^{(k)} \odot \hat{h}^{(k-1)}$$

Convolve with filter weights $\hat{w}^{(k)}$ in the spectral domain to get $\hat{g}^{(k)}$.
$\odot$ represents element-wise multiplication.

$$g^{(k)} = U_m \hat{g}^{(k)}$$

Convert $\hat{g}^{(k)}$ back to its natural representation $g^{(k)}$.

$$h^{(k)} = \sigma\left(g^{(k)}\right)$$

Apply a non-linearity σ to $g^{(k)}$ to get $h^{(k)}$.

Figure 22: Spectral Embedding Computation

Graph Attention Networks

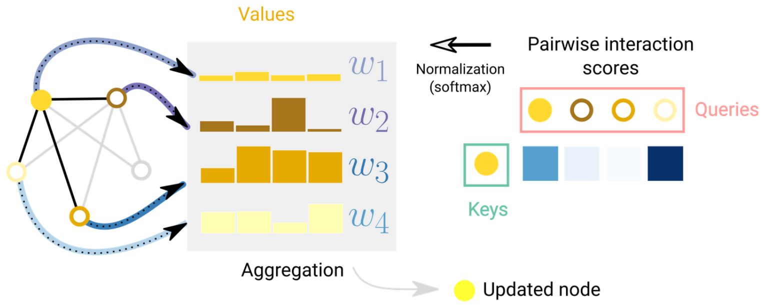


Figure 23: Graph Attention Networks

Check out the example at Graph Attention Networks

Graph Sample and Aggregat: GraphSAGE

1. K-layer GNNs generate embedding of a node using K-hop neighborhood structure and features
2. To compute the embedding of a single node all we need is the k-hop neighborhood

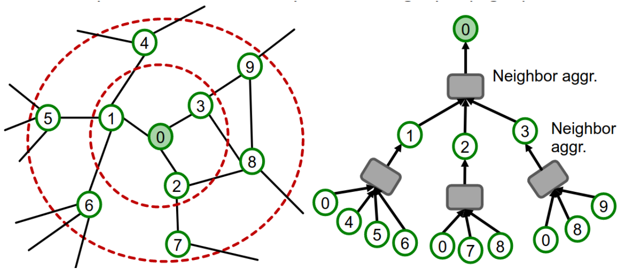


Figure 24: GraphSage

Graph Isomorphism Network

1. Apply an MLP, element-wise sum, followed by another MLP

$$MLP_{\Phi}(\sum_{x \in S} MLP_f(x))$$

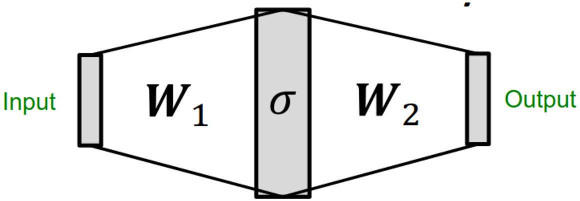


Figure 25: MLP

Graph Isomorphism Network

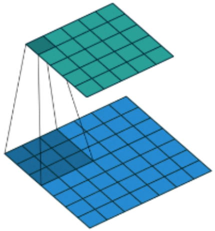
$$MLP_{\Phi}((1-\epsilon)MLP_f(c^k(v))) + \sum_{u \in N(v)} MLP_f(c^k(u)))$$

where ϵ is a learnable scalar

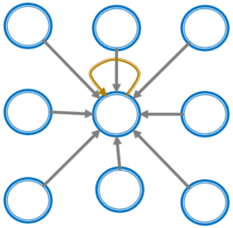
GNN Vs. CNN

CNN can be seen as a special GNN with fixed neighbor size and ordering:

- The size of the filter is pre-defined for a CNN
- The advantage of GNN is it processes arbitrary graphs with different degrees for each node



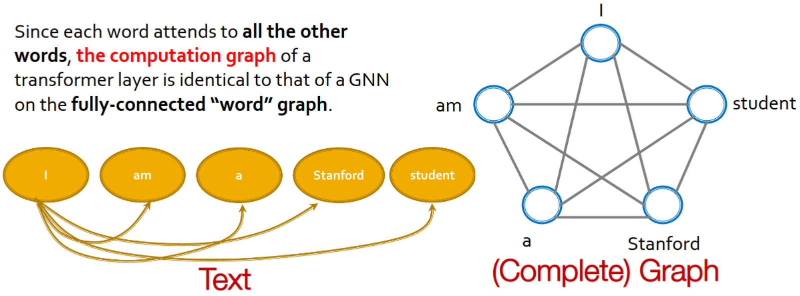
Image


Graph

GNN Vs. Transformer

Transformer layer can be seen as a special GNN that runs on a fully- connected “word” graph!

Since each word attends to **all the other words**, the computation graph of a transformer layer is identical to that of a GNN on the **fully-connected “word” graph**.



Graph Machine Learning Tools

- PyG (Pytorch Geometric) is the ultimate library for GNNs
- The Open Graph Benchmark (OGB) is a collection of realistic, large-scale, and diverse benchmark datasets for machine learning on graphs
- (Histocartography) is a python-based library designed to facilitate the development of graph-based computational pathology pipelines
- SNAP for python (snap.py) is a general purpose, high performance system for analysis and manipulation of large networks
- (NetworkX) is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks

References I

Guillaume Jaume, Pushpak Pati, Valentin Anklin, Antonio Foncubierta, and Maria Gabrani, *Histocartography: A toolkit for graph analytics in digital pathology*, MICCAI Workshop on Computational Pathology, PMLR, 2021, pp. 117–128.

Thank you