Carnegie Mellon University

Graph Neural Networks

Learning on Graph-Structured Data

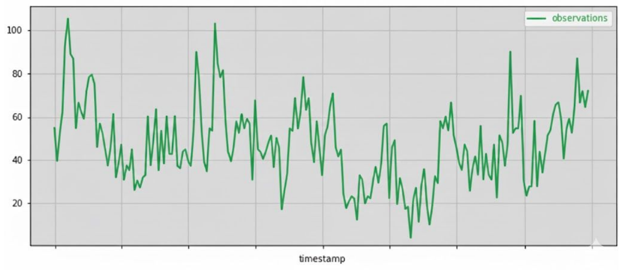
11-485/685/785: Introduction to Deep Learning
Fall 2025

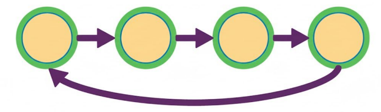
Rutvik Joshi and Ahmed Alhassan

26th November, 2025

We are familiar with Grid-Structured Data

Sequential Data
Recurrent Neural Network

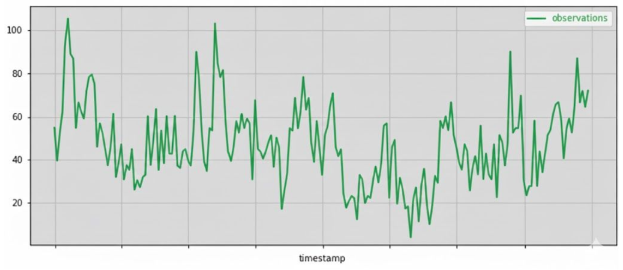


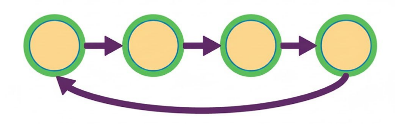


Carnegie Mellon University

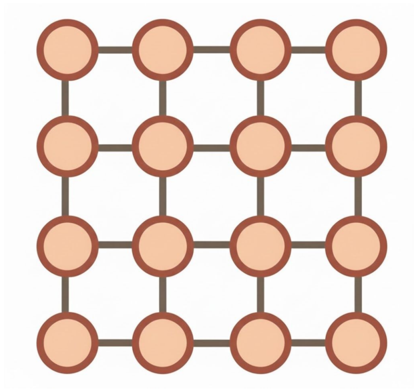
We are familiar with Grid-Structured Data(Euclidean Data)

Sequential Data
Recurrent Neural Network



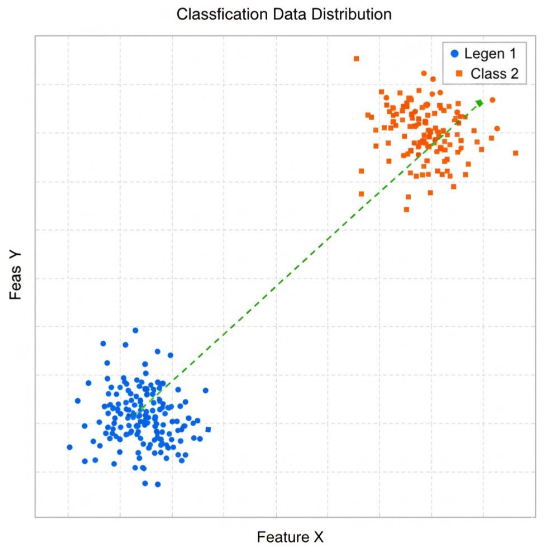


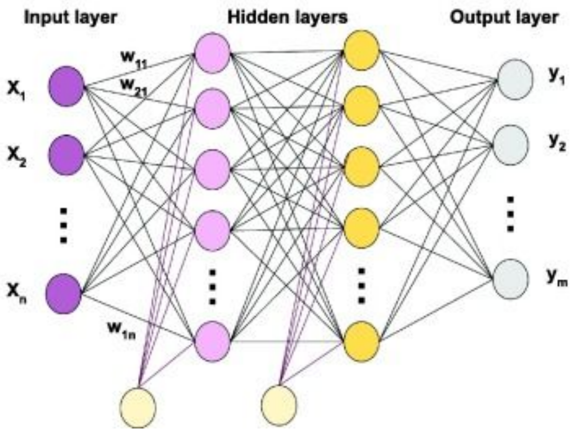
Images
Convolutional Neural Network



We are familiar with Grid-Structured Data(Euclidean Data)

Tabular Data
MLP

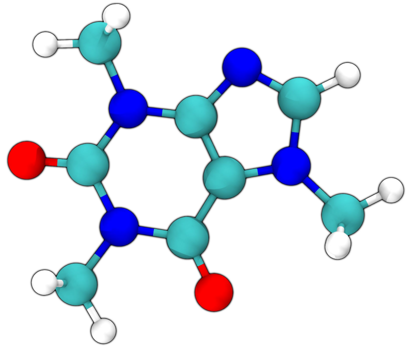




Credits: ScienceDirect Topics

How about

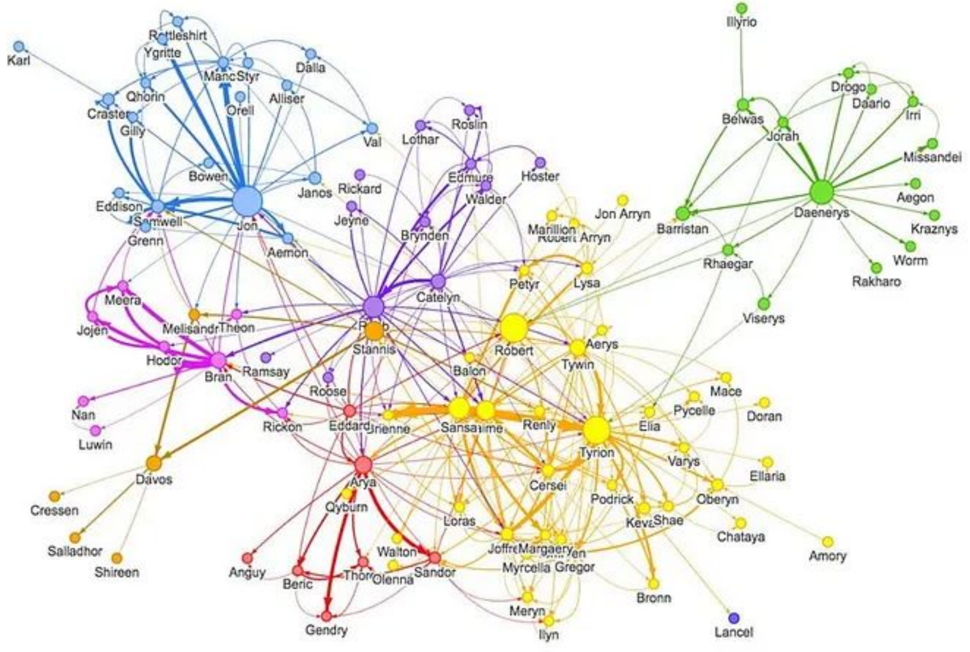
3d representation of the Caffeine molecule



Credits: https://distill.pub/2021/gnn-intro/

How about

Social Network



Credits:medium.com/dair-ai

Why Graphs Matter

Social networks: Relationships, influence propagation

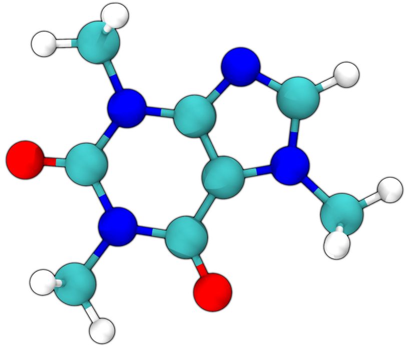
Molecules: Chemical properties depend on structure

Citation networks: Knowledge graphs, recommendations

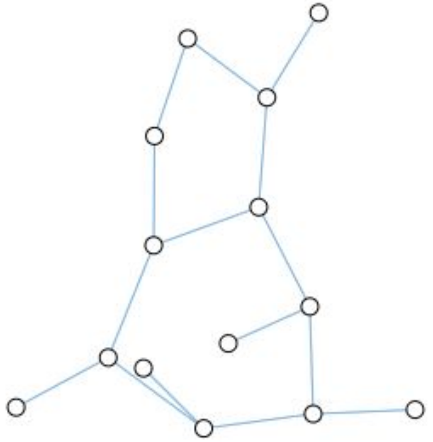
Traffic systems: Routing, congestion prediction

Why Graphs Matter

3d representation of the Caffeine molecule



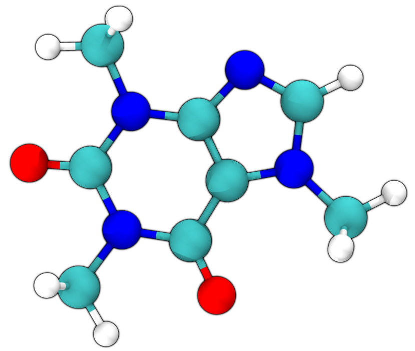
Graph representation of the Caffeine molecule



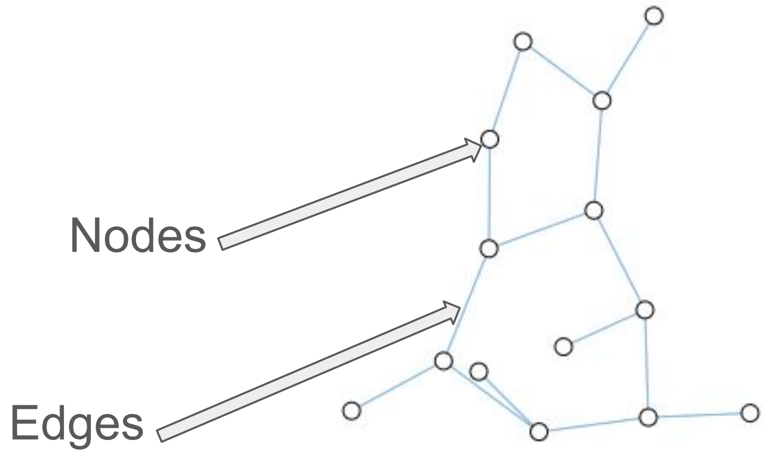
Credits: https://distill.pub/2021/gnn-intro/

Why Graphs Matter

3d representation of the Caffeine molecule



Graph representation of the Caffeine molecule



Credits: https://distill.pub/2021/gnn-intro/

Carnegie Mellon University

Why should we Deep Learning people care?

Why should we Deep Learning people care?

Real-World Applications

- **Drug Discovery:** Predict molecular properties, design better compounds
- **Recommendation:** User-item interactions, collaborative filtering
- **Forecasting:** Traffic, weather, energy consumption on grids

Challenges in handling Graph Data

Models like MLPs, CNNs, and RNNs are built on the assumption that data is structured in a regular, fixed grid (like pixels in an image or steps in a sequence).

Problem:

1. Graphs are irregular and unstructured. Nodes can have a variable number of neighbors, and the connections don't follow a simple, predetermined pattern.

Challenges in handling Graph Data

Models like MLPs, CNNs, and RNNs are built on the assumption that data is structured in a regular, fixed grid (like pixels in an image or steps in a sequence).

Problem:

1. Graphs are irregular and unstructured. Nodes can have a variable number of neighbors, and the connections don't follow a simple, predetermined pattern.

2. Graphs lack a natural order. Models must be permutation-invariant because re-labeling nodes cannot change the result.

Challenges in handling Graph Data

Models like MLPs, CNNs, and RNNs are built on the assumption that data is structured in a regular, fixed grid (like pixels in an image or steps in a sequence).

Problem:

1. Graphs are irregular and unstructured. Nodes can have a variable number of neighbors, and the connections don't follow a simple, predetermined pattern.

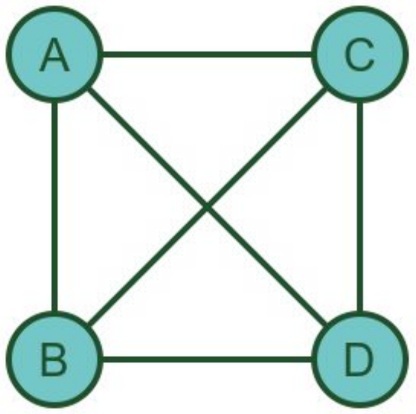
2. Graphs lack a natural order. Models must be permutation-invariant because re-labeling nodes cannot change the result.

3. Standard models cannot easily incorporate the connectivity information. They need a mechanism to aggregate features from neighbors

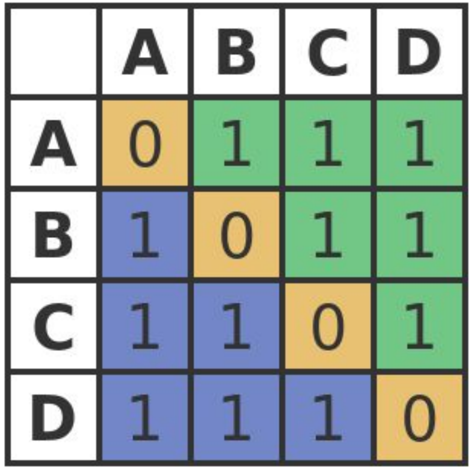
The entire research area dedicated to developing deep learning methods for Non-Euclidean data like graphs, manifolds, and more is called **Geometric Deep Learning**.

Graph Basics

Graph is an ordered pair G = (V, E):
Nodes V (set of vertices)
Edges E (associated with two distinct vertices) connecting them



Complete graph



	A	B	C	D
A	0	1	1	1
B	1	0	1	1
C	1	1	0	1
D	1	1	1	0

Credits: GraphicMaths

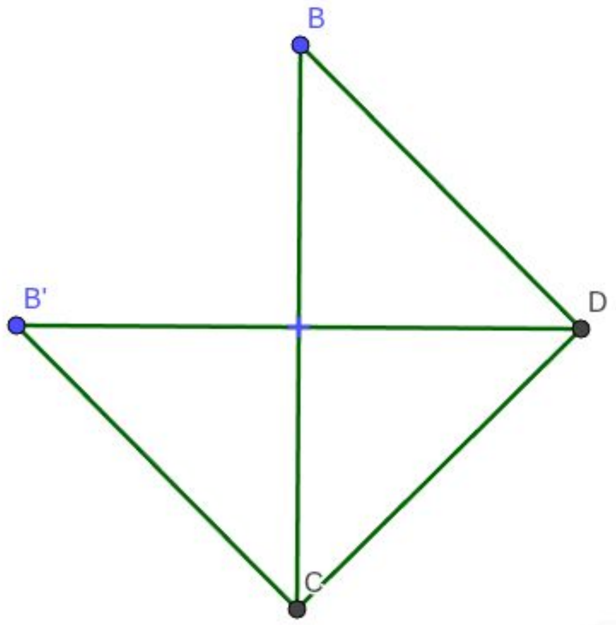
Adjacency Matrix

It is a n × n matrix in which:

- n is the number of vertices
- A(i, j) = 1 only if there is a link from i to j and
- A(i, j) = 0 if not

Adjacency Matrix:

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$



You can create your own Adjacency Matrix on Geogebra

Other common representation

- **Edge List**

  An edge list is a collection of pairs (or tuples), where each pair (u, v) indicates that there is an edge connecting node u to node v

- **Adjacency List**

  It is implemented as an array or map where the index/key represents a vertex, and the value is a list of all vertices adjacent to it (its neighbors).

Embeddings

Transform complex, high-dimensional data (like words or images) into dense, low-dimensional numerical vectors that machine learning models can easily process

- Word Embeddings: Convert words into vectors such that words with similar semantic meaning or that appear in similar contexts are positioned close together in the vector space
  Example: GloVe, Word2Vec, BERT

- Image Embeddings: Compress the high-dimensional pixel data into a dense vector that encodes the image's visual features (objects, patterns, textures)
  Example: CNN, Vision Transformer

These work well for unstructured or sequential data. Fail to capture the defining characteristic of graph data: interconnectedness and complex relationships.

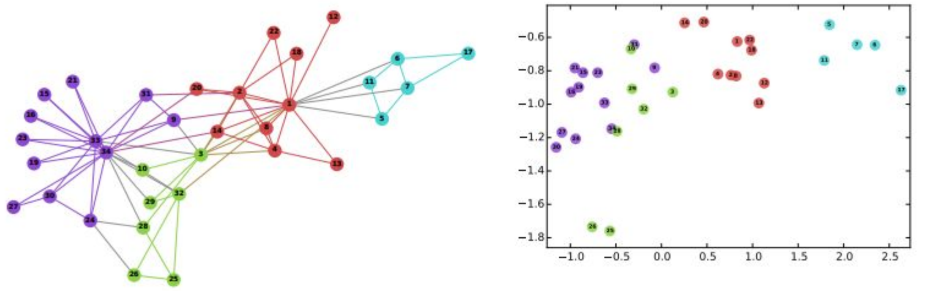
Graph Node Embeddings

To encode the topology and structural properties of the network. This includes:

- node's own features
- Its local neighborhood (dense group of nodes within a larger graph structure)
- Its structural role

Adjacency Matrix are often huge and sparse

Graph embeddings condense this high-dimensional, sparse data into a low-dimensional, dense vector space, such that similar nodes in the graph are embedded close together.



(a) Input: Karate Graph (b) Output: Representation

Image credit: DeepWalk: Online Learning of Social Representations

Three Core Graph Learning Tasks

- **Node Classification**
  Predict label for each node (e.g., social influence detection)
- **Edge Prediction**
  Predict if edges exist or their properties (e.g., friend suggestions)
- **Graph-Level Tasks**
  Classify or predict properties of entire graphs (e.g., molecular properties)

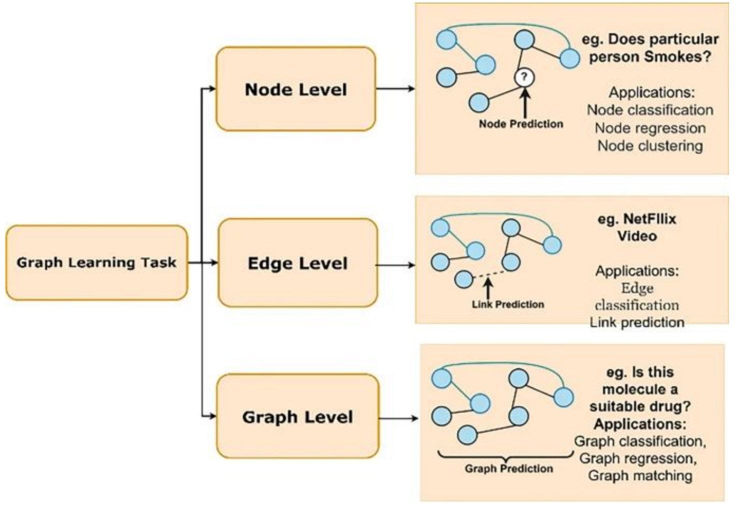

Graph Learning Task
Node Level
Edge Level
Graph Level
eg. Does particular person Smokes?
Node Prediction
Applications: Node classification Node regression Node clustering
eg. NetFlix Video
Link Prediction
Applications: Edge classification Link prediction
eg. Is this molecule a suitable drug?
Graph Prediction
Applications: Graph classification, Graph regression, Graph matching

Image credit: Khemani et al. A review of graph neural networks

Graph Embeddings & MLPs??

This is a two-step process

- First, the embedding is generated
- Second, the MLP is trained on the fixed embedding
- cannot flow back to fine-tune the embedding generation process

Graph Embeddings & MLPs??

This is a two-step process

- First, the embedding is generated
- Second, the MLP is trained on the fixed embedding
- cannot flow back to fine-tune the embedding generation process

Other Problems:

- Re-train the entire embedding space from scratch every time you add a new node or a new edge to the graph
- They ignore or struggle to incorporate the rich node features

Solution?

Graph Embeddings & MLPs??

This is a two-step process

- First, the embedding is generated
- Second, the MLP is trained on the fixed embedding
- cannot flow back to fine-tune the embedding generation process

Other Problems:

- Re-train the entire embedding space from scratch every time you add a new node or a new edge to the graph
- They ignore or struggle to incorporate the rich node features

Solution?

Graph Neural Networks (GNNs)

Graph Neural Networks

Graph Neural Networks (GNNs)

Designed to handle irregular connectivity and the lack of inherent order for nodes.

The core mechanism **Message Passing**

- **Gather**
- **Aggregate**
- **Update**

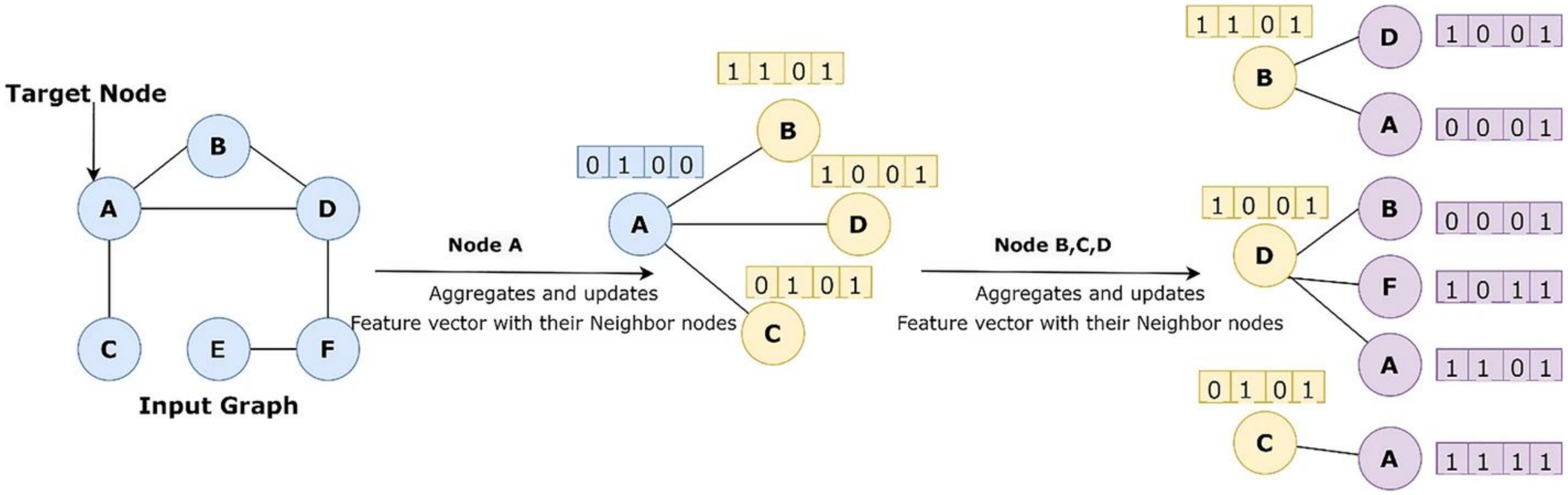


Image credit: Khemani et al. A review of graph neural networks

Graph Neural Networks (GNNs)

The Neighbourhood N_i of a node i is defined as the set of nodes j connected to i by an edge

Gather:

Each node will create a message, which will be sent to other nodes later

Example: A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

Aggregate:

Node v will aggregate the messages from its neighbors $u \, \varepsilon \, N(v)$ and combines them into a single summary vector $\mathbf{h}_v^{(l)}$

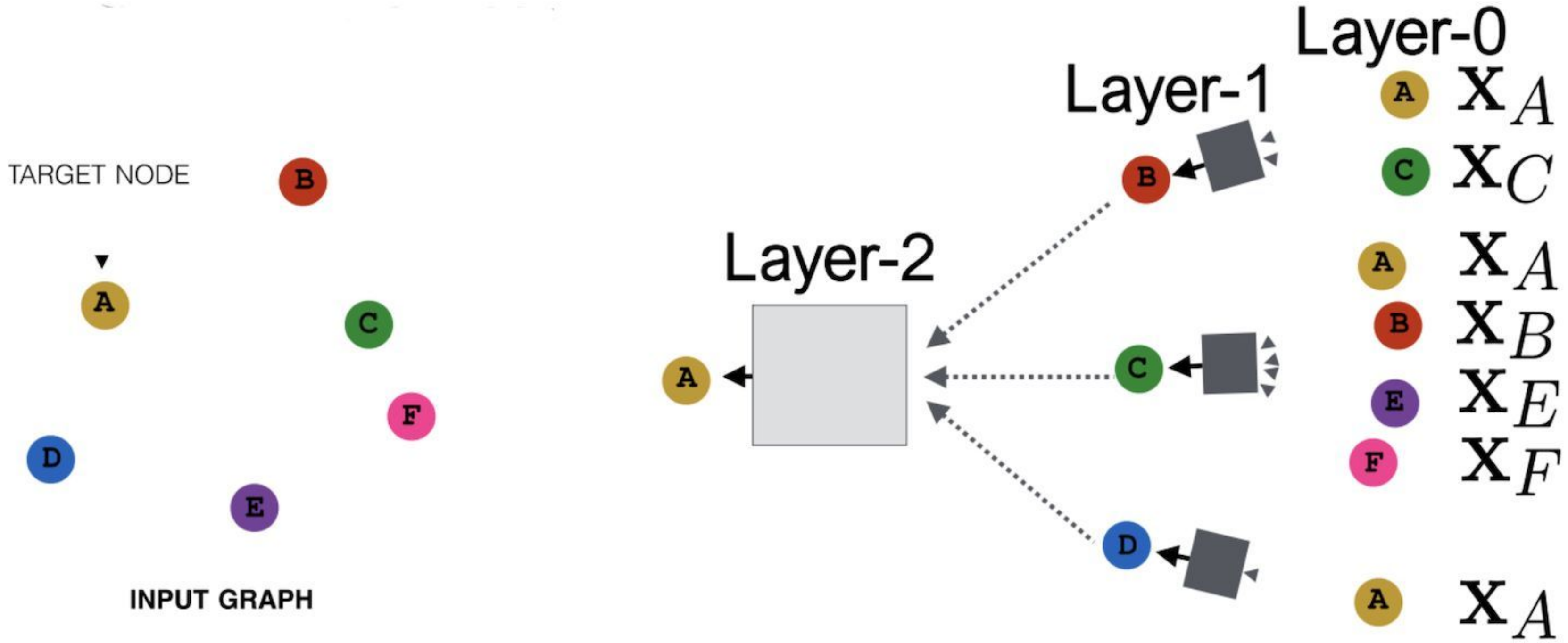
$$\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)$$

Node Update:

Node v combines the aggregated message $\mathbf{h}_v^{(l)}$ with its own feature vector from the previous layer $\mathbf{h}_v^{(l-1)}$. Easier to include $\mathbf{h}_v^{(l-1)}$ when computing $\mathbf{h}_v^{(l)}$.

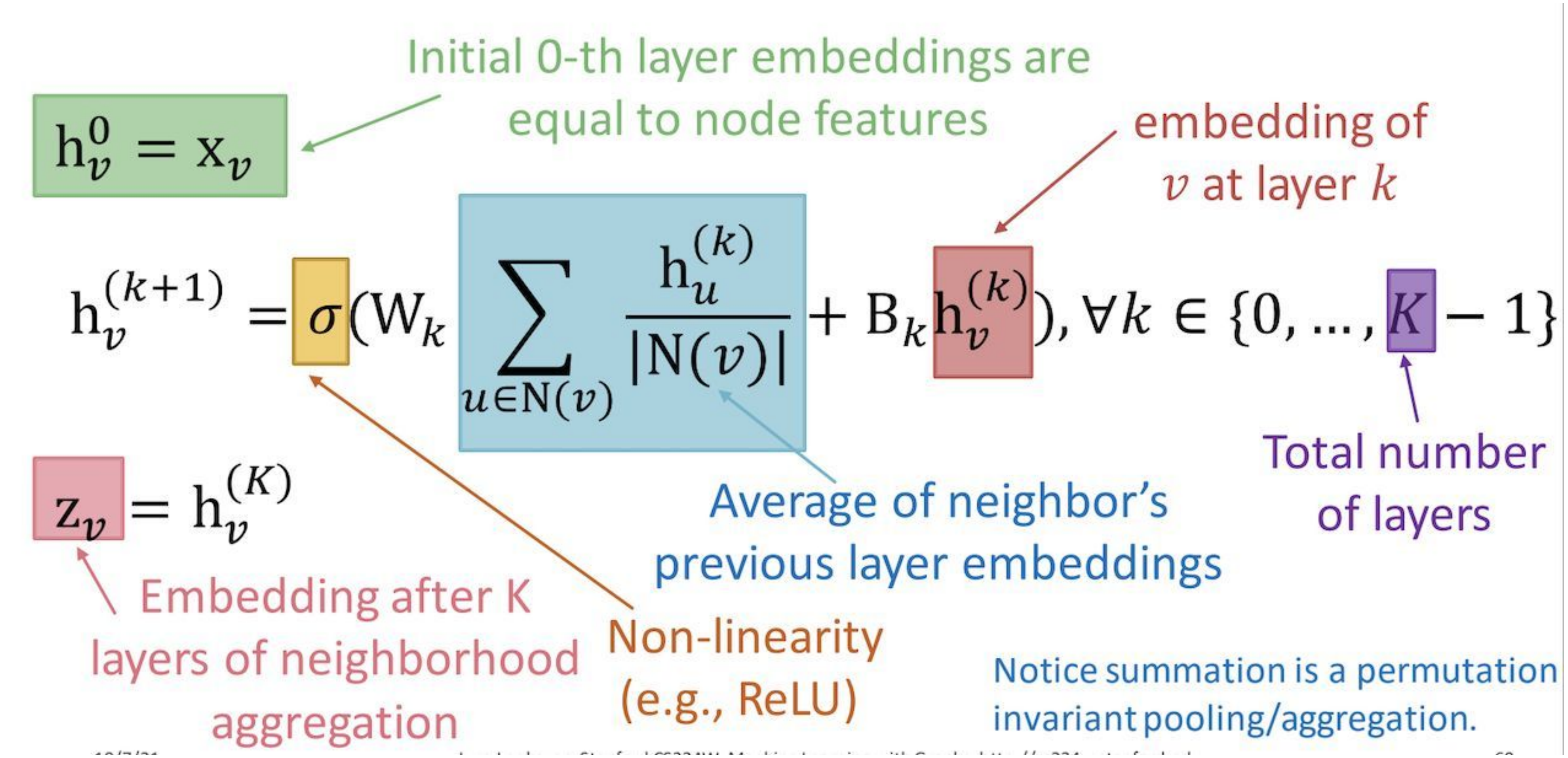
This process is repeated L times (where L is the number of GNN layers) to capture features from the L-hop neighborhood.

Graph Neural Networks (GNNs)



Jure Leskovec, Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.ed

Graph Neural Networks (GNNs)



$$\boxed{h_v^0 = x_v}$$

Initial 0-th layer embeddings are equal to node features

$$h_v^{(k+1)} = \boxed{\sigma}\left(W_k \boxed{\sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|}} + B_k \boxed{h_v^{(k)}}\right), \forall k \in \{0, \dots, \boxed{K} - 1\}$$

embedding of v at layer k

Total number of layers

Average of neighbor's previous layer embeddings

Non-linearity (e.g., ReLU)

$$\boxed{z_v} = h_v^{(K)}$$

Embedding after K layers of neighborhood aggregation

Notice summation is a permutation invariant pooling/aggregation.

Jure Leskovec, Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.ed

Aggregation Functions & Information Propogation

Aggregation function must be permutation-invariant. Commonly used are - Sum, Mean/Average, Max Pooling & Attention

Sum Aggregation: $h = \Sigma\, m$ (most common, permutation invariant)

Mean Aggregation: $h = (1/|N|)\, \Sigma\, m$ (normalized version)

Max Aggregation: $h = \max(m)$ (captures outliers)

Information Propagation

Layer 1 → Layer 2 → Layer L

Layer 1: Direct neighbors

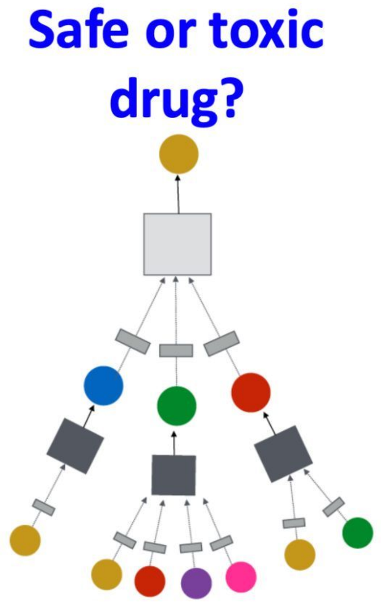
Layer 2: 2-hop neighbors

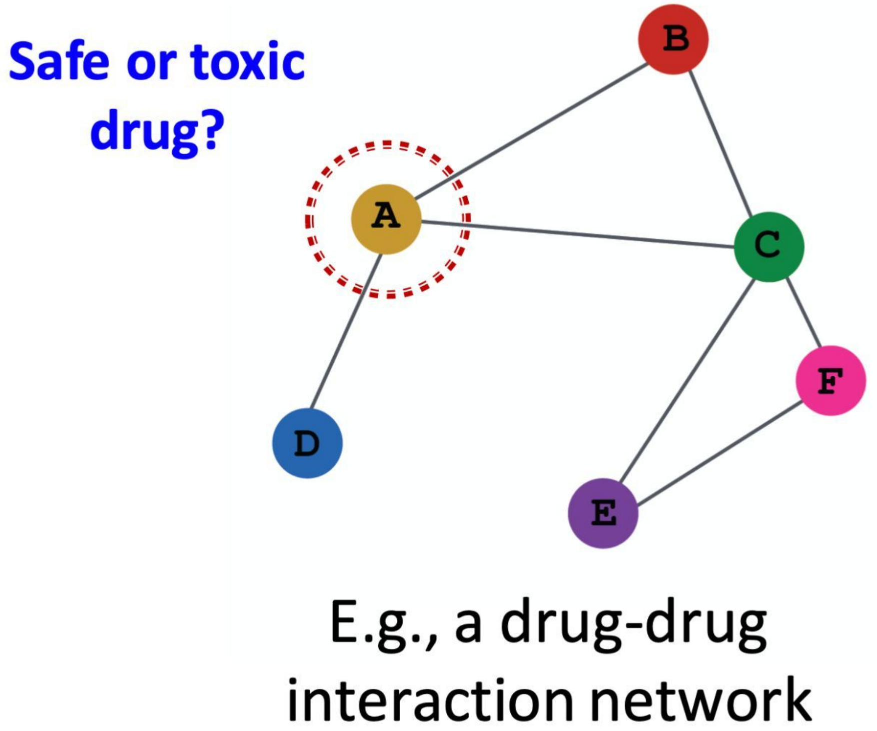
Layer L: L-hop neighborhood

Typical: 2-4 layers (diminishing returns & over-smoothing)

GNN Model Training

Directly train the model for a supervised task (e.g., node classification)





GNN Model Training

Use cross entropy loss for Node Classification

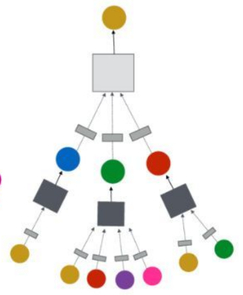
$$\mathcal{L} = -\sum_{v \in V} \boxed{y_v} \log(\sigma(\boxed{\mathrm{z}_v^{\mathrm{T}}}\boxed{\theta})) + (1 - \boxed{y_v})\log(1 - \sigma(\boxed{\mathrm{z}_v^{\mathrm{T}}}\boxed{\theta}))$$

Encoder output:
node embedding

Classification
weights

Node class
label

Safe or toxic drug?



What Types of Graphs can we train?

Directed/undirected graphs:

Focus is more on the flow of information. Standard Graph Convolutional Networks (GCNs), GraphSAGE is typically implemented for undirected graphs

For directed graphs a node v will only aggregate the messages from its neighbors $u \in N(v)$ (nodes u such that $u \to v$)

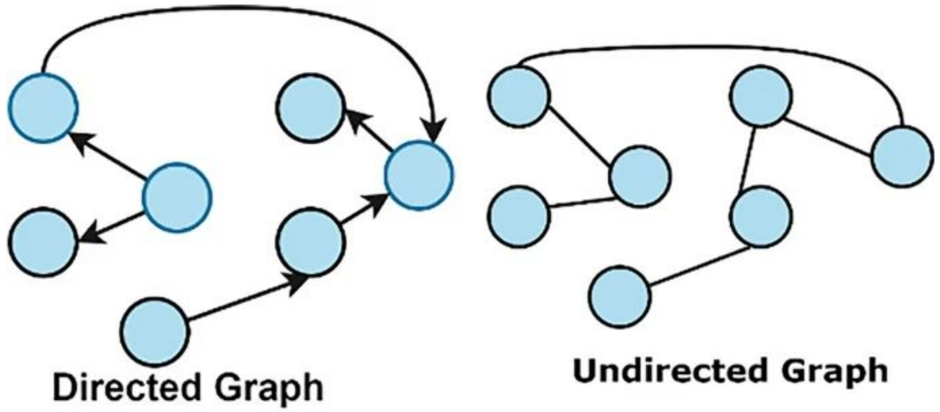


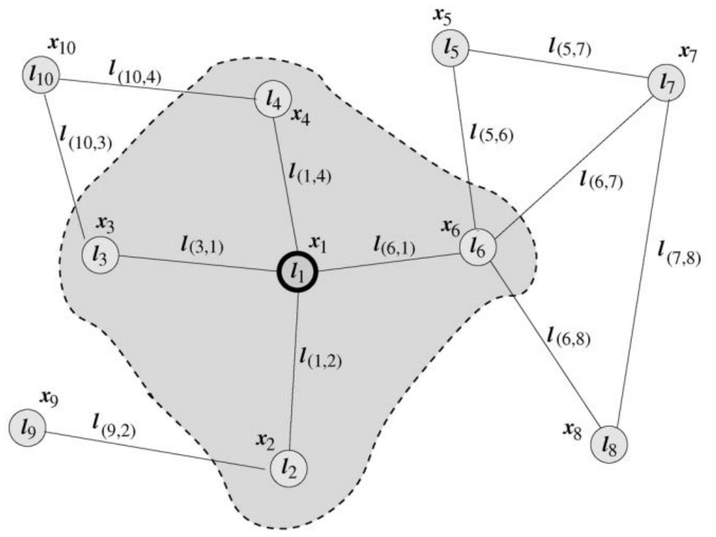
Image credit: Khemani et al. A review of graph neural networks

Are GNNs enough?

The first GNN paper proposed a recurrent approach that required iteration to a fixed point.

The Issue: Achieving convergence takes a significant, variable number of steps and requires backpropagation through a potentially very long sequence of updates.

This made the original GNNs computationally expensive and slow to train, especially on large graphs.



$$x_1 = f_w(l_1, \underbrace{l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}}_{l_{co[1]}}, \underbrace{x_2, x_3, x_4, x_6}_{x_{ne[1]}}, \underbrace{l_2, l_3, l_4, l_6}_{l_{ne[n]}})$$

Fig. 2. Graph and the neighborhood of a node. The state x_1 of the node 1 depends on the information contained in its neighborhood.

Image credit: The Graph Neural Network Model, Scarselli et. al

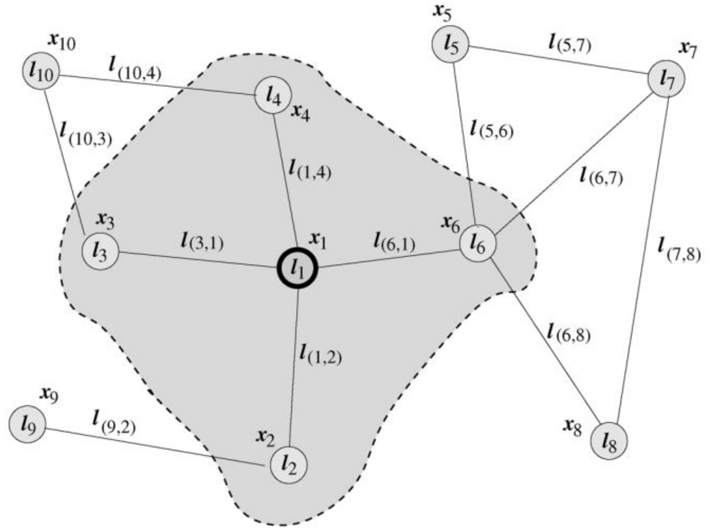
Are GNNs enough?

The first GNN paper proposed a recurrent approach that required iteration to a fixed point.

The Issue: Achieving convergence takes a significant, variable number of steps and requires backpropagation through a potentially very long sequence of updates.

This made the original GNNs computationally expensive and slow to train, especially on large graphs.

WAIT! Our MLPs also faced similar issue!!
What was the solution??



$$x_1 = f_w(l_1, \underbrace{l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}}_{l_{co[1]}}, \underbrace{x_2, x_3, x_4, x_6}_{x_{ne[1]}}, \underbrace{l_2, l_3, l_4, l_6}_{l_{ne[n]}})$$

Fig. 2. Graph and the neighborhood of a node. The state x_1 of the node 1 depends on the information contained in its neighborhood.

Image credit: The Graph Neural Network Model, Scarselli et. al

Are GNNs enough?

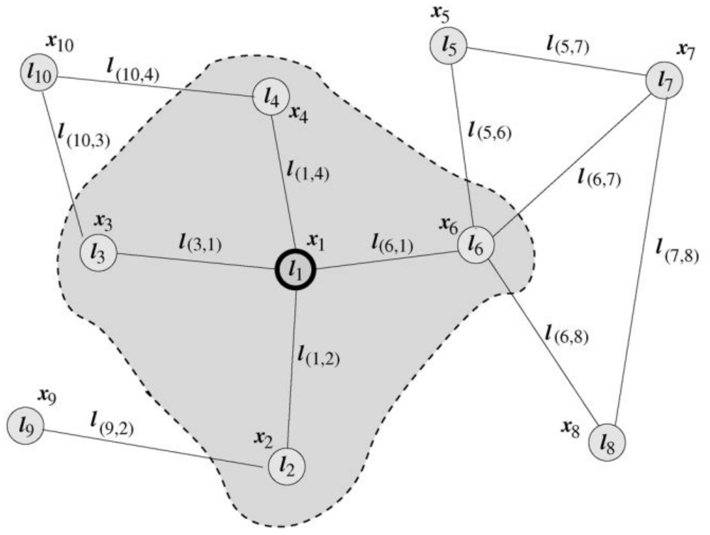
The first GNN paper proposed a recurrent approach that required iteration to a fixed point.

The Issue: Achieving convergence takes a significant, variable number of steps and requires backpropagation through a potentially very long sequence of updates.

This made the original GNNs computationally expensive and slow to train, especially on large graphs.

WAIT! Our MLPs also faced similar issue!!
What was the solution??

CONVOLUTION!!



$$x_1 = f_w(\underbrace{l_1, l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}}_{l_{co[1]}}, \underbrace{x_2, x_3, x_4, x_6}_{x_{ne[1]}}, \underbrace{l_2, l_3, l_4, l_6}_{l_{ne[n]}})$$

Fig. 2. Graph and the neighborhood of a node. The state x_1 of the node 1 depends on the information contained in its neighborhood.

Image credit: The Graph Neural Network Model, Scarselli et. al

What was the benefits of Convolution?

Extracts Local Patterns: Filters learn to recognize features by looking at a fixed, local neighborhood of pixels
Achieves Parameter Sharing: The same filter (weights) is applied across the entire image, making the model highly efficient and scalable
Is Translation Invariant: The model can recognize a feature regardless of where it appears in the image

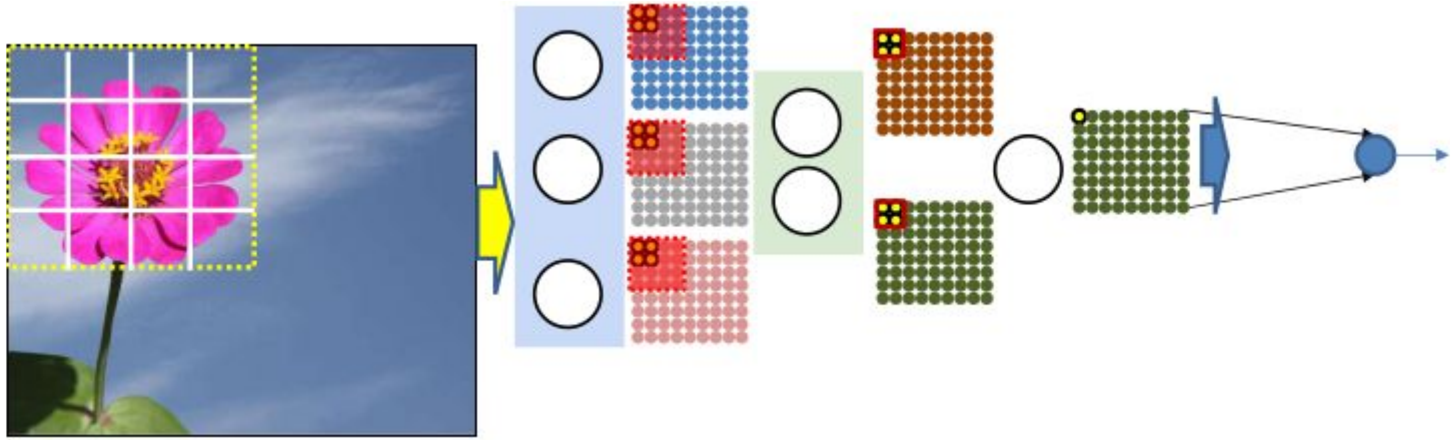


Image credit: 11-785, Prof. Bhiksha

Graph Convolutional Networks (GCN)

Foundational GNN architecture (Kipf & Welling 2017, Semi-supervised Classification with Graph Convolutional Networks)

Update rule:

h = σ(W * Σ (Â[i,j] * h))

Â = normalized adjacency matrix
W = learnable weight matrix
σ = ReLU activation

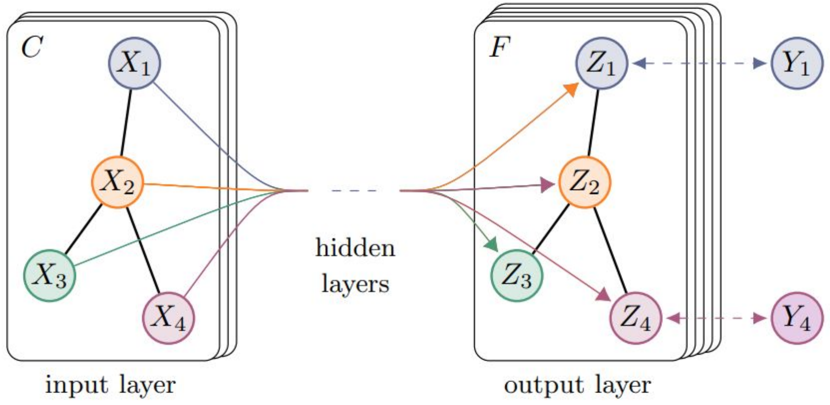


Image credit: Kipf et al. SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

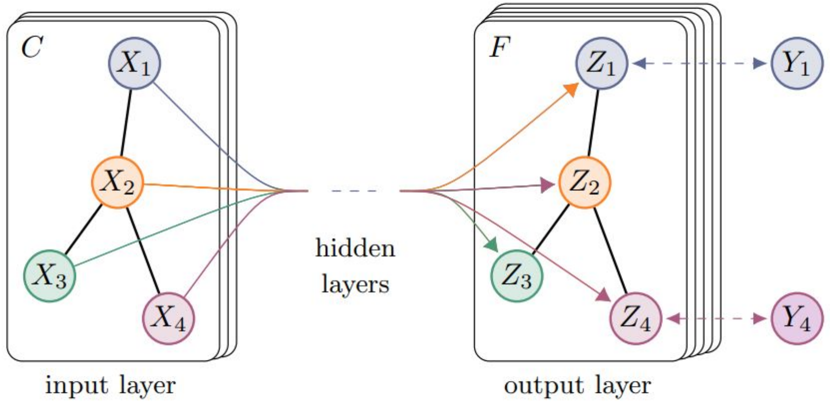
Graph Convolutional Networks (GCN)

Foundational GNN architecture (Kipf & Welling 2017, Semi-supervised Classification with Graph Convolutional Networks)

Update rule:

h = σ(W * Σ (Â[i,j] * h))

Â = normalized adjacency matrix
W = learnable weight matrix
σ = ReLU activation



This introduced scalable GCN model that uses the matrix multiplication formula to implement the message-passing (or convolution) step.

Solves convergence and complexity issues.

Pushed GNNs into the mainstream.

Image credit: Kipf et al. SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

GraphSAGE

Inductive learning through sampling (Hamilton et al. 2017)

Sample k random neighbors (not all)

Aggregate their embeddings

Learn inductive functions for new nodes

Key advantage: Works on unseen graphs, scalable

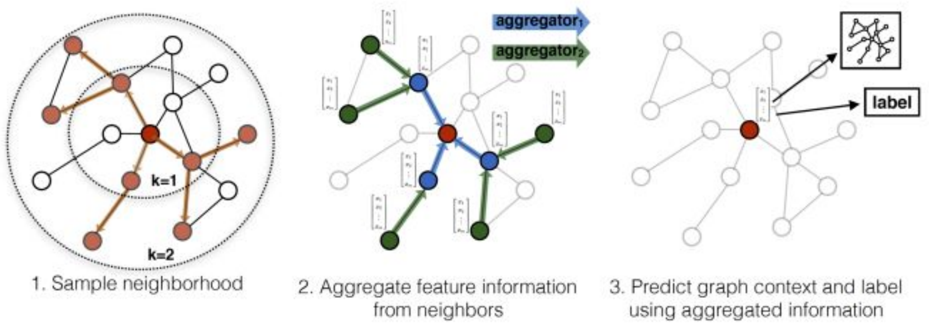
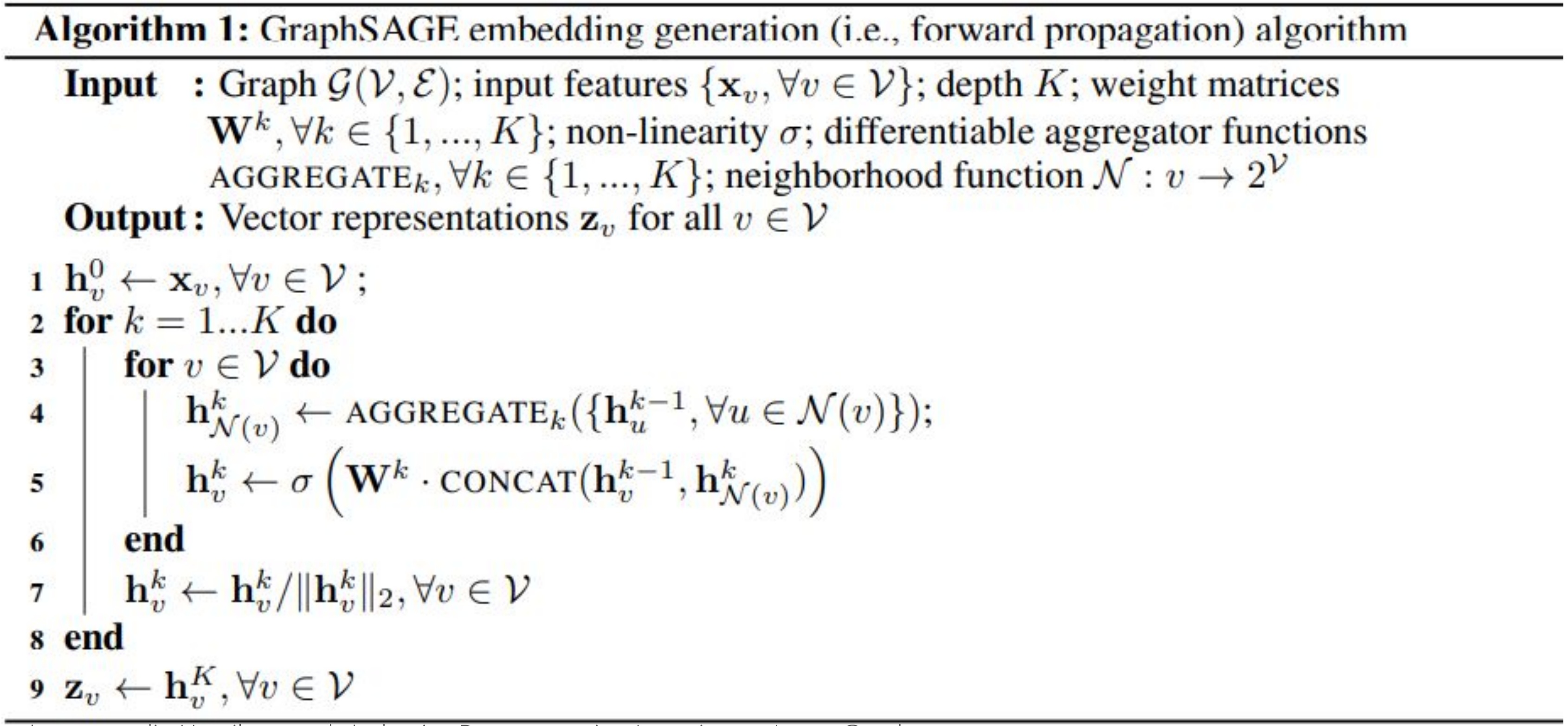

k=1
k=2
1. Sample neighborhood
aggregator₁
aggregator₂
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Image credit: Hamilton et al., Inductive Representation Learning on Large Graphs

Carnegie Mellon University

GraphSAGE



Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

```
$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
for $k = 1...K$ do
    for $v \in \mathcal{V}$ do
        $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
        $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
    end
    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

Image credit: Hamilton et al., Inductive Representation Learning on Large Graphs

Issues with GCNs

The standard GCN uses a predefined, isotropic (uniform) aggregation function. This means that when a node receives messages from its neighbors, it treats all neighbors equally

Issues with GCNs

The standard GCN uses a predefined, isotropic (uniform) aggregation function. This means that when a node receives messages from its neighbors, it treats all neighbors equally irrespective of their importance, feature relevance, or the nature of their connection.

GCN update rule

$$\mathbf{H}^{(l+1)} = \sigma\left(\overbrace{\boxed{\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}}}^{\text{Normalization Term}}\underbrace{\boxed{\mathbf{H}^{(l)}\mathbf{W}^{(l)}}}_{\text{Feature Transformation}}\right)$$

Before aggregation, the current node features are transformed by a learnable weight matrix (shared by all nodes)

The normalized adjacency matrix (Â) is the "convolution" part

SO WE NEED “Attention”

Graph Attention Networks (GAT)

Learn adaptive neighbor weights (Veličković et al. 2018)

Attention mechanism:

α = softmax(LeakyReLU(a[Wh||Wh]))

Each node learns which neighbors are important

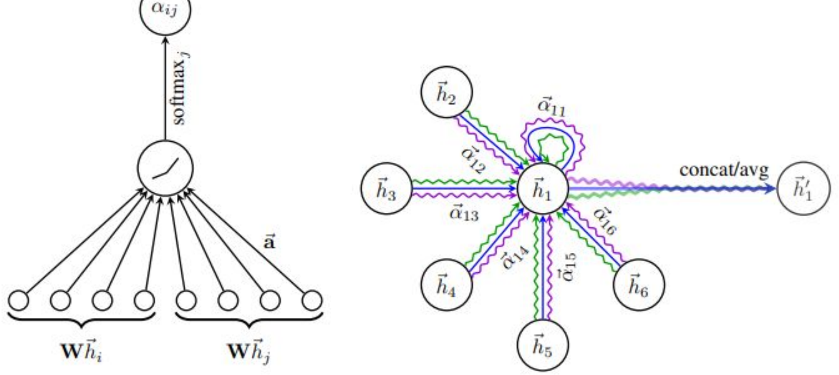


Image credit: Veličković et al., GRAPH ATTENTION NETWORKS

Training & Optimization

Layer depth: 2-4 layers optimal (over-smoothing at depth)

Pooling: Global sum/mean for graph-level tasks

Batching: Mini-batch graphs or node sampling

Regularization: Dropout, weight decay, batch norm

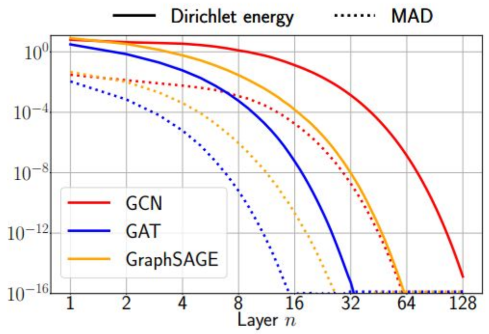
The Over-Smoothing Problem

As layers increase, node embeddings converge to similar values

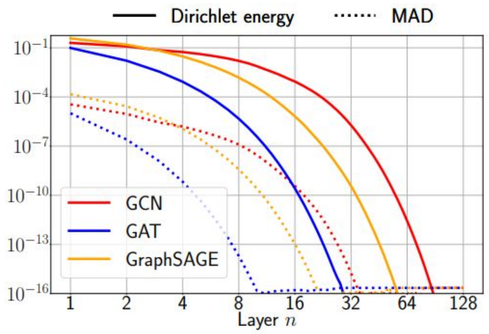
Problem:

Deep GNNs lose expressiveness. Nodes become indistinguishable.

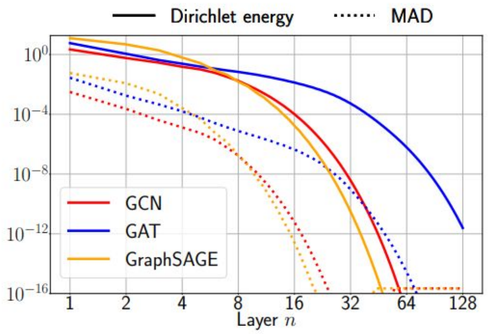
This is why 2-4 layers is standard practice



small-scale Texas graph



medium-scale Cora citation network



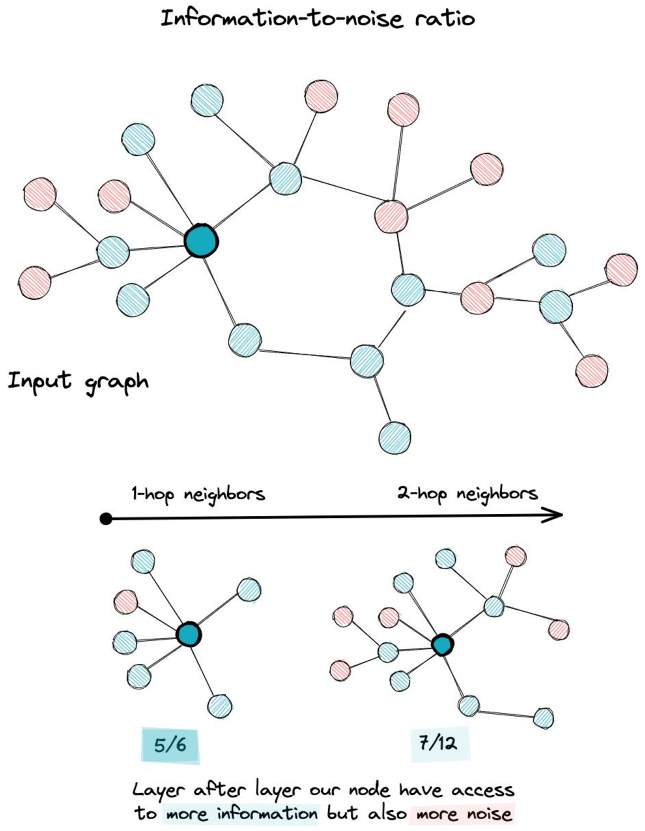
large-scale Facebook network (Cornell5)

Image credit: Rusch et al., A SURVEY ON OVERSMOOTHING IN GRAPH NEURAL NETWORKS

The Over-Smoothing Problem

This generalization was built on the main hypothesis that while nodes are interacting, they have **either access to important information** from nodes from the same class **or noise** by interacting with nodes from other classes.

While node access to more parts of the graph we may access to noisy nodes that affect the final embedding.


Information-to-noise ratio
Input graph
1-hop neighbors
2-hop neighbors
5/6
7/12
Layer after layer our node have access to more information but also more noise

Credit: Aomar Anas, Over smoothing issue in graph neural network, towards datascience

GAT's Limitation

- Like the GCN, the GAT still operates within the **confines of the Message Passing paradigm** i.e aggregates information from a node's immediate neighbors (1-hop)

- Calculates **weights based on node features and local connectivity**, it doesn't explicitly encode the full positional or structural relationship between two nodes that are far apart

- Attention is calculated only on a **subset of features** (neighbors), limiting the complexity of the feature interactions

To capture global patterns or long-range dependencies, you must stack many layers (L).

Leads to the over-smoothing problem

GAT's Limitation

- Like the GCN, the GAT still operates within the **confines of the Message Passing paradigm** i.e aggregates information from a node's immediate neighbors (1-hop)

- Calculates **weights based on node features and local connectivity**, it doesn't explicitly encode the full positional or structural relationship between two nodes that are far apart

- Attention is calculated only on a **subset of features** (neighbors), limiting the complexity of the feature interactions

To capture global patterns or long-range dependencies, you must stack many layers (L).

Leads to the over-smoothing problem

SO WE NEED GLOBAL ATTENTION!

Transformer??

Recap Transformers

- Self-Attention is used to weigh the importance of all other words in the sequence when processing one word.
- Compute attention scores using a function of Q(Query), K (Key), and V (Value) matrices, typically via the Scaled Dot-Product Attention
- Use a Feed-Forward Network block after the attention mechanism to enhance expressiveness.

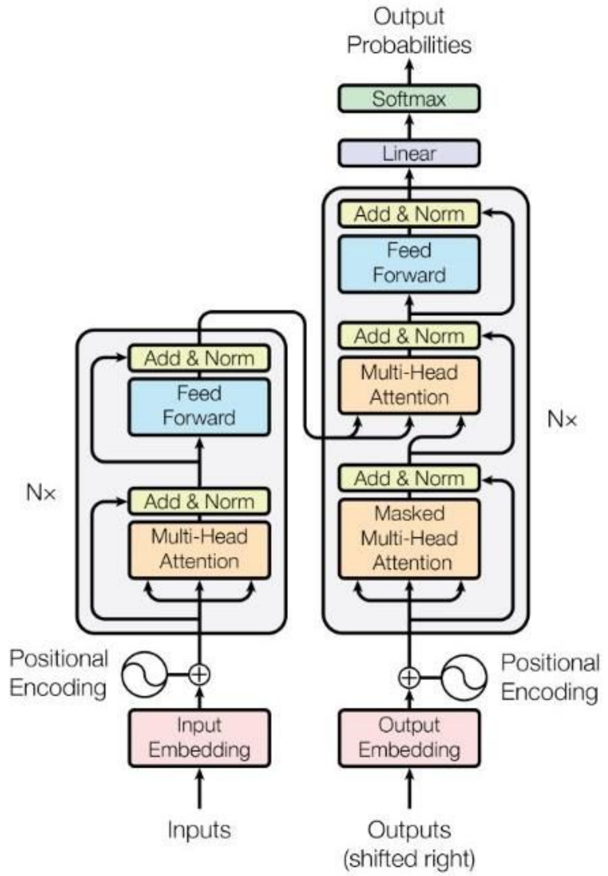


Image credit: Vaswani, A. "Attention is all you need."

Recap Transformers

- **Self-Attention** is used to weigh the importance of all other words in the sequence when processing one word.
- Compute attention scores using a function of **Q(Query), K (Key), and V (Value) matrices**, typically via the Scaled Dot-Product Attention
- Use a **Feed-Forward Network** block after the attention mechanism to enhance expressiveness.

The key mechanisms are the same for Graph Transformer

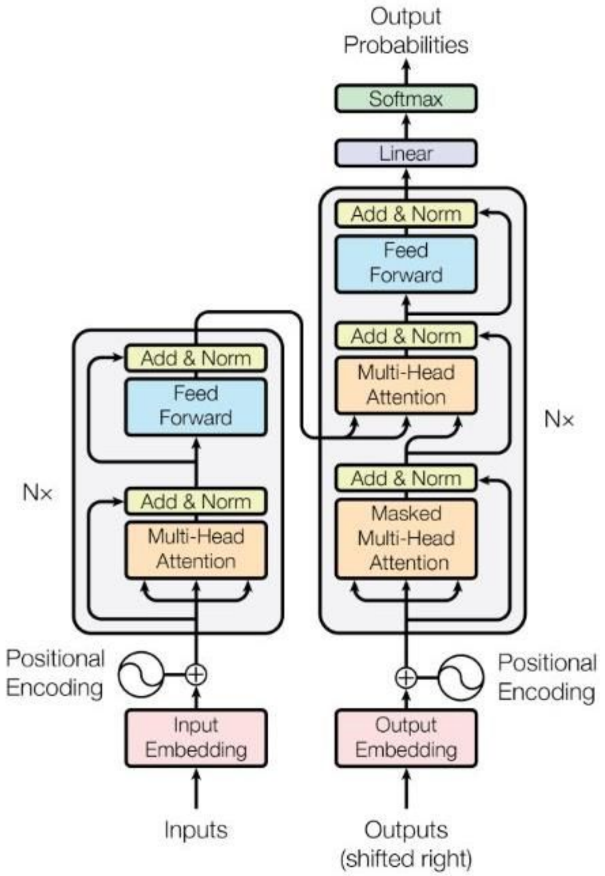


Image credit: Vaswani, A. "Attention is all you need."

Recap Transformers

- **Self-Attention** is used to weigh the importance of all other words in the sequence when processing one word.
- Compute attention scores using a function of **Q(Query) K (Key), and V (Value) matrices**, typically via the Scaled Dot-Product Attention
- Use a **Feed-Forward Network** block after the attention mechanism to enhance expressiveness.

The key mechanisms are the same for Graph Transformer

Any Differences??

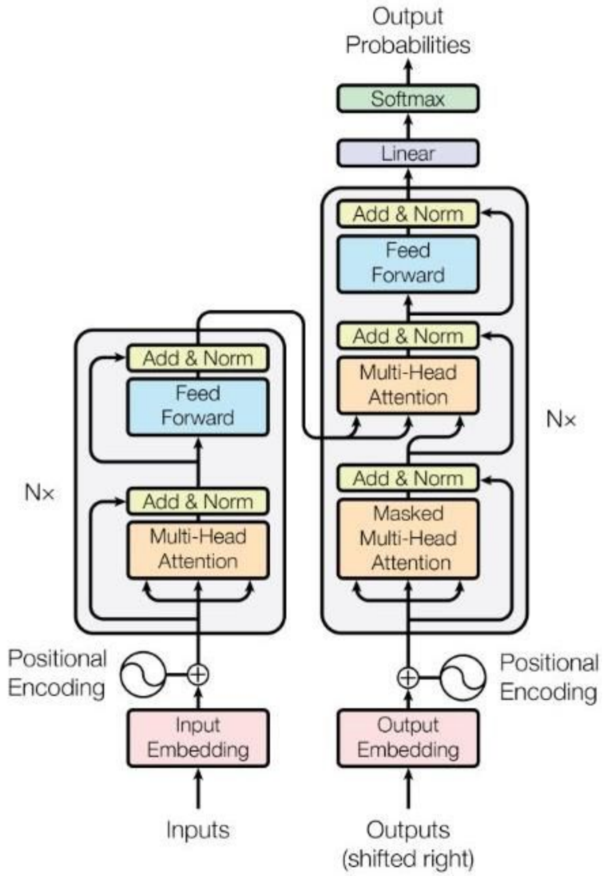


Image credit: Vaswani, A. "Attention is all you need."

Transformer(NLP) vs Graph Transformer(GNN) Differences

The differences arise from the need to adapt the sequence-based operations of the original Transformer to the irregular, non-sequential structure of a graph

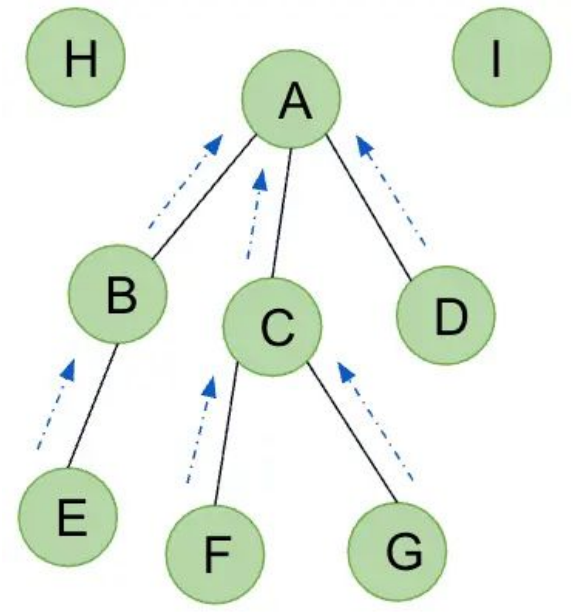
Attention Scope: Customizable for Local Attention (similar to GAT) or Global Attention

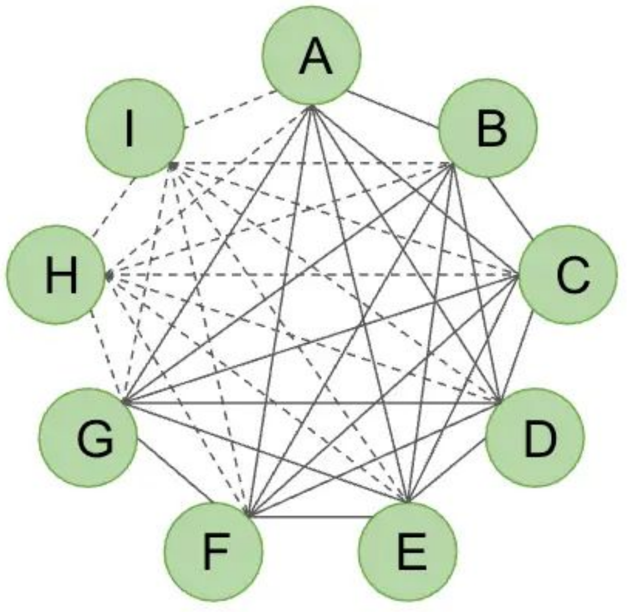
Positional Encoding: Cannot use sinusoidal function or learnable vector. No sequential order.
Must incorporate Graph Structure Encoding

Structural Encoding: Explicitly models graph properties

Masking: Cannot use Causal Masking. Adjacency Masking is an option

Graph Transformer(GNN) Information Flow





Left: **GNN with local message passing**. Disconnected nodes like “H” and “I” do not influence “A.”
Right: **Graph Transformer with global attention**. “A” can attend to all nodes, capturing long-range dependencies.

Credit: Lopez,Fey,Leskovec. "An Introduction to Graph Transformers" - kumo.ai

Graph Transformers

Full attention over all nodes (no just neighbors)

Spectral Transformers: Learnable PE (positional encoding)

Graph-scaled attention: Efficiency via subgraph sampling

- SOTA on many benchmarks

Blurs lines between local and global information

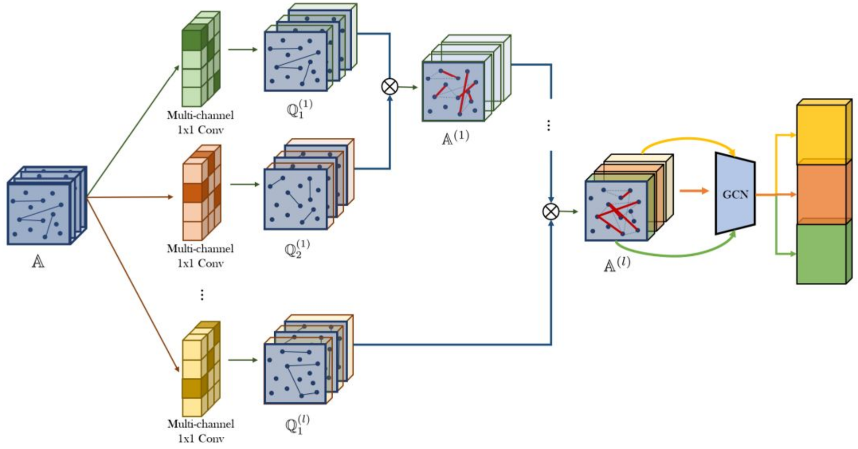


Image credit: Yun et al., Graph Transformer Networks

Confused??

GNNs vs GCNs vs GAT vs GT

Confused??

GNNs vs GCNs vs GAT vs GT

Forgot what they stand for?

Carnegie Mellon University

Confused??

Graph Neural Network(GNNs) vs

Graph Convolutional Networks(GCNs) vs

Graph Attention Networks(GAT) vs

Graph Transformer(GT)

Summary

Architecture	Pros	Cons	When to Use
GNN	Message-passing paradigm.	Not a model	Parent of all Graph model!
GCN	Computationally fast	Treats all neighbors equally	For a quick, efficient solution on medium-sized, simple graphs
GAT	Assigns different, learned importance scores to neighbors using attention	High Compute Cost. Still limited to local aggregation. risk over smoothing in deep models	High-Relational Data (social networks or knowledge graphs)
Graph Transformer	Long-range dependencies, structural encoding No over-smoothing	Highest Compute Cost. Attention $O(N^2)$ for N nodes. Complicated	Long-range dependencies are critical. Prioritize expressiveness over speed. Eg (protein folding)

Ecosystems for GNNs

PyTorch Geometric

Industry standard, rich models, production-ready

DGL

Framework-agnostic, optimized for large graphs

NetworkX

Data preprocessing, analysis, visualization

Image credit: pytorch-geometric, dgl.ai, networkx.org

NetworkX Example

A temporal, directed knowledge graph illustrating a person's health triage information. Arrows show progression from symptoms to diagnostic tests and subsequent treatment cycles.

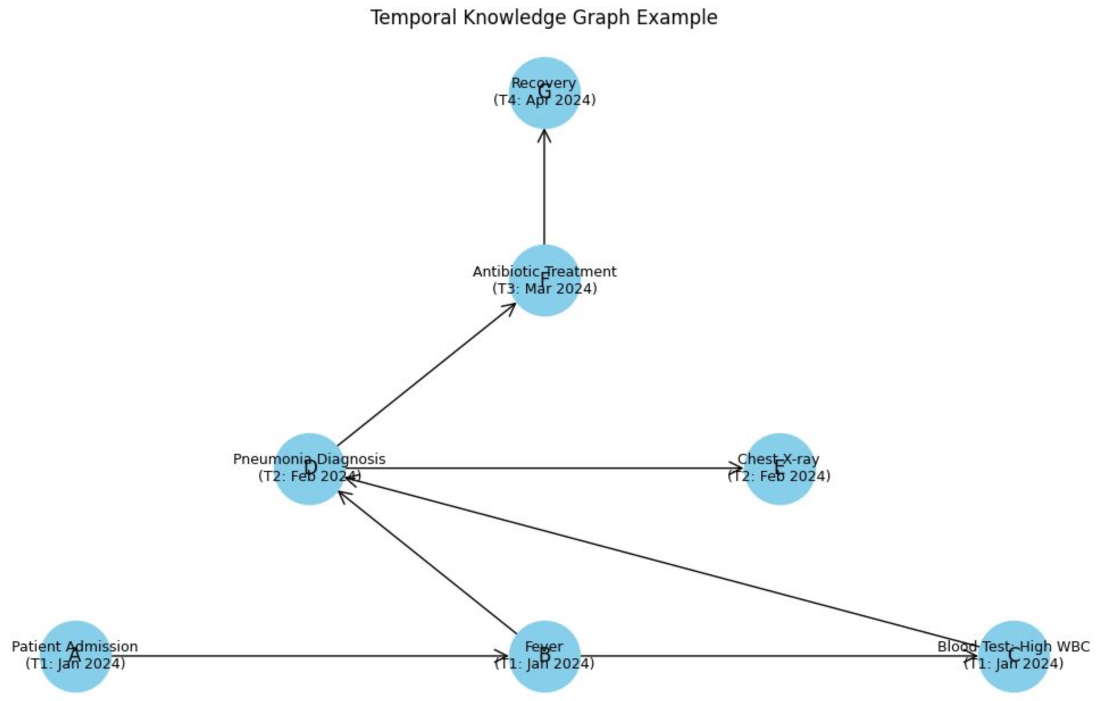


Image credit: Rutvik Joshi, 11785 Fall 25 TA

Going Deeper

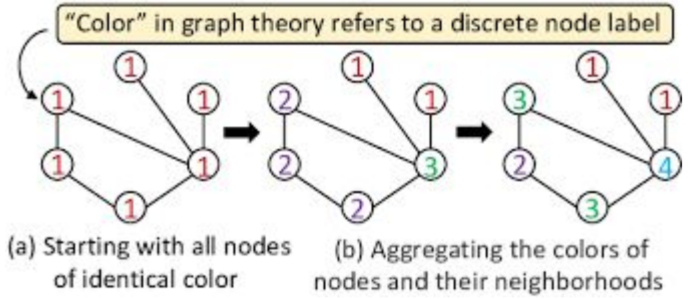
Advanced Topics to Explore

Expressivity: Weisfeiler-Lehman Test

Not all GNNs can distinguish all graph structures

- WL-test: Iterative node coloring algorithm
- GIN (Graph Isomorphism Networks): Matches WL power
- Higher-order GNNs: Use subgraph patterns

Current research: GNNs beyond message passing


"Color" in graph theory refers to a discrete node label
(a) Starting with all nodes of identical color
(b) Aggregating the colors of nodes and their neighborhoods

Two iterations of the Weisfeiler-Lehman test on an example graph, starting with nodes of identical color/feature 1. The test performs graph coloring by first aggregating the colors of nodes and their neighborhoods and then generating unique new colors. The colors embed the structural roles of
vertices in the graph. For simplicity, we assign the same initial color for all the nodes. However, if each node is assigned a unique feature vector that captures other non-structural properties, such information will also be accounted for when labeling the nodes.

Credit: Alrahis et al., Graph Neural Networks for Reverse Engineering of Gate-Level Netlists

Equivariance Groups

- S_n, Permutation Group: Reordering elements (like node indices) without changing the underlying structure or connections.
- $T(3)$, Translation Group: Shifting all points by a constant vector in space without rotating or distorting the shape.
- $SO(3)$, Special Orthogonal Group: Rotating objects around a fixed point (origin) while preserving their orientation and shape.
- $SE(3)$, Special Euclidean Group: Rigid body motions that combine both rotation and translation, preserving relative distances and orientation (chirality).
- $E(3)$, Euclidean Group: The full set of spatial transformations including rotation, translation, and reflection (mirroring), preserving distances but not necessarily orientation.

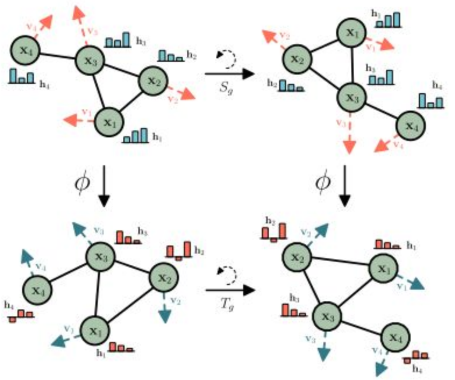


Image credit: Satorras et al., E(n) Equivariant Graph Neural Networks

Carnegie Mellon University

Equivariant & Geometric GNNs

For molecules and 3D structures, symmetries matter

- SE(3)-equivariant networks: Rotation invariant
- SchNet, NequIP: Molecular geometries
- Enables: Drug discovery, material design

Ready for Some more Concepts??

Carnegie Mellon University

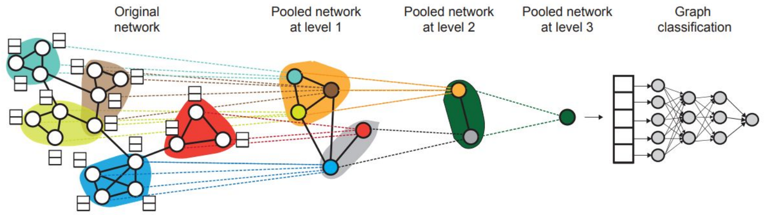
Hierarchical Graph Pooling

Coarsen graphs like CNNs pool spatial dimensioı

DiffPool: Differentiable pooling with assignm

TopK pooling: Select important nodes

Benefit: Multi-scale graph representations


Original network
Pooled network at level 1
Pooled network at level 2
Pooled network at level 3
Graph classification

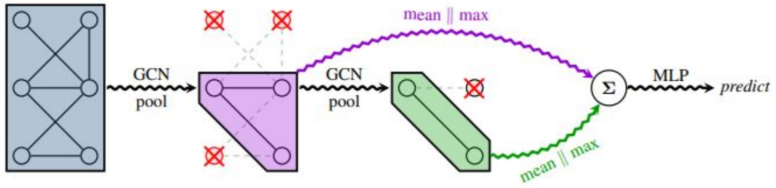


Image credit: Ying et al., Hierarchical Graph Representation Learning with Differentiable Pooling
Cangea et al., Towards Sparse Hierarchical Graph Classifiers

Spectral Graph Convolutions

- Signals on graphs: Laplacian eigenbasis approach

- Graph Fourier Transform: Eigendecomposition
- Chebyshev approximation: Efficient spatial filtering
- Connection: Spatial methods approximate spectral

- Theory-grounded signal processing perspective

Generative Models for Graphs

- Generate new graphs by learning structure and features
- Graph autoencoders: Encode → latent → decode
- Variational approaches: VAE for graphs
- Applications: Molecule generation, design

Key Takeaways

Message passing: Universal mechanism for graph learning

Architectures: GCN, GraphSAGE, GAT → easy to build variants

Practical: 2-4 layers, watch for over-smoothing

Frontier: Expressivity, geometry, generation still open

Thank you

Open Research Directions

Scalability: Billion-node graphs, distributed training

Theory: Expressivity guarantees, universality results

Robustness: Adversarial attacks, certification, fairness