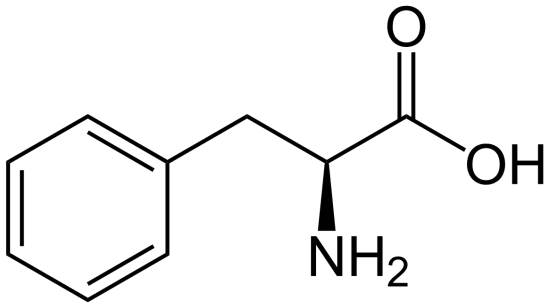
Graph Neural Networks

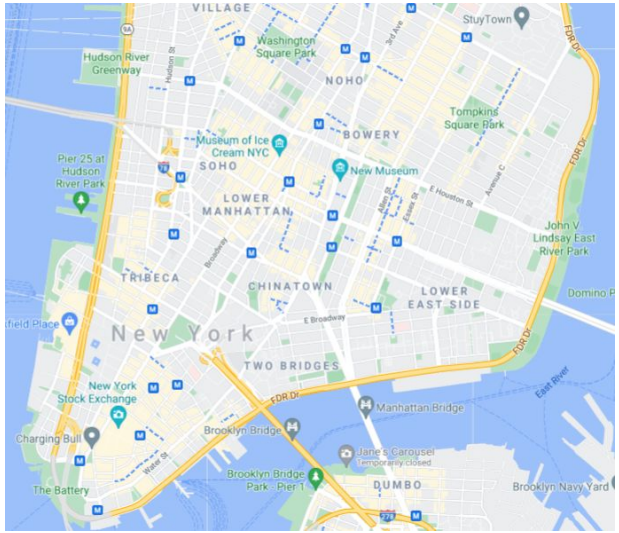
Everett Knag, Justin Saluja, Chaitanya Srinivasan, Prakarsh Yadav

11-785 Deep Learning Spring 2021

Graphs in the World



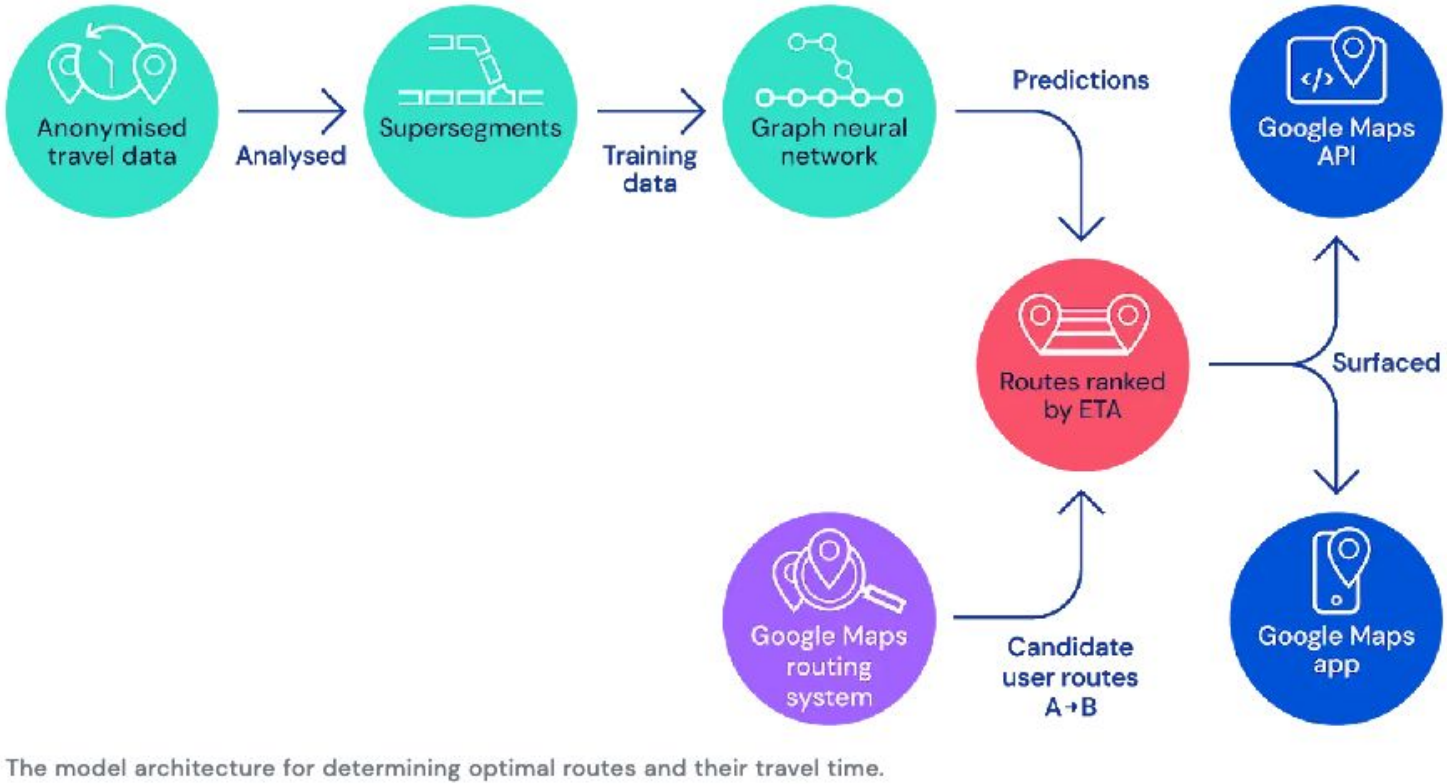
phenylalanine



Map of Manhattan

Social Network

Breakthrough in GNN



The model architecture for determining optimal routes and their travel time.

Image Credit: DeepMind

Breakthrough in GNN

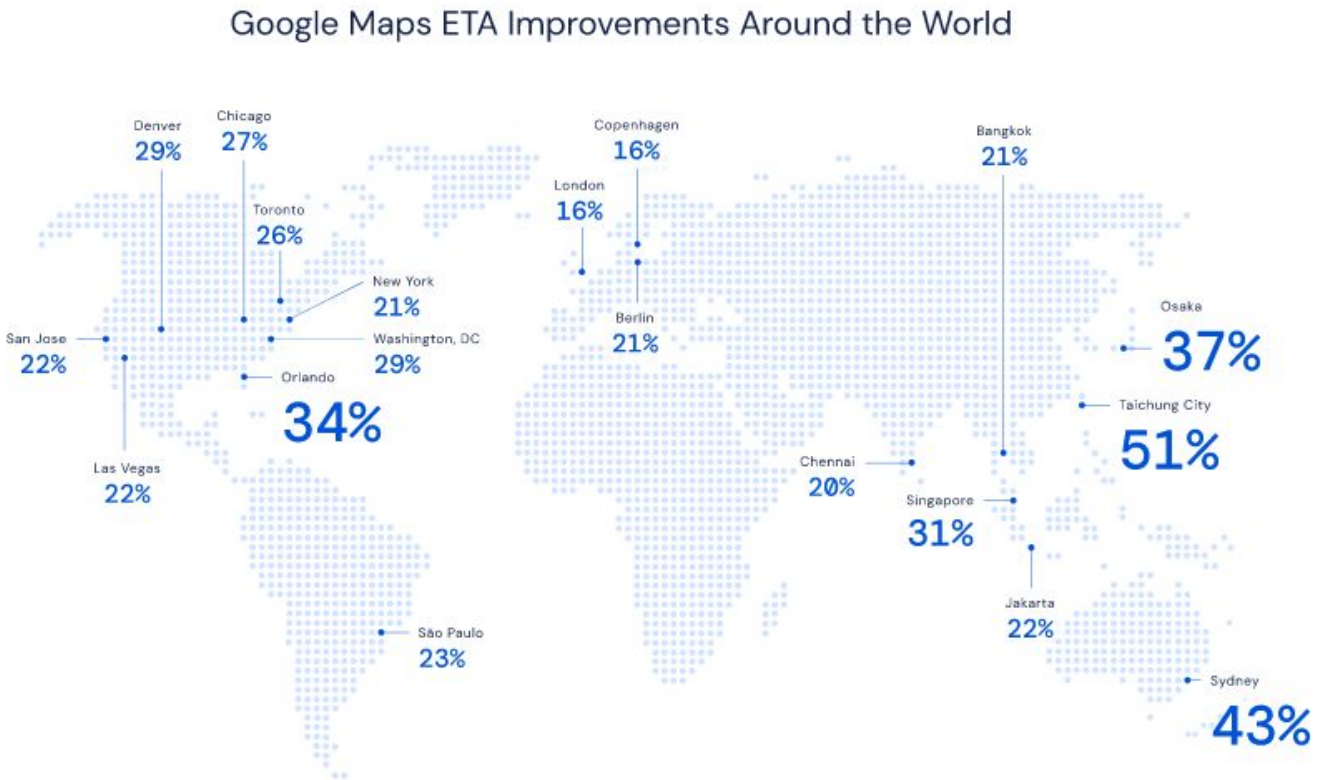
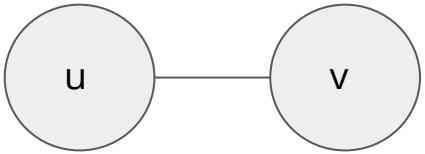


Image Credit: DeepMind

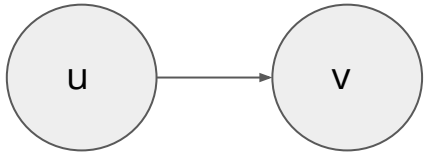
Graph Definitions

G = (V, E)

- V is a set of nodes
- E is a set of tuples of form (u, v), where there is an edge between u and v
- G is a graph



Undirected edge



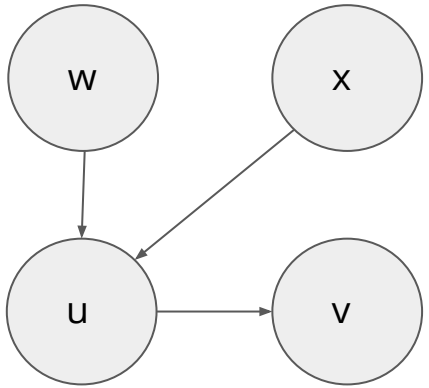
Directed edge

Graph encoding as a matrix

Adjacency Matrix: $\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$

- In this example, binary matrix encoding of a unweighted graph
- Rows/columns number the nodes, matrix elements encode edges

$V = \{u, v, w, x\}$; $E = \{(w, u), (x, u), (u, v)\}$



$\mathbf{A} =$

	u	v	w	x
u	0	**1**	0	0
v	0	0	0	0
w	**1**	0	0	0
x	**1**	0	0	0

(to)

(from)

Do the matrices encode the same graph?

0	**1**	0	0
0	0	0	0
1	0	0	0
1	0	0	0

0	0	0	0
0	0	**1**	0
1	0	0	0
0	0	**1**	0

Hint: Have we given you enough information?

They are the same encoding!

u	v	w	x	
0	**1**	0	0	u
0	0	0	0	v
1	0	0	0	w
1	0	0	0	x

v	w	u	x	
0	0	0	0	v
0	0	**1**	0	w
1	0	0	0	u
0	0	**1**	0	x

Considerations for GNN

1) Nodes are not i.i.d* (we are modeling an interconnected set of nodes)

2) A NN modeling a graph should be permutation invariant and equivariant

   - Adjacency matrix orders nodes arbitrarily

For permutation matrix **P**, function f that takes in an adjacent matrix **A**:

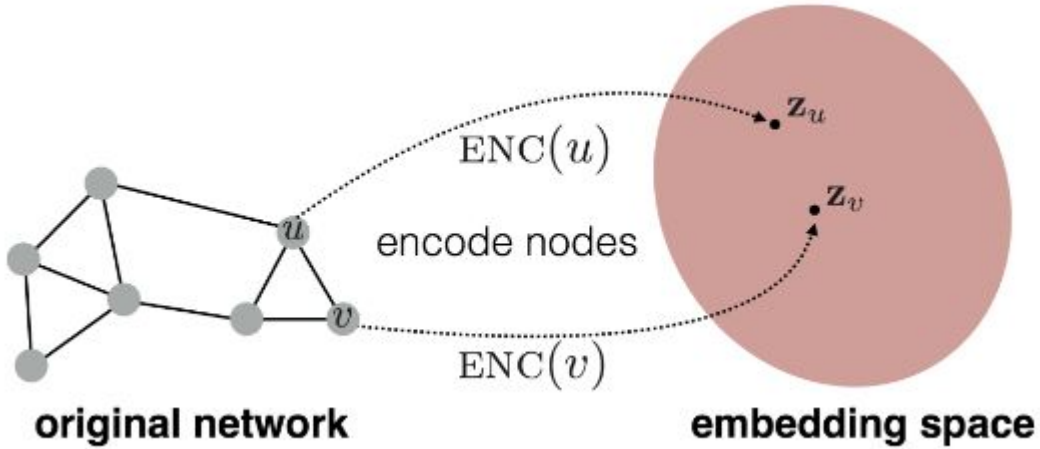
Permutation Invariance Property: $f(\mathbf{PAP^T}) = f(\mathbf{A})$

Permutation Equivariance Property: $f(\mathbf{PAP^T}) = \mathbf{P}f(\mathbf{A})$

*i.i.d = independent and identically distributed

Considerations for GNN

3) Find an encoding that preserves the graph structure



Insight: exploit homophily - a neighborhood of nodes tend to have shared attributes

Figure 3.1, *Graph Representation Learning*

NEURAL MESSAGE PASSING

Graph

$$G = (V, E)$$

Node Features

$$\boldsymbol{X} \in \mathbb{R}^{d \times |V|}$$

Node Embeddings

$$\boldsymbol{z_u}, \forall u \in V$$

Hidden embedding:

$$\boldsymbol{h} = \{\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_N}\}$$

Node vs Edge embeddings:

$$h_u^{(k)}, u \in V$$

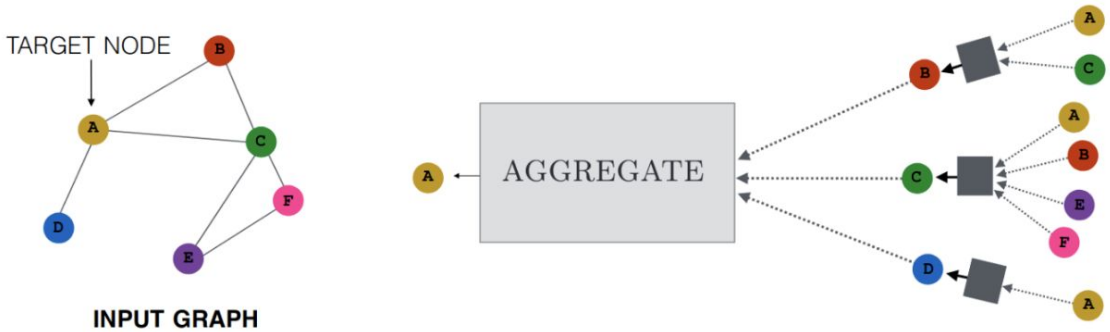
$$h_{(u,v)}^{(k)}, (u, v) \in E$$

Goal: Combine the information from neighboring nodes to encode contextual graph information

At every iteration, each node receives information from it's neighbors

The information is then combined with the current features with a learnable function

MESSAGE PASSING FRAMEWORK



$$h_u^{(k+1)} = \text{UPDATE}^{(k)}\left(h_u^{(k)}, \text{AGGREGATE}^{(k)}\left(\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\}\right)\right)$$

$$h_u^{(k+1)} = \text{UPDATE}^{(k)}\left(h_u^{(k)}, m_{\mathcal{N}(u)}^{(k)}\right)$$

Image source: https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf

AT EACH ITERATION k OF THE GNN:

- $AGGREGATE$ all embeddings from u's neighbors to generate a message $m_{\mathcal{N}(u)}^{(k)}$ based on this aggregated neighborhood information
- $UPDATE$ the embedding $h_u^{(k)}$ of node u by combining information from the previous embedding $h_u^{(k-1)}$ and with the message $m_{\mathcal{N}(u)}^{(k)}$

AFTER RUNNING K ITERATIONS:

- Use the output of the final layer to define the embeddings for each node:

$$z_u = h_u^{(K)}, \forall u \in V$$

THE BASIC GNN

$$h_u^{(k)} = \sigma\left(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_{\text{neigh}}^{(k)} \sum_{v \in N_u} h_v^{(k-1)} + b^{(k)} \right)$$

- $h_u^{(k-1)} \in \mathbb{R}^{d^{(k-1)}}$: Node embeddings
- $W_{\text{self}}^{(k)}, W_{\text{neigh}}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$: Learnable parameters
- $b^{(k)} \in \mathbb{R}^{d^{(k)}}$: Bias term
- σ: Elementwise non-linearity (e.g., a tanh or ReLU)

REMARKS

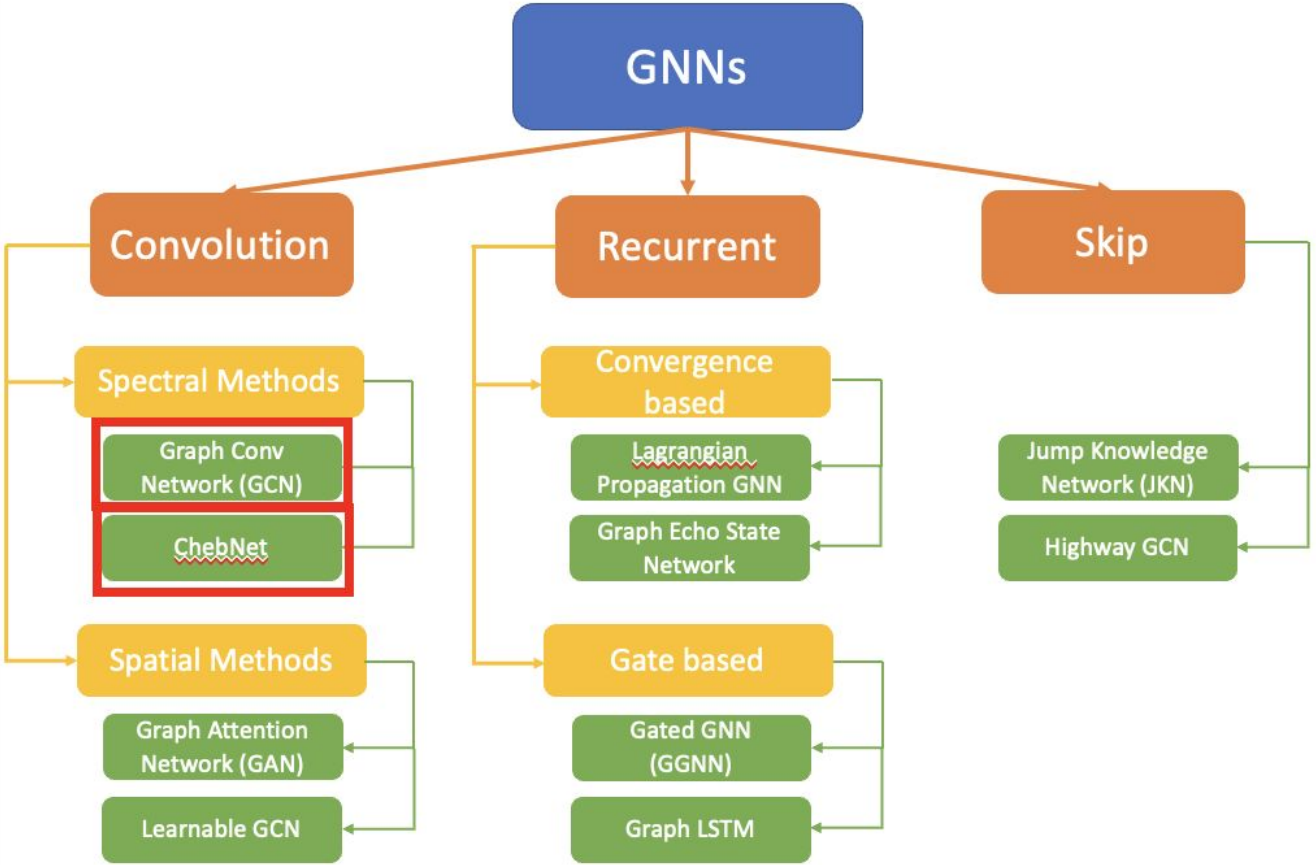
- The learnable parameters can be shared across GNN message passing iterations or trained separately for each layer
- We just described only the node-level GNN operations. There exists graph-level formalisms as well.
- The message passing example is analogous to a standard Multi-Layer Perceptron (MLP) in that it relies on linear operations followed by an elementwise non-linearity

SUMMARY

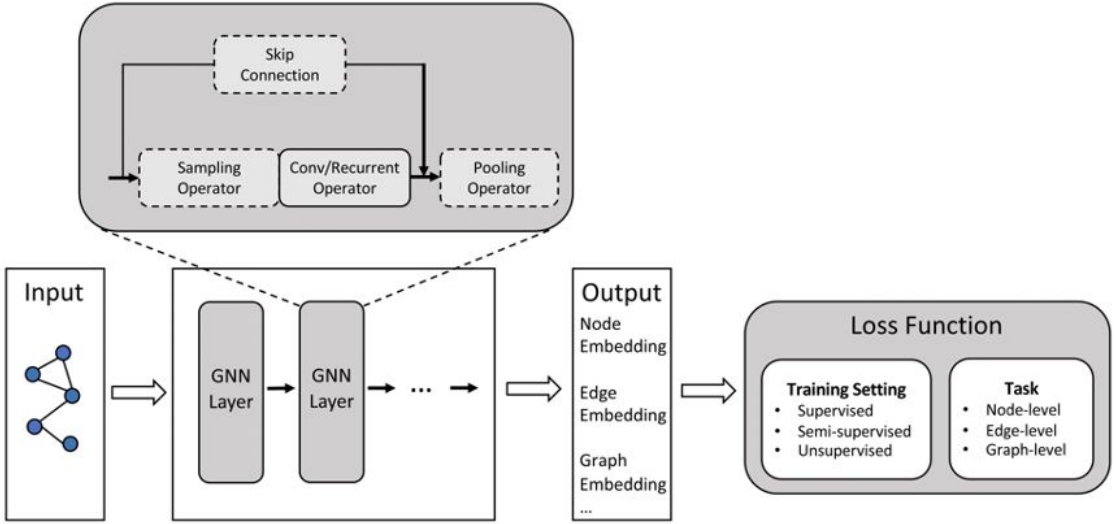
1. Sum the messages incoming from the neighbors
2. Combine the neighborhood information with the node's previous embedding using a linear combination
3. Apply an elementwise non-linearity

Deep learning on GNN

Division of GNNs



General structure of GNNs

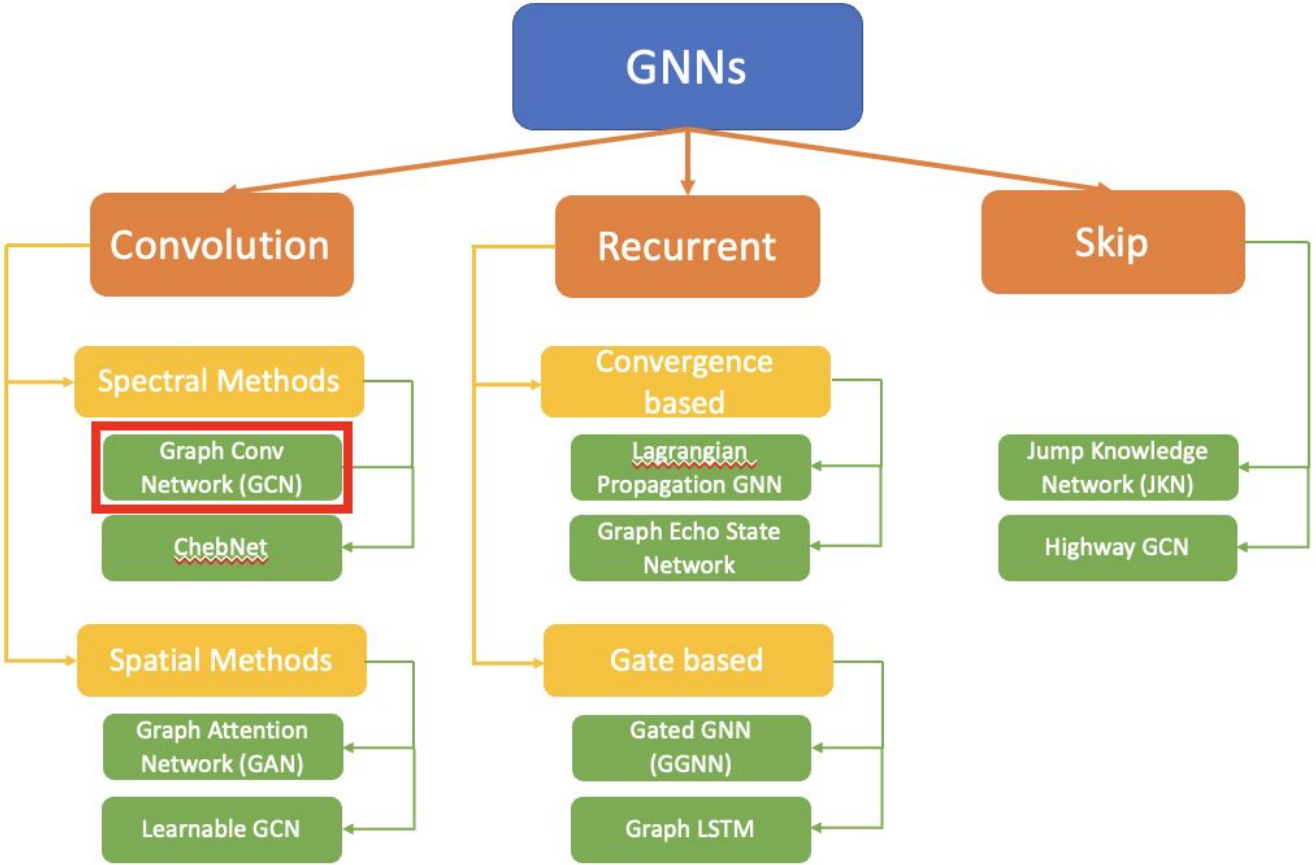


- **Spatial approaches** define convolutions directly on the graph based on the graph topology

- In **spectral methods**, an input graph signal X is firstly transformed to the spectral domain by the graph Fourier transform F, then the convolution operation is conducted. After the convolution, the resulted signal is transformed back using the inverse graph Fourier transform F^{-1}
- The spectral method needs at most O(n) parameters per feature map, allowing efficient forward propagation

Graph neural networks: A review of methods and applications, Jie Zhou et al in 2020

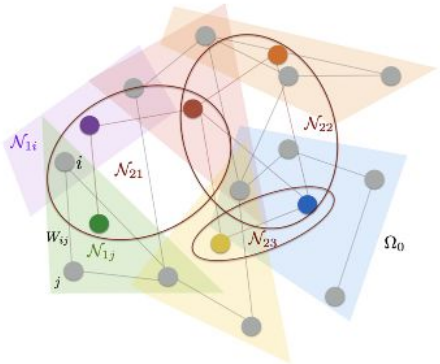
Division of GNNs



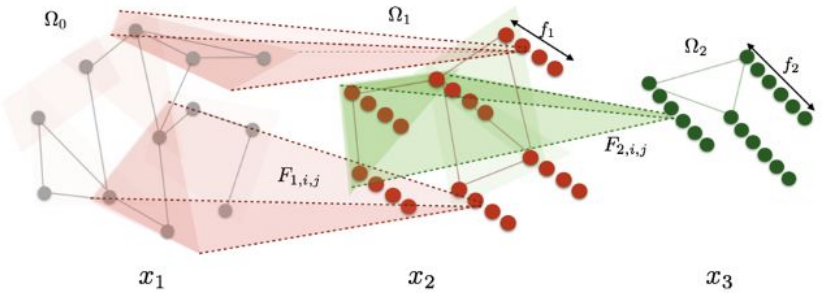
Convolutional GNNs

- Introduced in "Spectral Networks and Deep Locally Connected Networks on Graphs" by Bruna *et al,* in 2014
- Convolution on undirected graphs
- Defining graphs for convolution:
  - Undirected graph with Ω nodes and W edge weights, $G = (\Omega, W)$
  - Neighbors of a given node within threshold δ, $N_\delta(j) = \{i \in \Omega : W_{ij} > \delta\}$
  - Convolution operation on a given G, $x_{k+1,j} = L_k h\left(\sum_{i=1}^{f_{k-1}} F_{k,i,j} x_{k,i}\right) \quad (j = 1 \dots f_k)$

Undirected Graph definition

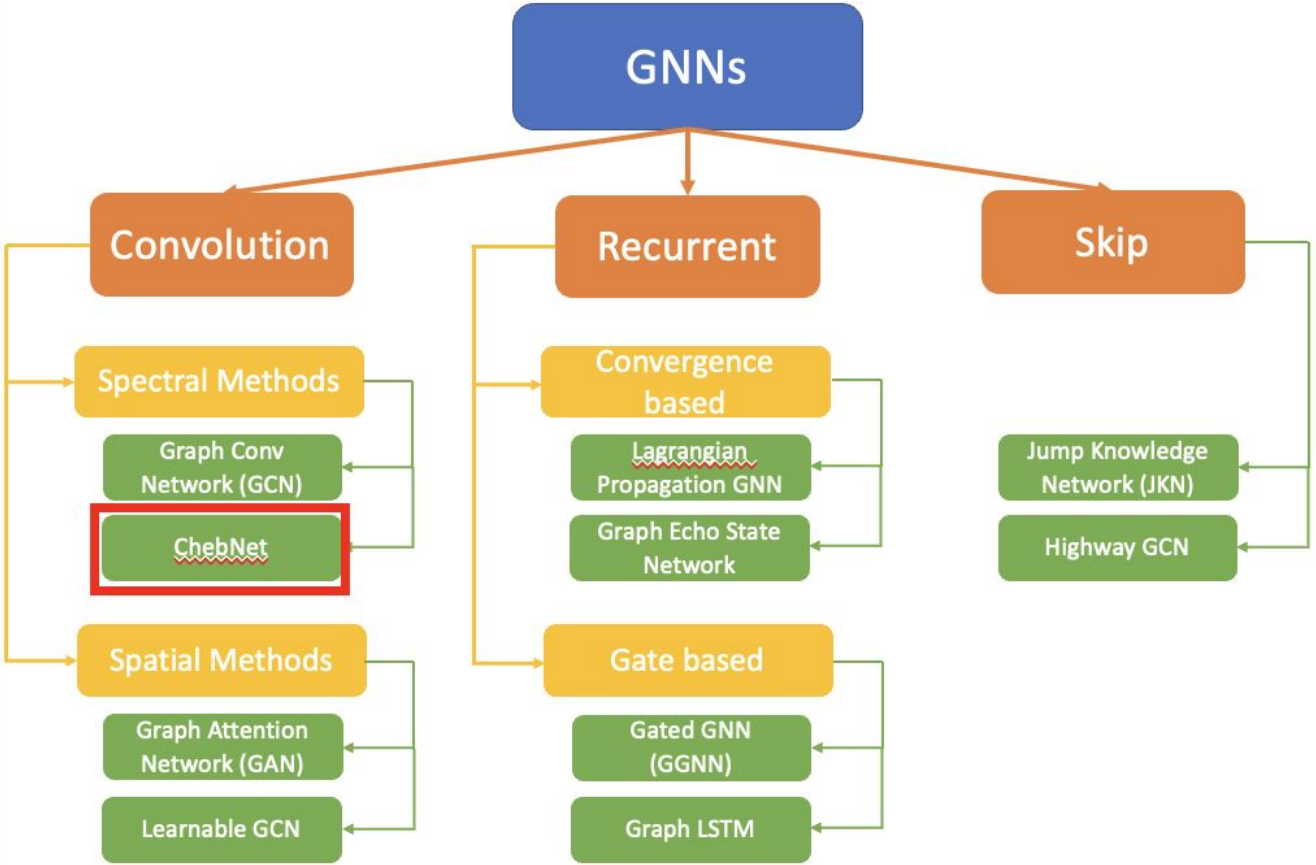


Convolution operation on undirected graph



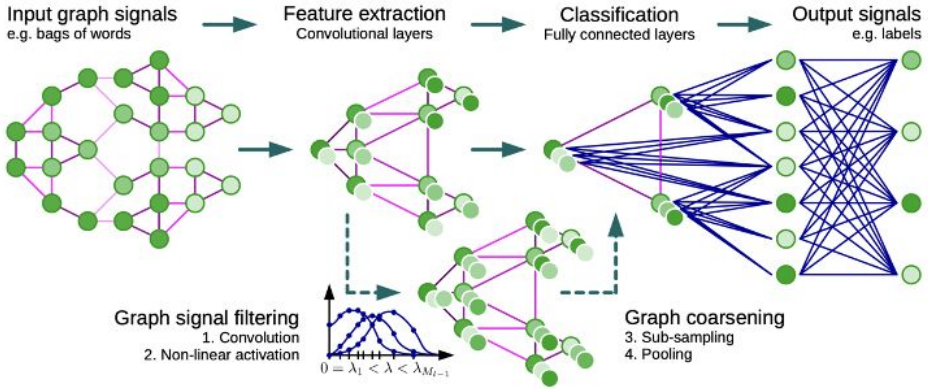
"Spectral Networks and Deep Locally Connected Networks on Graphs" by Bruna *et al,* in 2014.

Division of GNNs



Convolutional GNNs-Continued

- Introduced in "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" by Defferrard *et al,* in 2016
- Proposed 3 steps for CNN translation to graphs:
  i. The design of localized convolutional filters on graphs
  ii. a graph coarsening procedure that groups together similar vertices (Graclus multilevel clustering algorithm)
  iii. a graph pooling operation that trades spatial resolution for higher filter resolution



"Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" by Defferrard *et al,* in 2016

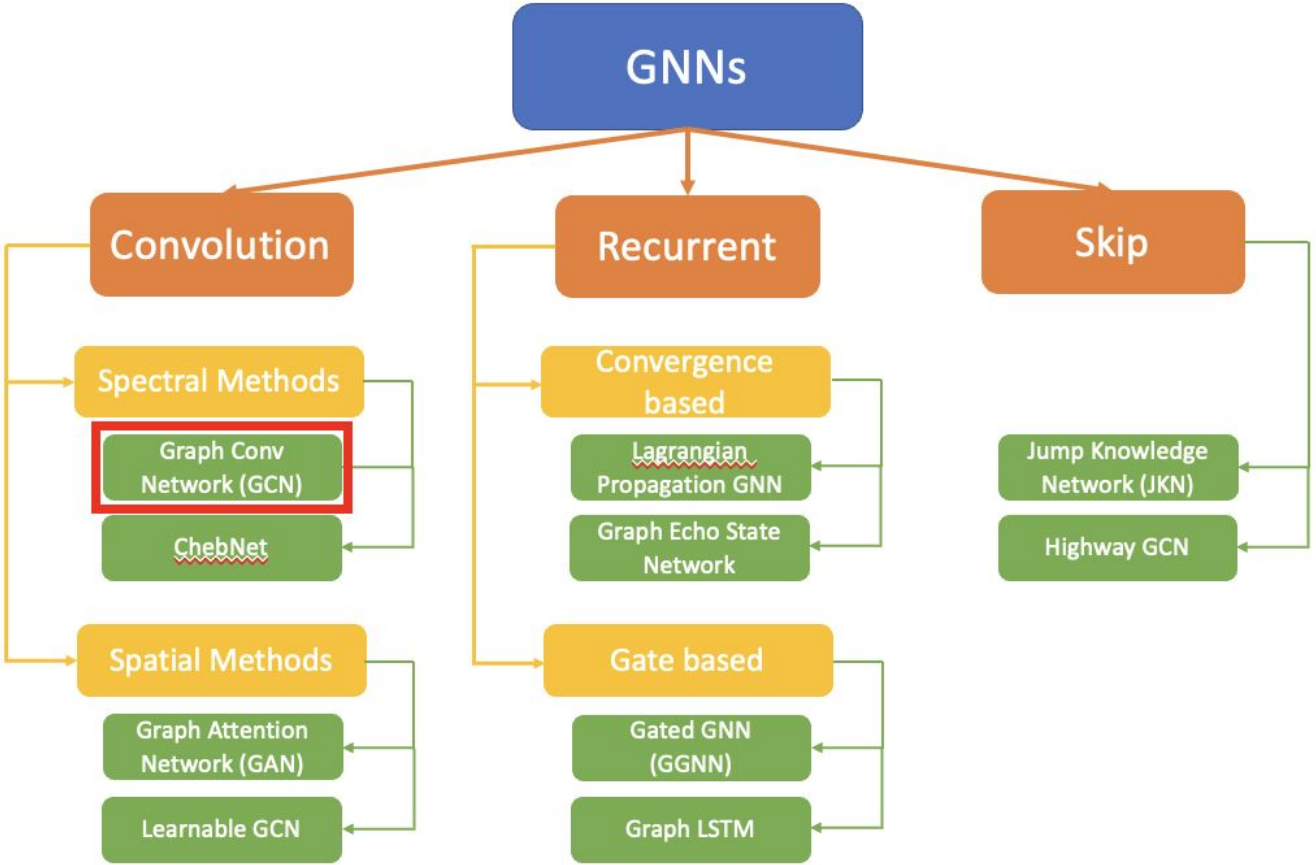
Operations in ChebNet

- For a graph G = (V,E,W); nodes, edges and edge weights respectively
- Graph Laplacian, L = D – W, where D is diagonal matrix such that $D_{ij} = \sum_j W_{ij}$
- The convolution operation is defined as, $y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = U g_\theta(\Lambda)U^T x.$
  here, L = $U\Lambda U^T$ (U is the Fourier basis) and $\theta \in R^n$ is a vector of Fourier coefficients
- Using a polynomial filter overcomes the limitations of nonparametric filters by localizing it in space and reducing learning complexity and $\theta \in R^k$ is a vector of polynomial coefficients $g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$
- The operation can thus be parametrized as a truncated expansion by using Chebyshev polynomial, $T_k(x)$ of order k, such that reduces the learning complexity of the filters from O(N) to O(K) $g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$
- Finally, a learnable operation can be defined as, $y_{s,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(L) x_{s,i} \in \mathbb{R}^n$

  where the $x_{s,i}$ are the input feature maps and the $F_{in} \times F_{out}$ vectors of Chebyshev coefficients $\theta_{i,j} \in R^K$ are the layer's trainable parameters

"Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" by Defferrard *et al,* in 2016

Division of GNNs



Convolutional GNNs-Continued

- Simpler definition of ChebNet introduced by Kipf *et al.* in "Semi-Supervised Classification With Graph Convolutional Networks" in 2017
- Here for a given operation using Chebyshev polynomial, simplifying assumption can be made

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x$$

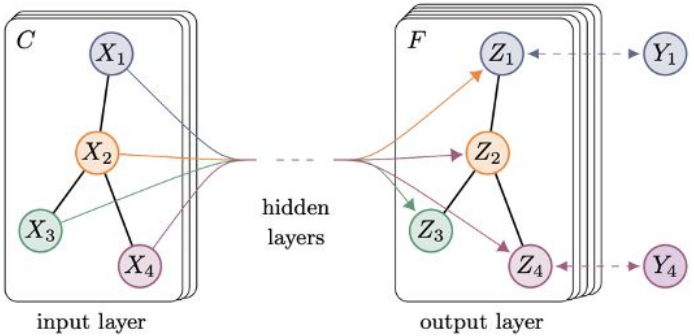
- If K = 1, or only First order Chebyshev polynomials are considered,

$$g_{\theta'} \star x \approx \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

  here, A is the weight matrix and D is diagonal matrix such that $D_{ij} = \sum_j A_{ij}$
- The forward operation for a two-layer model is defined as:

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A} X W^{(0)}\Big)\, W^{(1)}\Big)$$

  here $W^{(0)} \in R^{C \times H}$ is weight matrix for first layer with H feature maps and C input channels, $W^{(1)} \in R^{H \times F}$ is weight matrix for output layer with F output filters and $\hat{A} = D^{-1/2} A\, D^{-1/2}$



"Semi-Supervised Classification With Graph Convolutional Networks" by Thomas N. Kipf *et al,* in 2017.

Summary

- Bruna et al presented the first spectral method of performing convolution on a given graph
- Defferrard et al proposed expressing the nonparametric filters of convolution as Chebyshev Polynomials
- Kipf et al reworked the Graph Convolution methodology by simplifying concepts of ChebNet

- Even more networks designs possible by changing the operation on graphs (eg., recurrent, skip)
- GNNs outperform conventional deep learning methods in some tasks
- GNNs are a rapidly developing field!