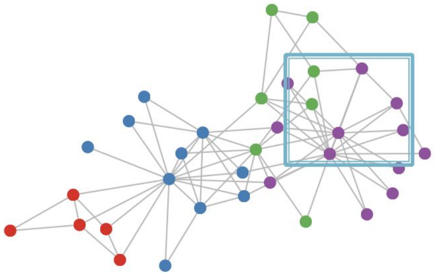


Graph Neural Network Intro

"I checked it very thoroughly, said the computer, and that quite definitely is the answer. I think the problem, to be quite honest with you, is that you've never actually known what the question is."

- Douglas Adams, The Hitchhiker's Guide to the Galaxy (1979)

Jay Urbain, Ph.D.

Electrical Engineering and Computer Science Department

Milwaukee School of Engineering

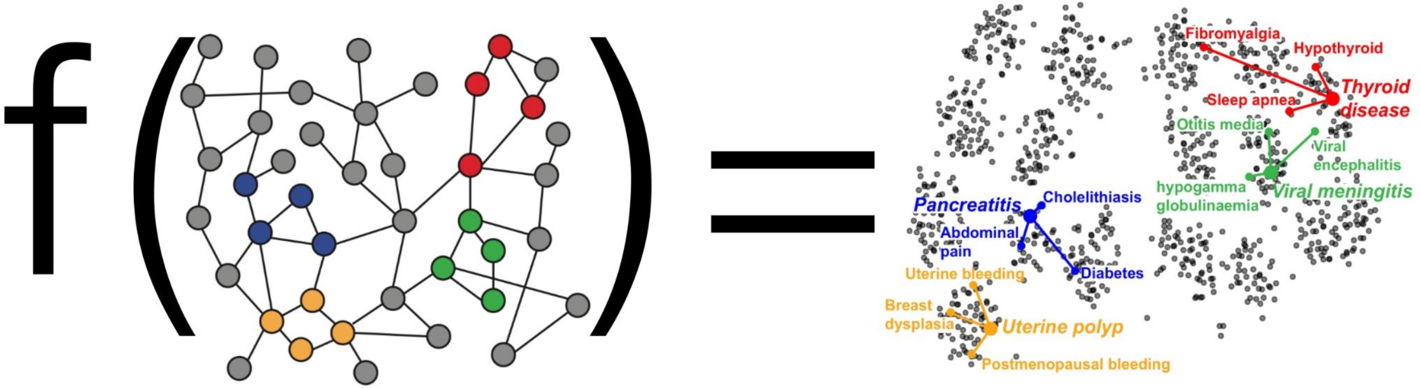
Credits:

[Kipf and Welling, ICLR 2017]

Review: Node Embeddings

Idea: Map nodes to a *d-dimensional* embeddings such that similar nodes in the graph are embedded close together.

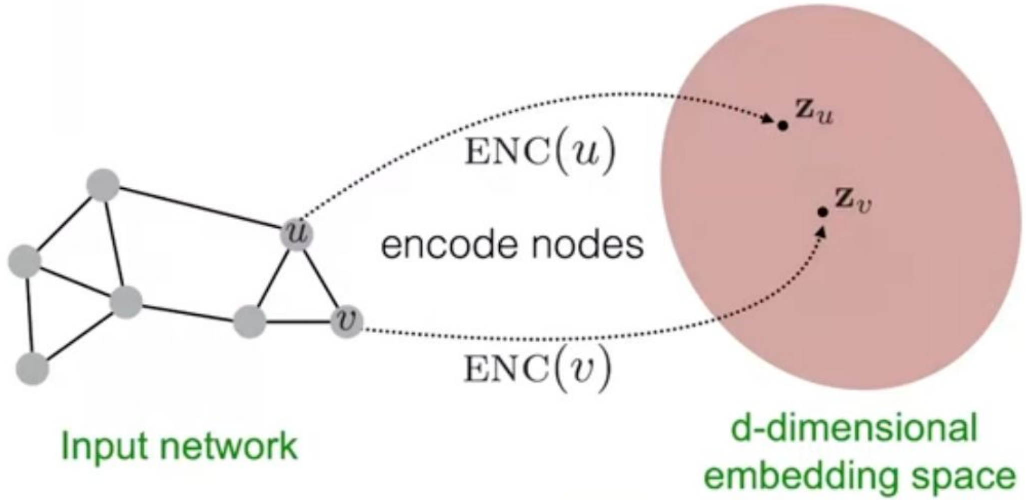
- Need to learn a mapping function ***f*** to perform the embedding.



Review: Encoder-Decoder Framework

Encoder-Decoder:

- Node Embeddings Goal: ***similarity(u,v)*** $\approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$

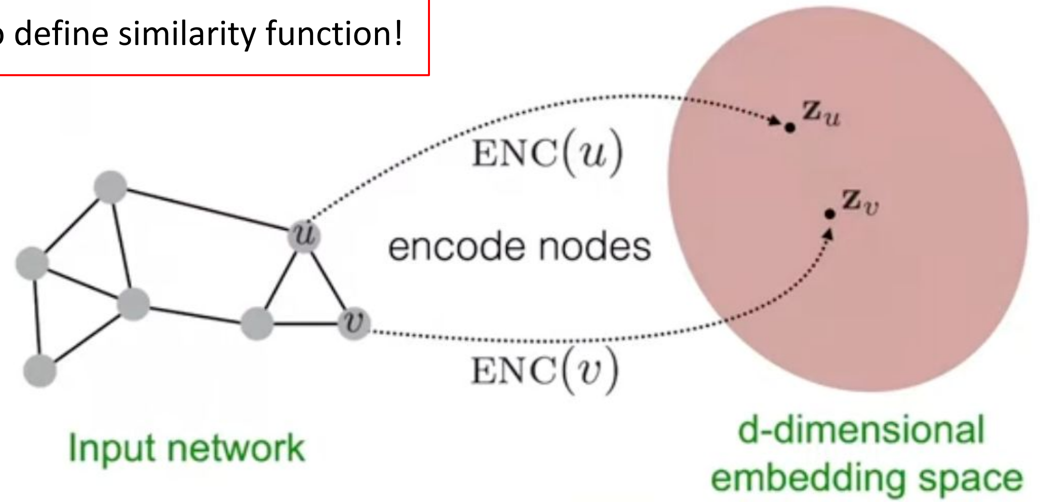


Encoder-Decoder Framework

Encoder-Decoder:

- Node Embeddings Goal: ***similarity(u,v)*** $\approx \mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$

Need to define similarity function!



Two key components

Encoder: Maps each node ***v*** to a low-dimensional embedding z_v.

$$\mathrm{ENC}(v) = \boxed{\mathbf{z}_v}$$

Similarity function: Specifies how the relationships in embedding space map to the relationships in the original network.

$$\mathrm{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$$

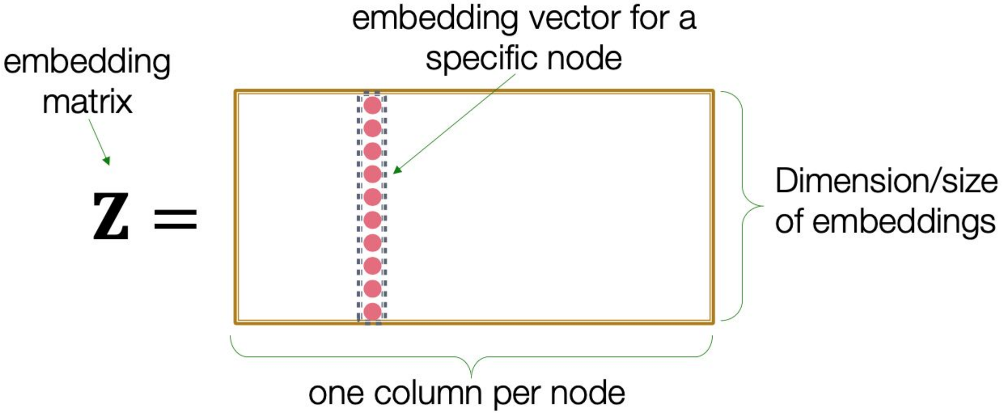
DEC

Similarity between ***u*** and ***v*** in original network

Dot product between node embeddings

Shallow Encoding

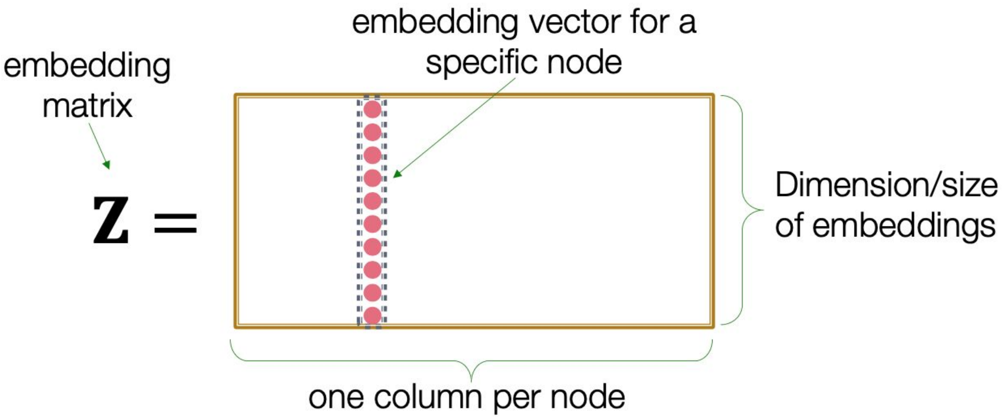
Simplest encoding approach: Encoder is just an embedding-lookup. Directly learning the embedding of each node.



Shallow Encoding

Simplest encoding approach: Encoder is just an embedding-lookup.

Directly learning the embedding of each node.



Need to learn the embedding of each node.

Shallow Encoders

Limitations of shallow embedding methods?

Bueller? Ferris Bueller?

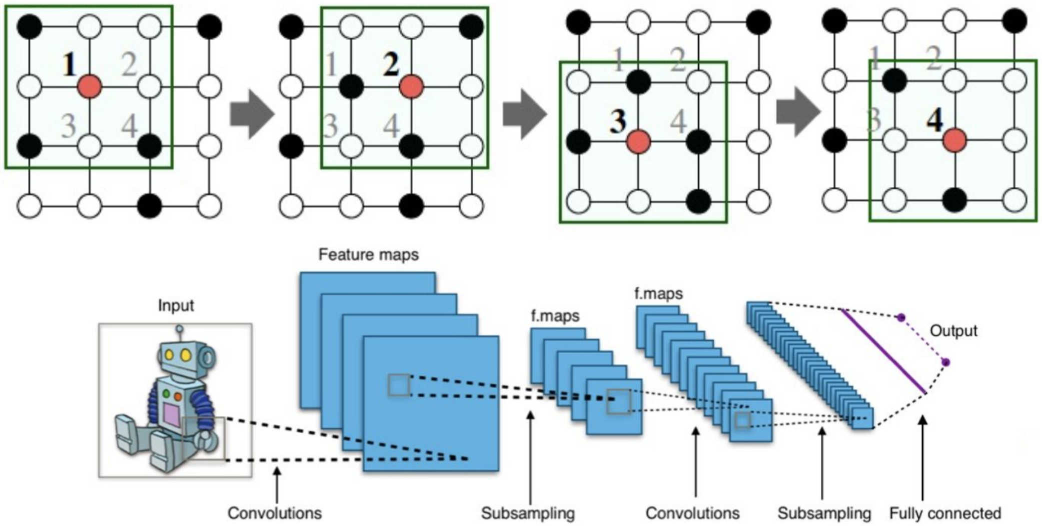
Shallow Encoders

Limitations of shallow embedding methods:

- $O(|V|)$ parameters are needed:
  - No sharing of parameters between nodes.
  - Every node has its own unique embedding.
- Inherently "transductive":
  - Cannot generate embeddings for nodes that are not seen during training.
  - Must learn embedding for each node.
  - Want inductive learning.
- Does not incorporate node features:
  - Nodes in many graphs have features that we can and should leverage.

Idea convolution:

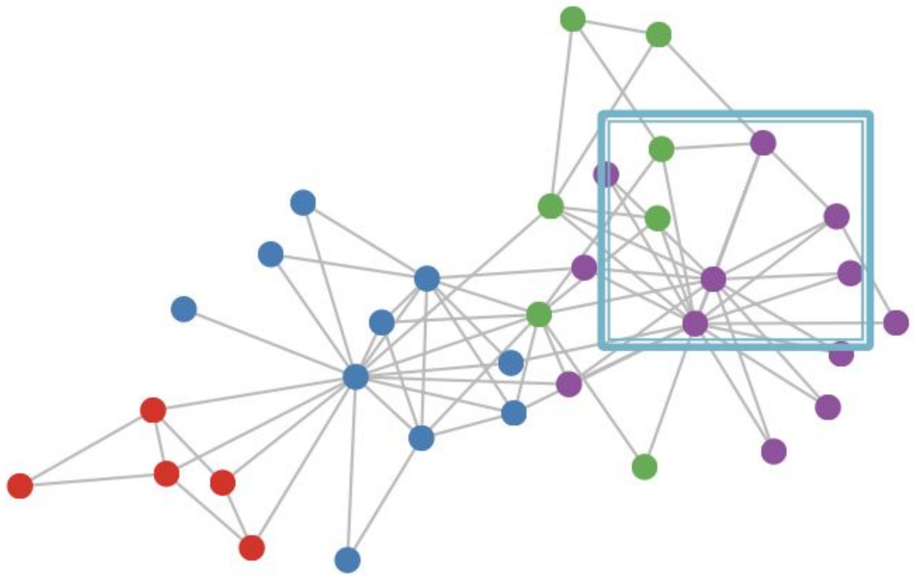
CNN on an image:



- Goal is to generalize convolutions beyond simple lattices
- Leverage node features/attributes (e.g., text, images)

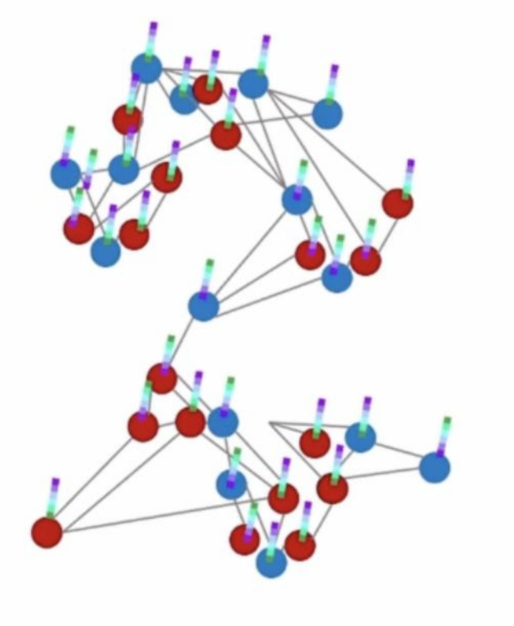
Real-World Graphs

- There is no fixed notion of locality or sliding window on the graph
- Graph is permutation invariant



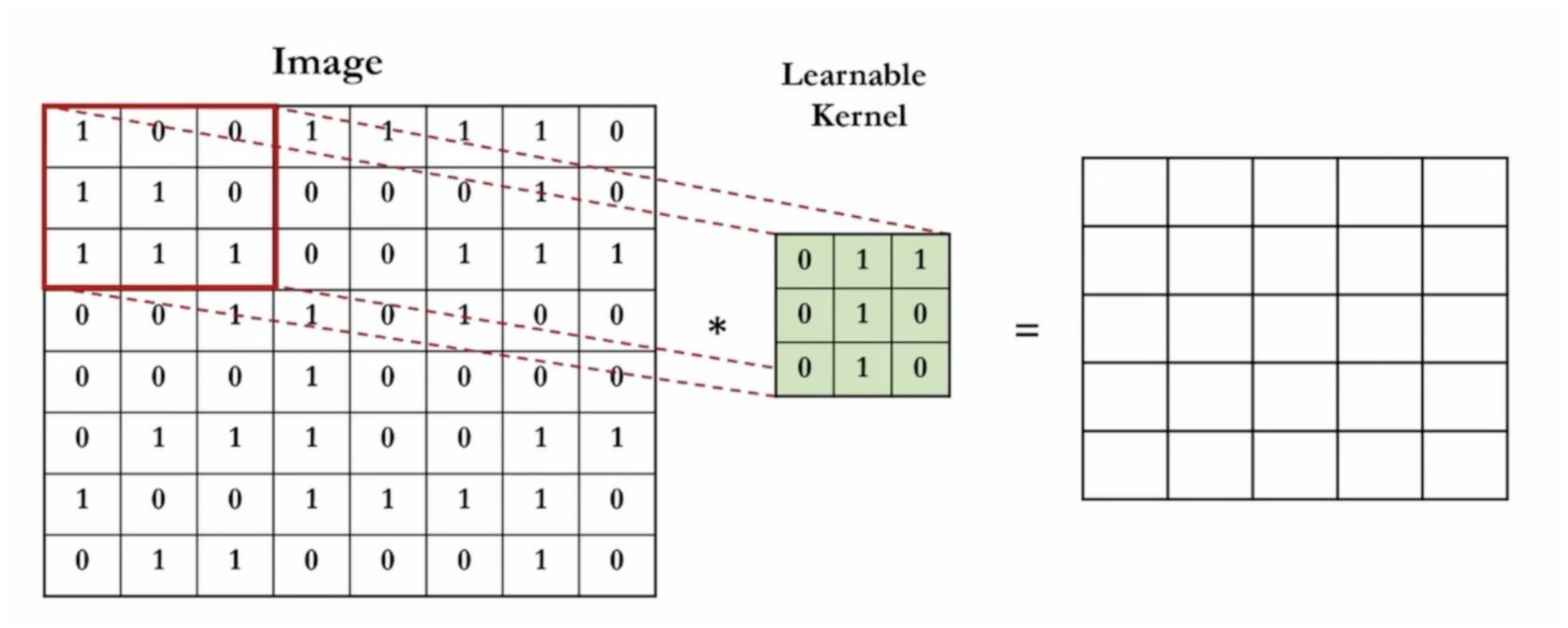
Generalizing Convolution

How to generalize CNN to GCN?

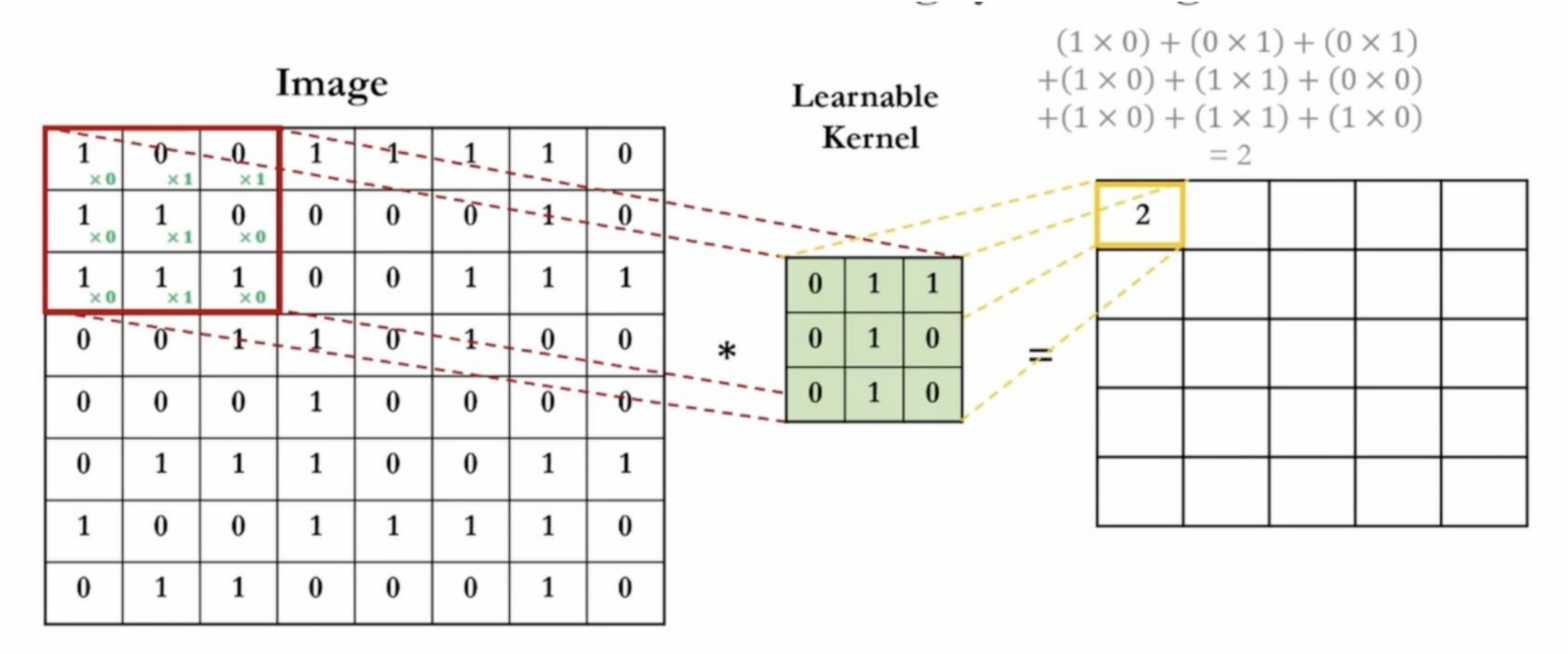


Review

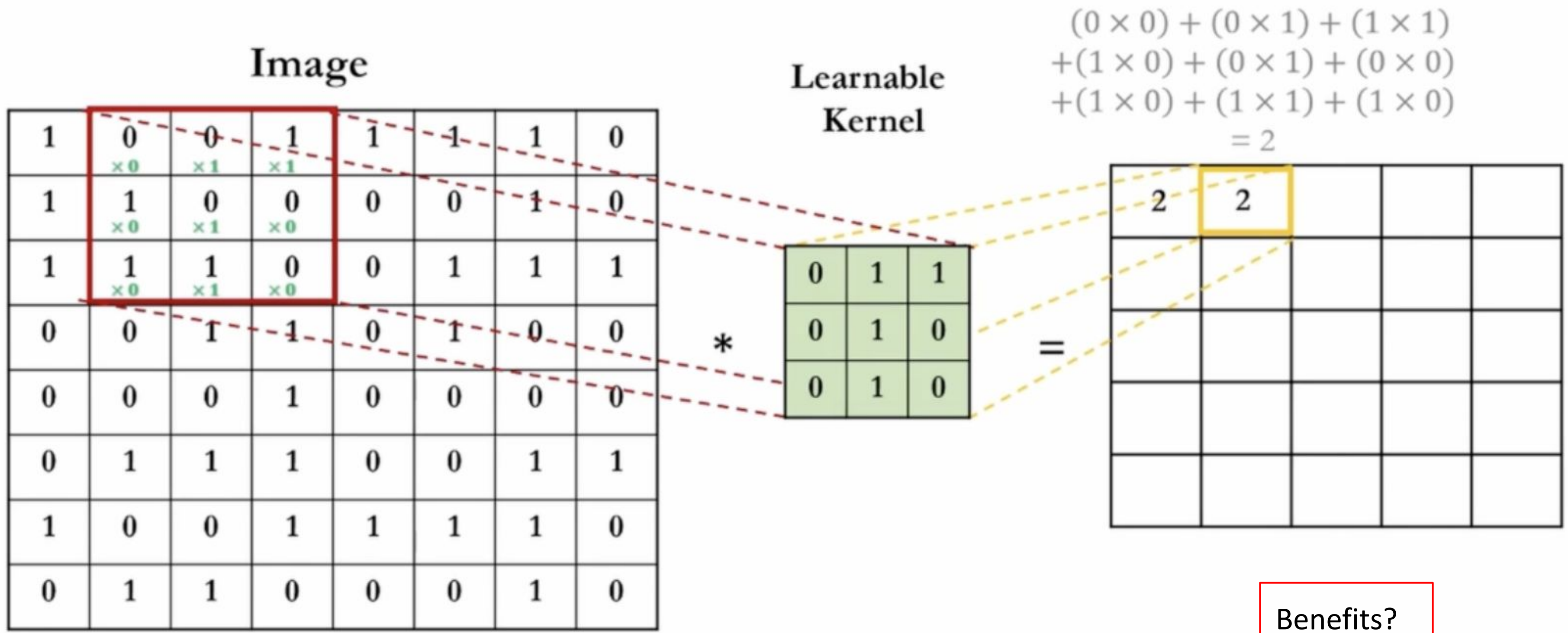
Convolution in 2D



Convolution in 2D

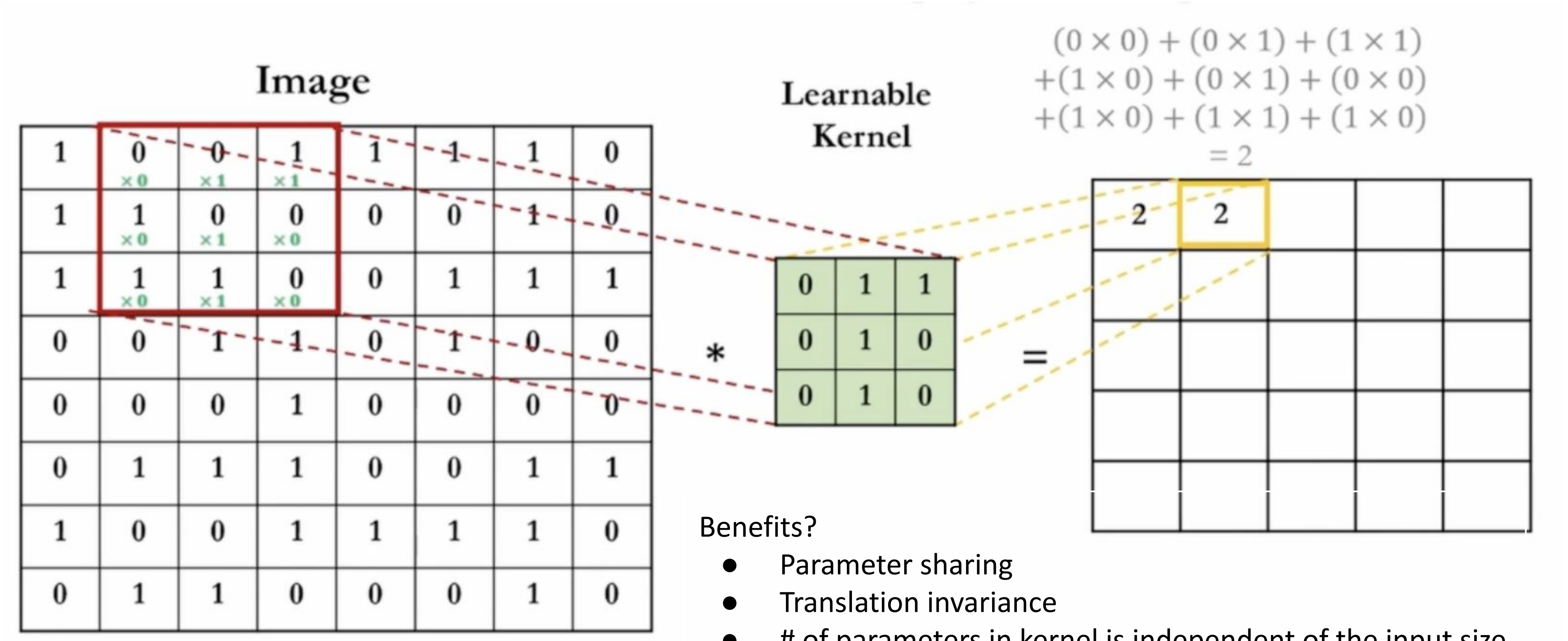


Convolution in 2D



Benefits?

Convolution in 2D



Benefits?

- Parameter sharing
- Translation invariance
- # of parameters in kernel is independent of the input size

Deep Learning for Graphs

Deep Graph Encoders

Deep learning methods based on graph neural networks (GNNs):

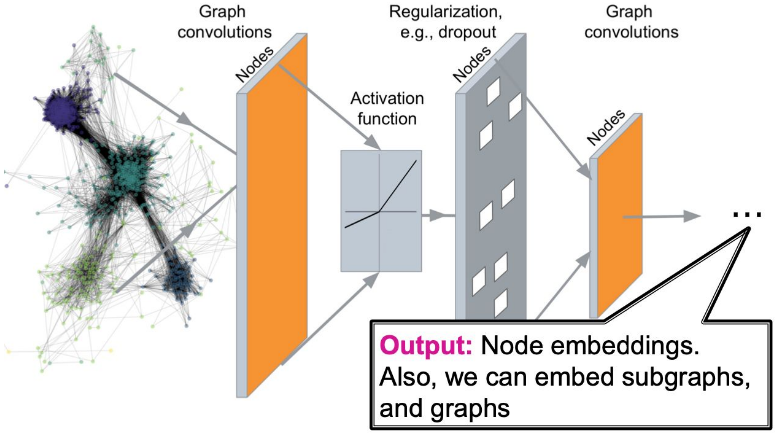
- ***ENC(v)*** - multiple layers of non-linear transformations based on graph structure.
- Deep encoders can be combined with node similarity functions like PageRank, or learn end-to-end.
- Can train for graph prediction tasks.

Deep Graph Encoders

Goal: Train end-to-end.

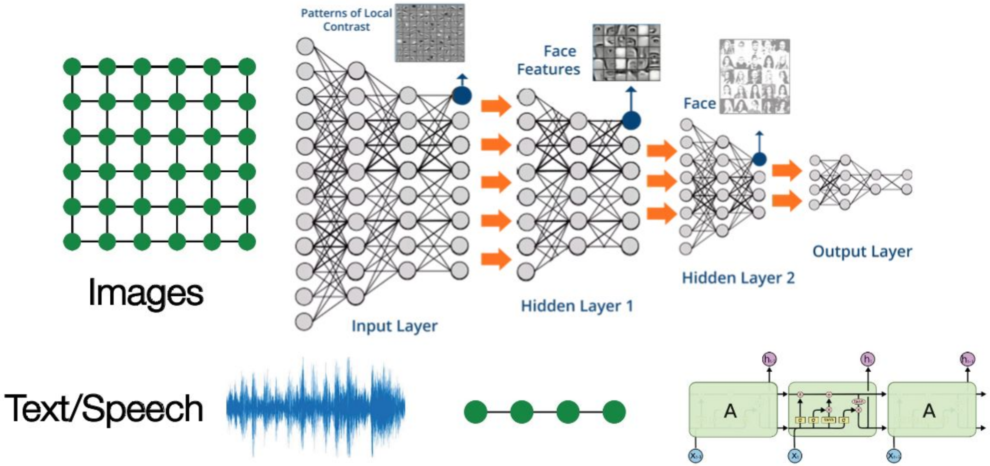
Tasks to solve:

- Node classification - Predict a type of a given node.
- Link prediction - Predict whether two nodes are linked
- Community detection - Identify densely linked clusters of nodes
- Network similarity - How similar are two networks or sub-networks



Machine Learning

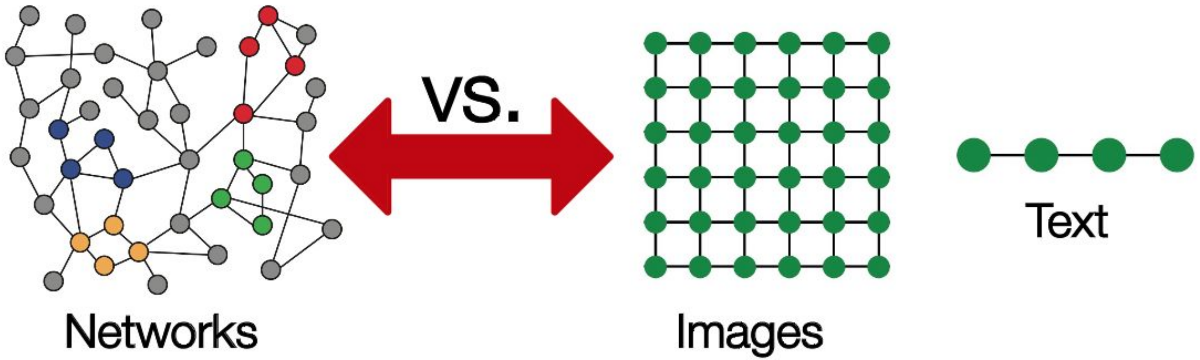
- Modern deep learning methods are designed for simple sequences and grids.
- Assume IID data.
- How to apply to more complex data types like graphs?



Why is ML on graphs hard?

Networks are far more complex.

- Arbitrary size and complex topological structure - no spatial locality like grids.
- No fixed ordering, no left or right, no direction, no starting reference point.
- Often dynamic and have multimodal features.



GML Setup

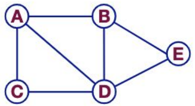
Assume we have a graph $\boldsymbol{G}$:

- V is the vertex set
- $\boldsymbol{A}$ is the adjacency matrix (assume binary)
- $\boldsymbol{X} \in \mathbb{R}^{m \times |V|}$ is a matrix of node features
- v: a node in V; Nv: the set of neighbors of v.
- Node features:
  - Social networks: User profile, User image
  - Biological networks: Gene expression profiles, gene functional information
  - When there is no node feature in the graph dataset:
    - Indicator vectors (one-hot encoding of a node)
    - Vector of constant 1: [1, 1, ..., 1]

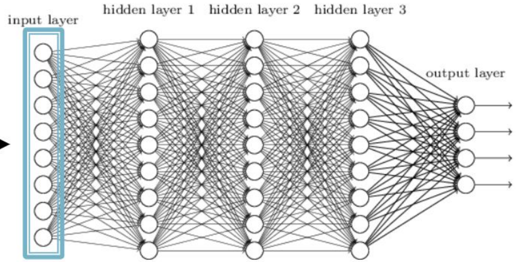
Naive Approach

Join adjacency matrix and features.

Feed them into a deep neural network:



$$
\begin{array}{c|ccccccc}
 & A & B & C & D & E & \text{Feat} & \\
\hline
A & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
B & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
C & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
D & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
E & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}
$$

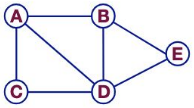


Issues?

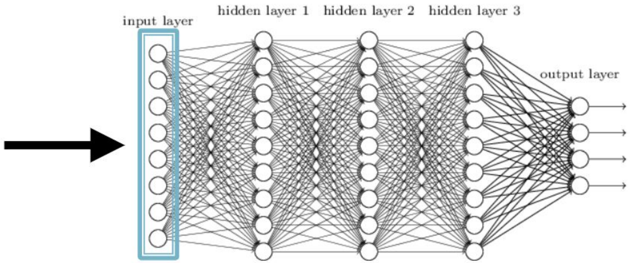
Naive Approach

Join adjacency matrix and features.

Feed them into a deep neural network:



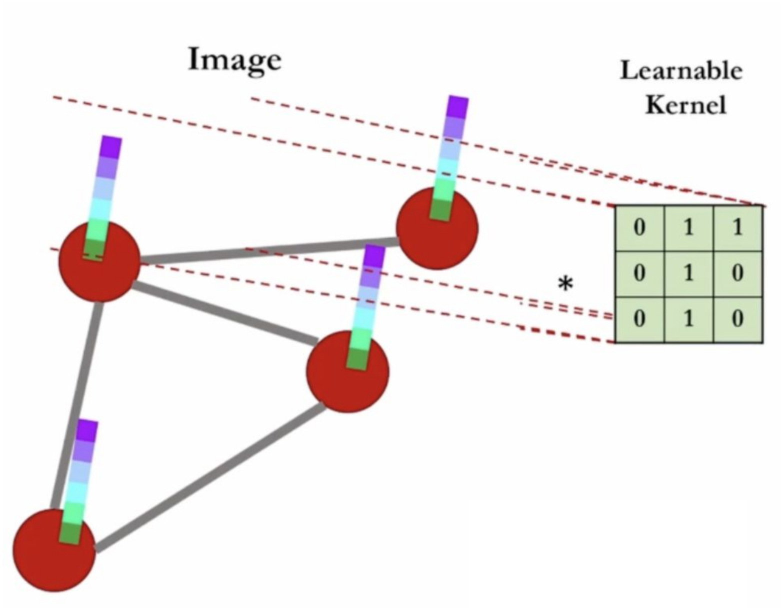
$$
\begin{array}{c|ccccc|cc}
 & A & B & C & D & E & \multicolumn{2}{c}{Feat} \\
\hline
A & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
B & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
C & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
D & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
E & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}
$$



Issues:

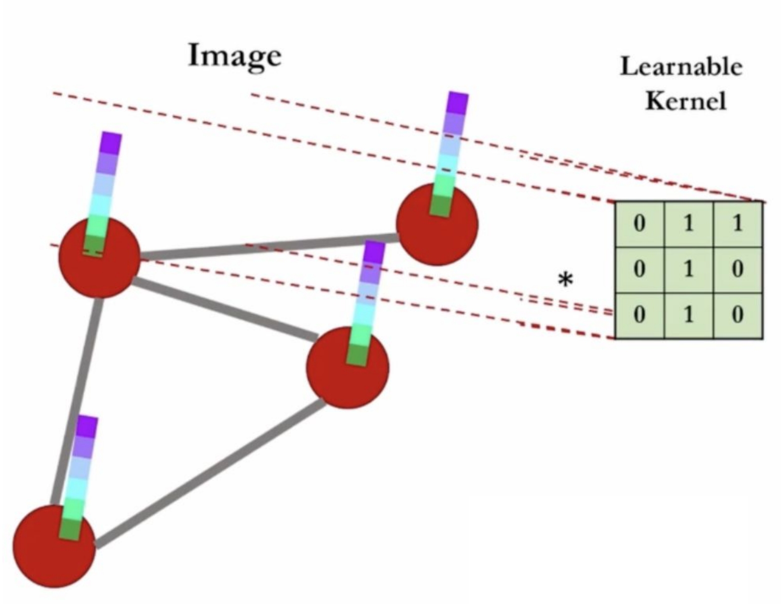
- ***O|V|*** parameters - not scalable!
- Not applicable to graphs of different sizes
- Sensitive to node ordering

Challenges in generalizing convolution to graph



Challenges in applying convolution to graph?

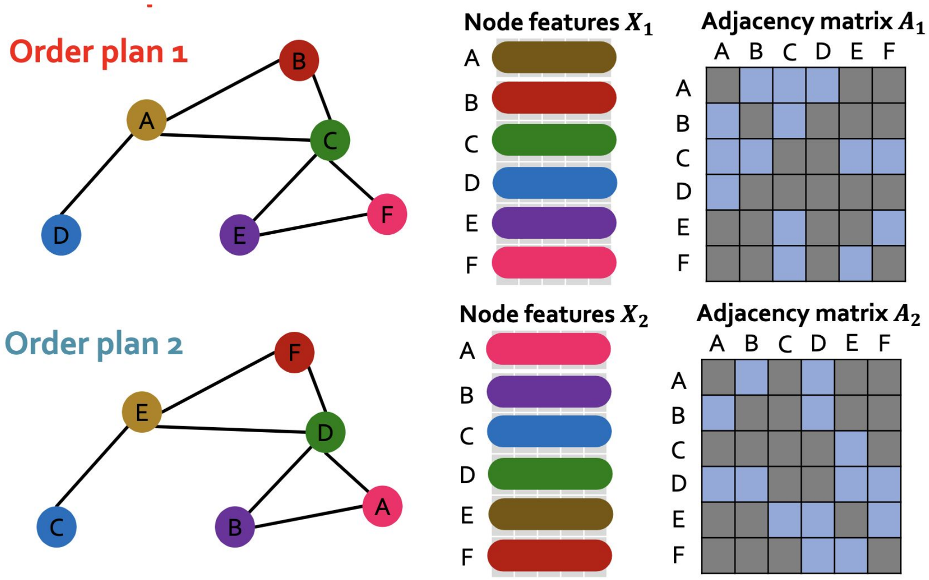
Challenges in generalizing convolution to graph



- Number of neighbor nodes changes
- Distance between node changes
- Number of attributes can vary (features)
- We may have a heterogeneous graph - different nodes with different meaning and attributes
- Node ordering can change

Permutation Invariance

- Graph does not have a canonical order of the nodes.



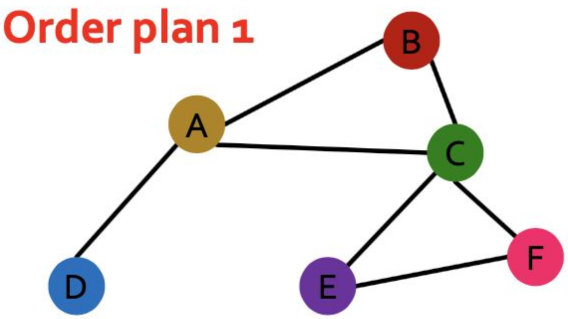
Graph and node representations should be the same for Order plan 1 and 2

Permutation Invariance

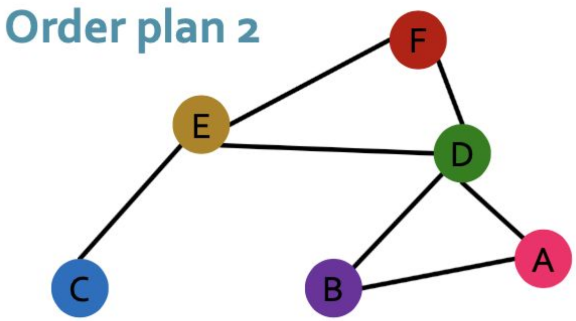
What does it mean by “graph representation is same for two order plans”?

Learn a function f that maps a graph $G = (A, X)$ to a vector R^d then

$$f(A_1, X_1) = f(A_2, X_2)$$

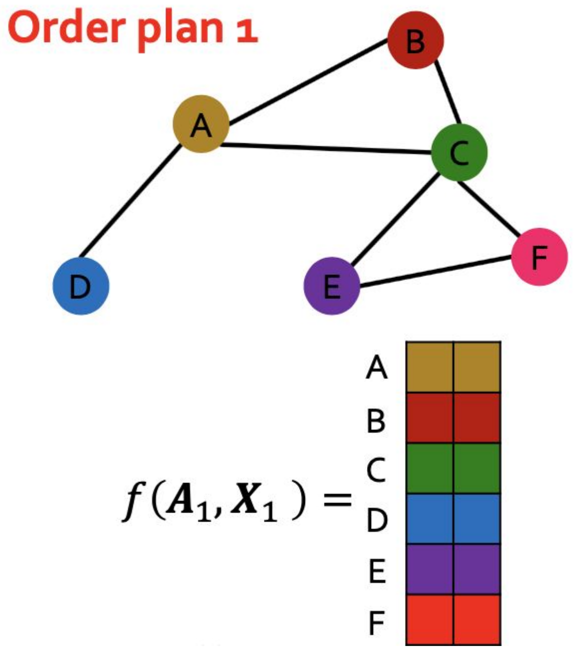


For two order plans, output of f should be the same.
A - adjacency matrix
X - node similarity matrix

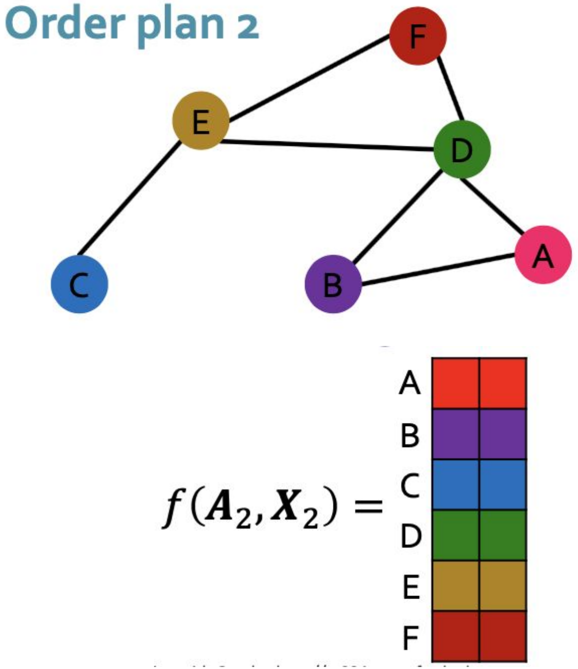


Permutation Invariance

- Similarly for node representation: Learn a function f that maps nodes of G to a matrix $\boldsymbol{R}$.

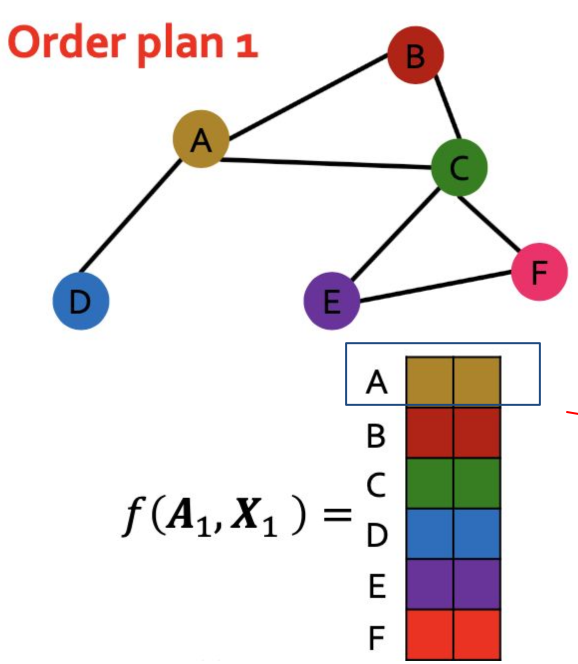


For two order plans, output of $\boldsymbol{f}$ should be the same.
A - adjacency matrix
X - node similarity matrix



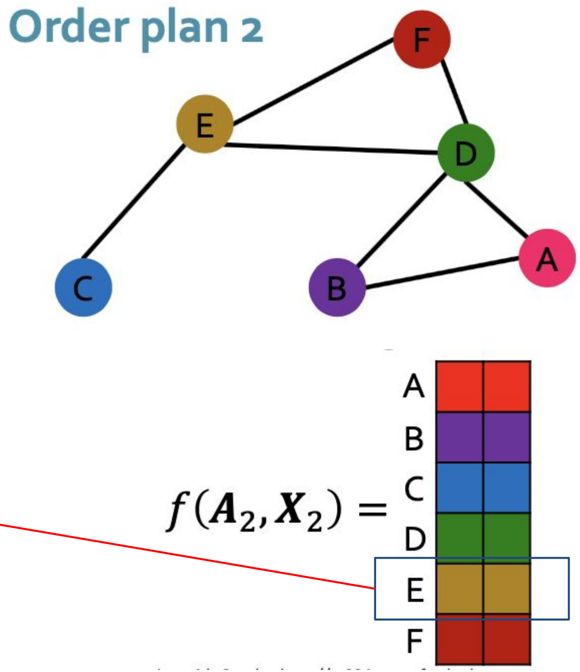
Permutation (ordering) Invariance

- Similarly for node representation: Learn a function f that maps nodes of G to a matrix $\boldsymbol{R}$.



For two order plans, the vector of node at the same position is the same!

Representation of brown node A, and node E



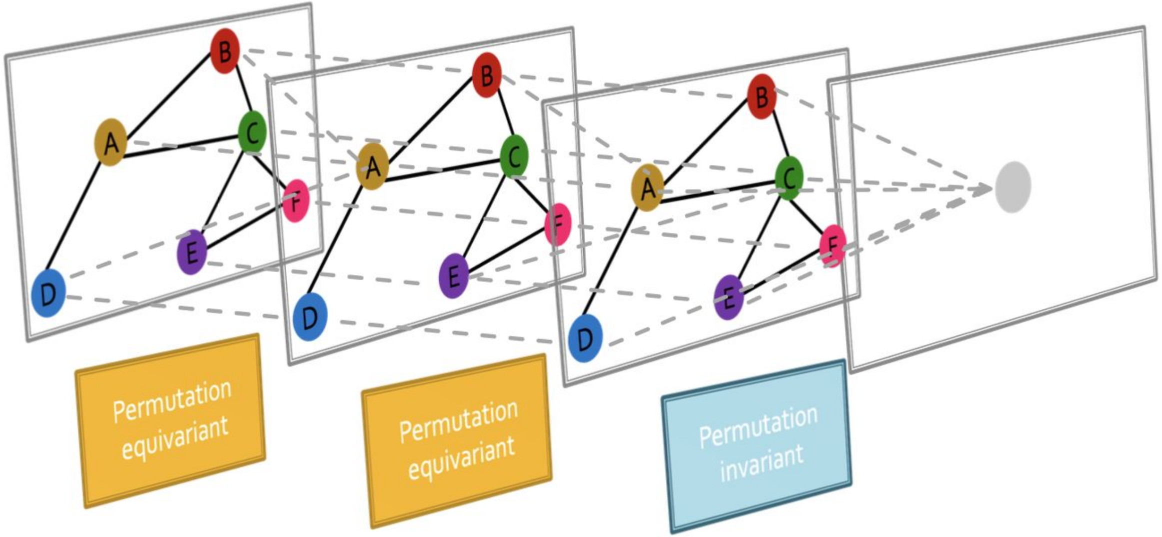
Permutation Equivariance

For node representation

- Learn a function f that maps a graph $G = (A, X)$ to a matrix R.
  - graph has m nodes, each row is the embedding of a node.
- Similarly, if this property holds for any pair of order plan i and j, we say f is a permutation equivariant function.
- All nodes learn their embeddings from their connections and any function applied to any node will result in the same value independent of noder ordering!

Graph Neural Network Overview

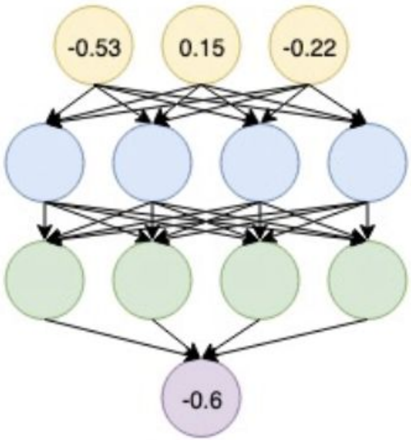
- Graph neural networks consist of multiple permutation equivariant/invariant functions.



Graph Neural Network Overview

Are other neural network architectures, e.g., MLPs, permutation invariant / equivariant?

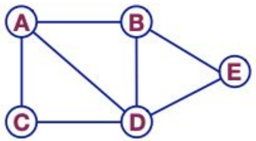
- Nop! - switching the order of the input leads to different outputs.



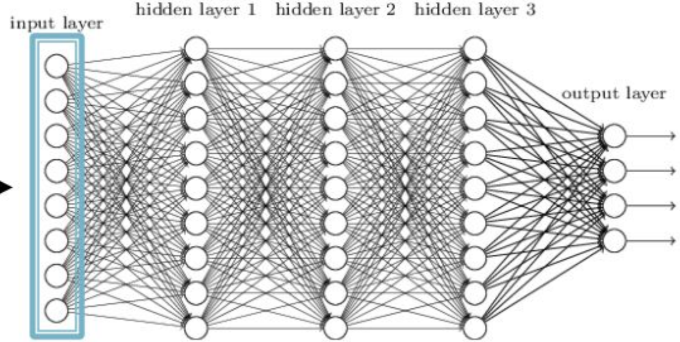
Graph Neural Network Overview

Are other neural network architectures, e.g., MLPs, permutation invariant / equivariant?

- Nop! - switching the order of the input leads to different outputs.



$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{cccccccc} A & B & C & D & E & & \text{Feat} & \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & & 1 & 0 \\ \boxed{1 & 0 & 0 & 1 & 1 & & 0 & 0} \\ 1 & 0 & 0 & 1 & 0 & & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & & 1 & 0 \end{pmatrix} \end{array}$$

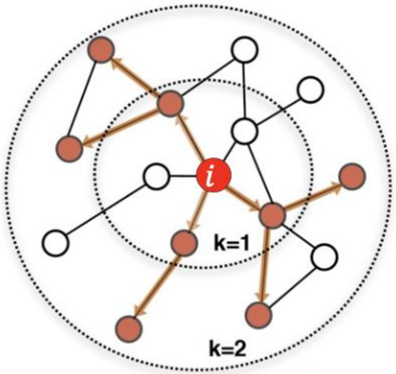


Should we give up?

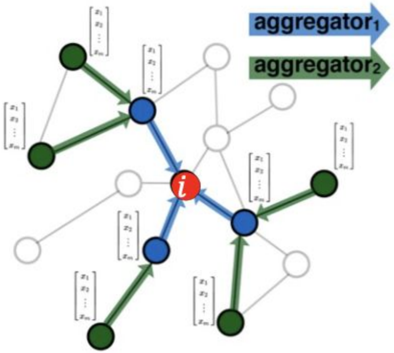
Graph Convolutional Networks

- Idea: Node’s neighborhood defines a computation graph.
- Think message passing!
- Learn how to propagate information across the graph to compute node features.

[Kipf and Welling, ICLR 2017]



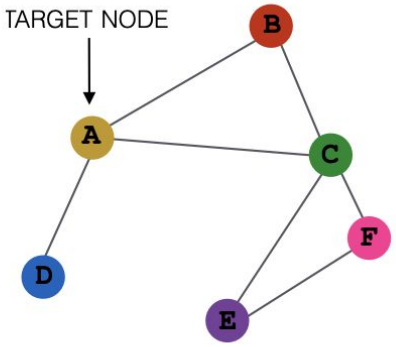
Determine node computation graph



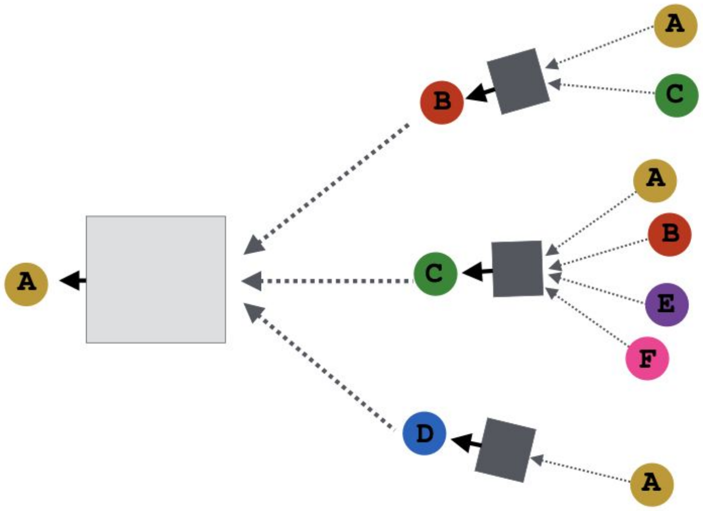
Propagate and transform information

Graph Convolutional Networks

- Key idea: Generate node embeddings based on local network neighborhoods



INPUT GRAPH

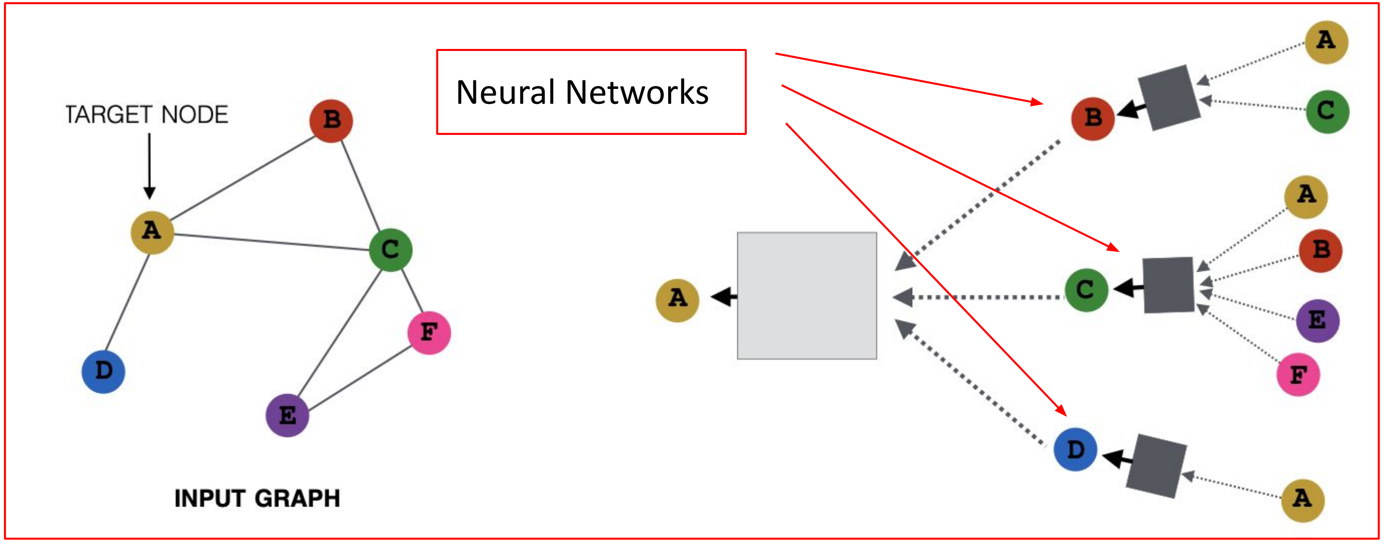


[Kipf and Welling, ICLR 2017]

Graph Convolutional Networks

Aggregate Neighbors.

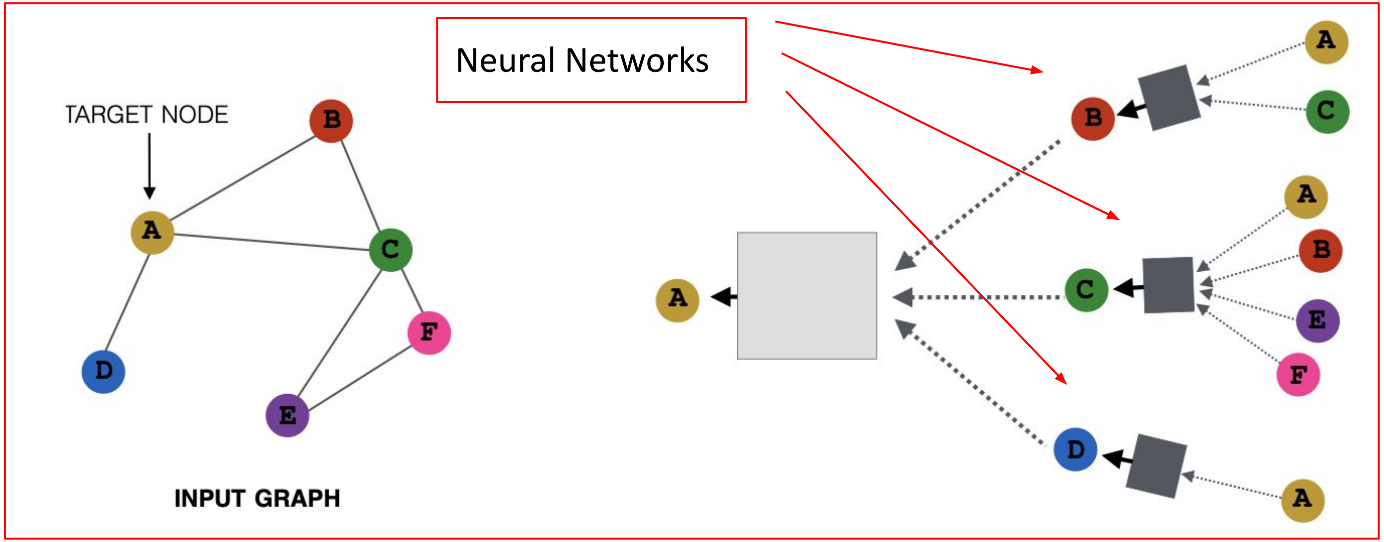
- Idea: Nodes aggregate information from their neighbors using neural networks



Graph Convolutional Networks

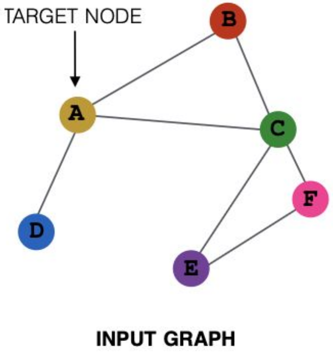
Aggregate Neighbors.

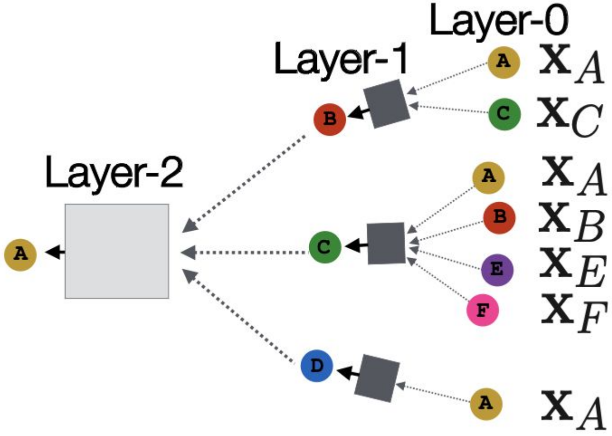
- Idea: Nodes aggregate information from their neighbors using neural networks.
- Every node defines a computation graph.



Deep Model: Many layers

- Model can be of arbitrary depth:
- Nodes have embeddings at each layer
- ***Layer-0*** embedding of node v is its input feature, x_v
- ***Layer-k*** embedding gets information from nodes that are k hops away.





Neighborhood Aggregation

- Key distinctions in algorithms are in how different approaches aggregate information across the layers.
- Basic approach: Average messages from neighbors.

Initial 0-th layer embeddings are equal to node features

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

embedding of v at layer k

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

Total number of layers

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

Average of neighbor's previous layer embeddings

Embedding after L layers of neighborhood aggregation

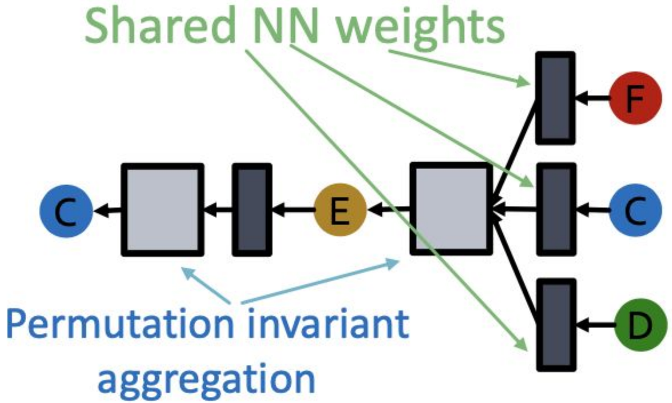
Non-linearity (e.g., ReLU)

Notice summation is a permutation invariant pooling/aggregation.

Equivariant Property

Message passing and neighbor aggregation in graph convolution networks is permutation equivariant.

The target node has the same computation graph for different order plans.



Training

Feed embedding into any loss function and run SGD to train the weight parameters.

$\boldsymbol{B_k}$ - weight matrix for transforming hidden vector of self

$\boldsymbol{W_k}$ - weight matrix for neighborhood aggregation

Trainable weight matrices
(i.e., what we learn)

$$\mathrm{h}_v^{(0)} = \mathrm{x}_v$$

$$\mathrm{h}_v^{(k+1)} = \sigma\left(\boxed{\mathrm{W}_k} \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(k)}}{|\mathrm{N}(v)|} + \boxed{\mathrm{B}_k} \mathrm{h}_v^{(k)}\right), \forall k \in \{0..K-1\}$$

$$\mathrm{z}_v = \mathrm{h}_v^{(K)}$$

Final node embedding

Training

Supervised - minimize loss.

$$\mathcal{L} = \sum_{z_u, z_v} \mathrm{CE}(y_{u,v}, \mathrm{DEC}(z_u, z_v))$$

Node similarity can be anything from earlier in the class:

- Random walks (PageRank, DeepWalk, node2vec)
- Matrix factorization
- Node proximity.

Unsupervised - no node labels, use graph structure (like pagerank)

Supervised Training

Supervised - minimize loss.

- Directly train the model for a supervised task (e.g., node classification)
- Use cross entropy loss

$$\mathcal{L} = \sum_{z_u, z_v} \mathrm{CE}(y_{u,v}, \mathrm{DEC}(z_u, z_v))$$

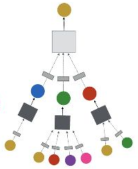
$$\mathcal{L} = \sum_{v \in V} \boxed{y_v} \log(\sigma(\boxed{\mathrm{z}_v^{\mathrm{T}}}\boxed{\theta})) + (1 - \boxed{y_v}) \log(1 - \sigma(\boxed{\mathrm{z}_v^{\mathrm{T}}}\boxed{\theta}))$$

Encoder output:
node embedding

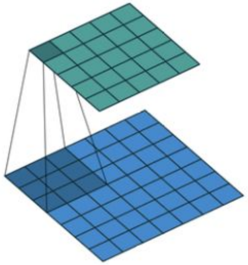
Classification
weights

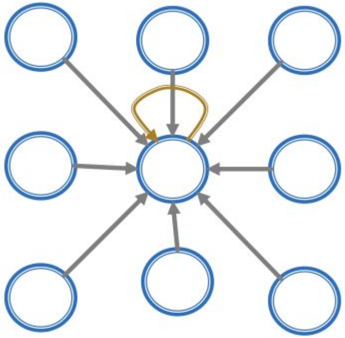
Node class
label

Safe or toxic drug?



Generalize CNN as GNN

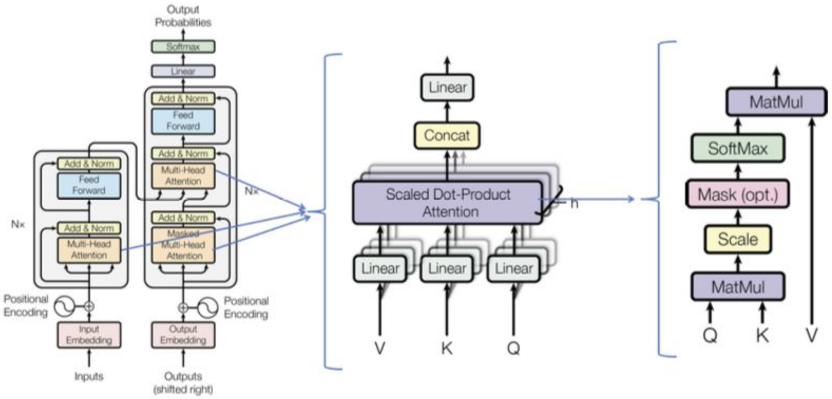




$$\mathrm{h}_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \dots, L-1\}$$

Generalize Transformer as GNN

- Transformer is one of the most popular architectures that achieves great performance in many sequence modeling tasks.
- Key component: self-attention
  – Input text as graph of connected words: Every token/word attends to all the other tokens/words via matrix calculation.



Deep Learning on Graphs: Topics

- Basics of neural networks
  - Loss, Optimization, Gradient, SGD, non-linearity, MLP
- Idea for Deep Learning for Graphs
  - Multiple layers of embedding transformation
  - At every layer, use the embedding at previous layer as the input
  - Aggregation of neighbors and self-embeddings
- Graph Convolutional Network
  - Mean aggregation; can be expressed in matrix form
- GNN is a general architecture
  - CNN and Transformer can be viewed as a special GNN