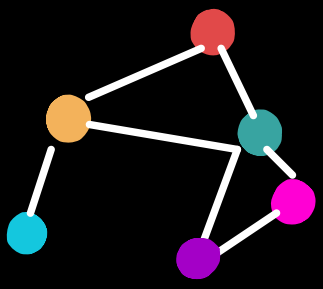
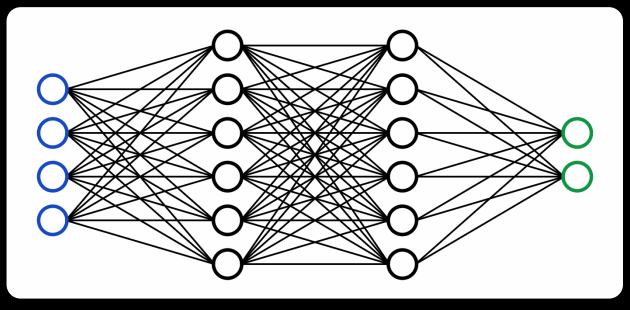
GraphT$_{10}$

Graph

Neural Networks



geometric deep learning

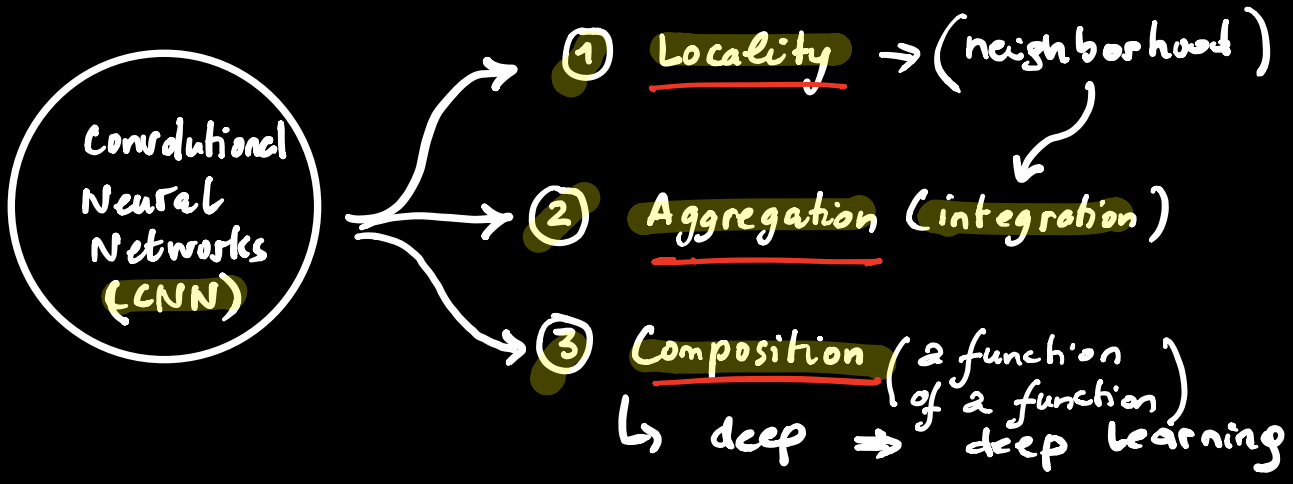


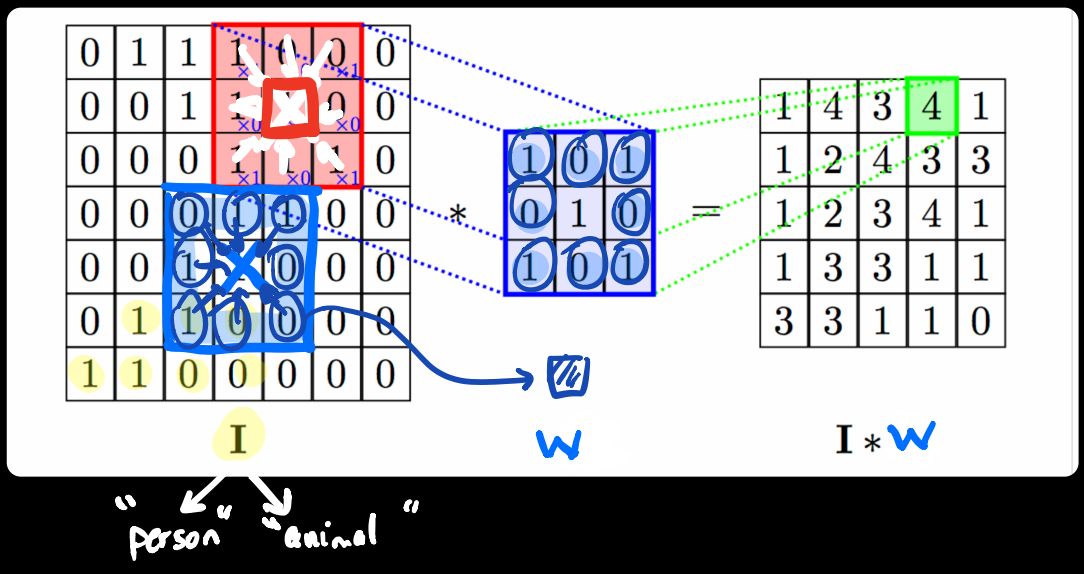
hand-crafted features

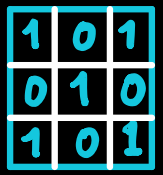
learn feature representations

☺ RULE 1 : "Tell me who your neighbors are, I will tell you who you are!"

↳ Learn from your neighbors.



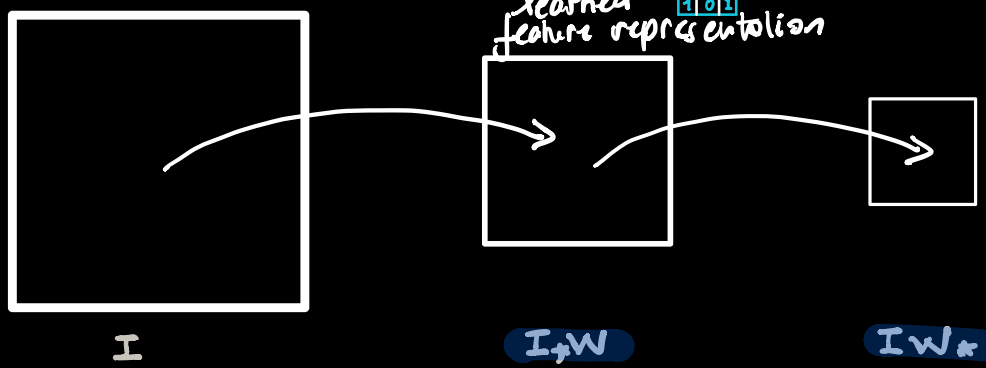




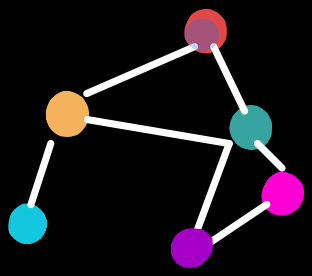
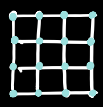
= filter = {set of weights}

① locality

② aggregation



How to define these three properties for graphs?

$G =$  $\neq$  $= I$ 

- arbitrary size
- complex topology (no spatial locality like grids)
- graphs are non-Euclidean objects
- No fixed node ordering

- grid like ⇒ spatial locality
- structured data in a Euclidean space
- Deep learning models are designed for simple sequences & grids.

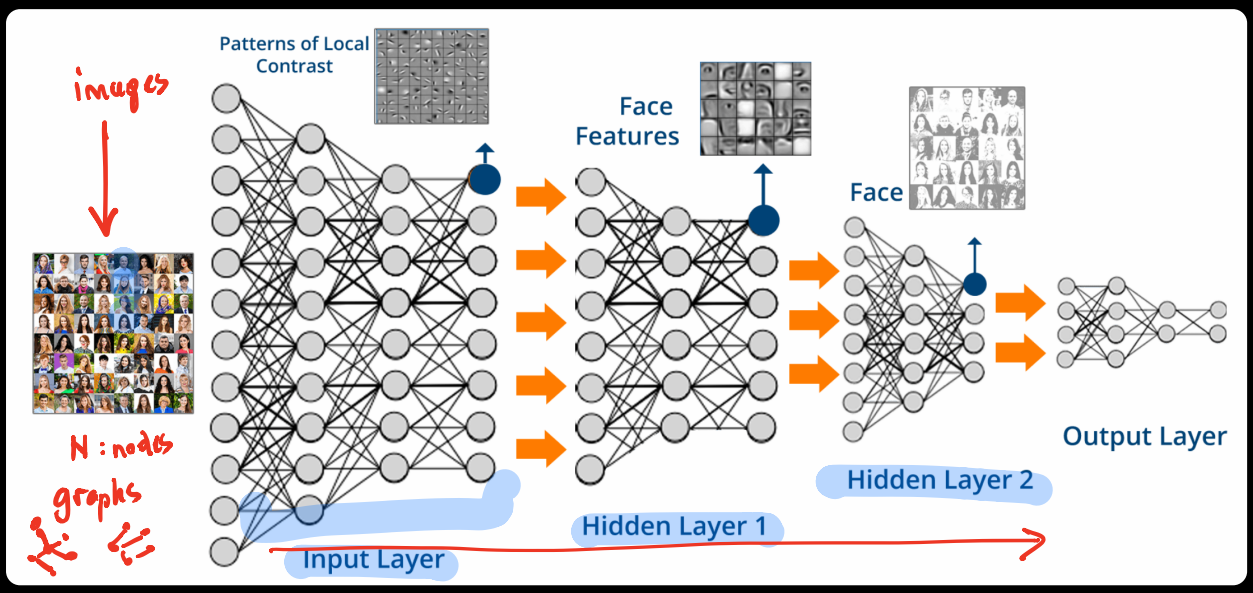
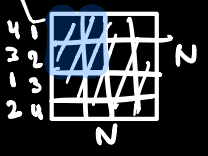
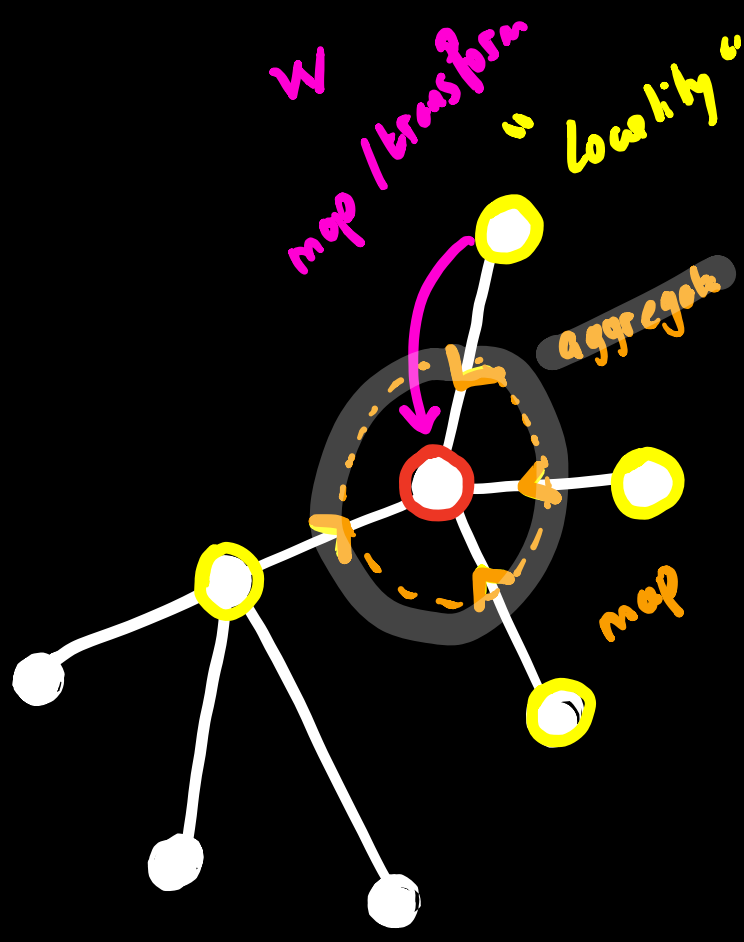


Image from: https://cdn.edureka.co/blog/wp-content/uploads/2017/05/Deep-Neural-Network-What-is-Deep-Learning-Edureka.png

(1) What is the most naive approach to train this CNN on graphs?

$G \rightarrow A$  $\{A_1, \ldots, A_{tr}\}$

(2) What if the graphs have different sizes?

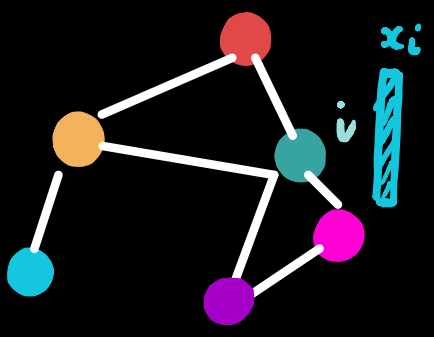

W
map / transform
"locality"
aggregate
map

☺ HINT : Think about how to define the following tools on graphs.

Graph Neural Networks (GNN)

1. Locality (neighborhood) *
2. Aggregation (integration) *
3. Composition (a function of a function) (stacking layers) *

decorated / annotated graphs

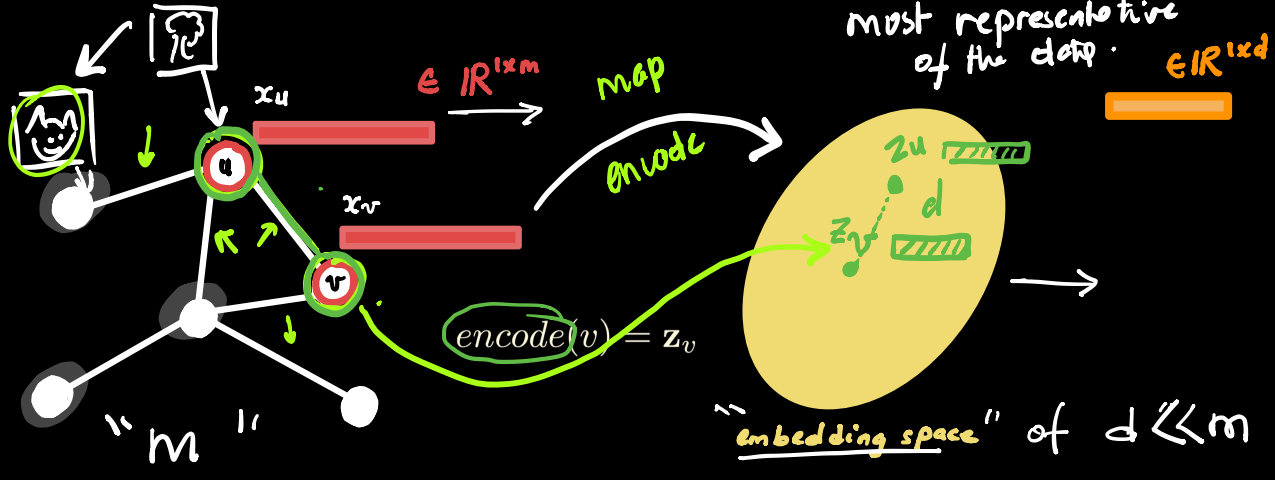


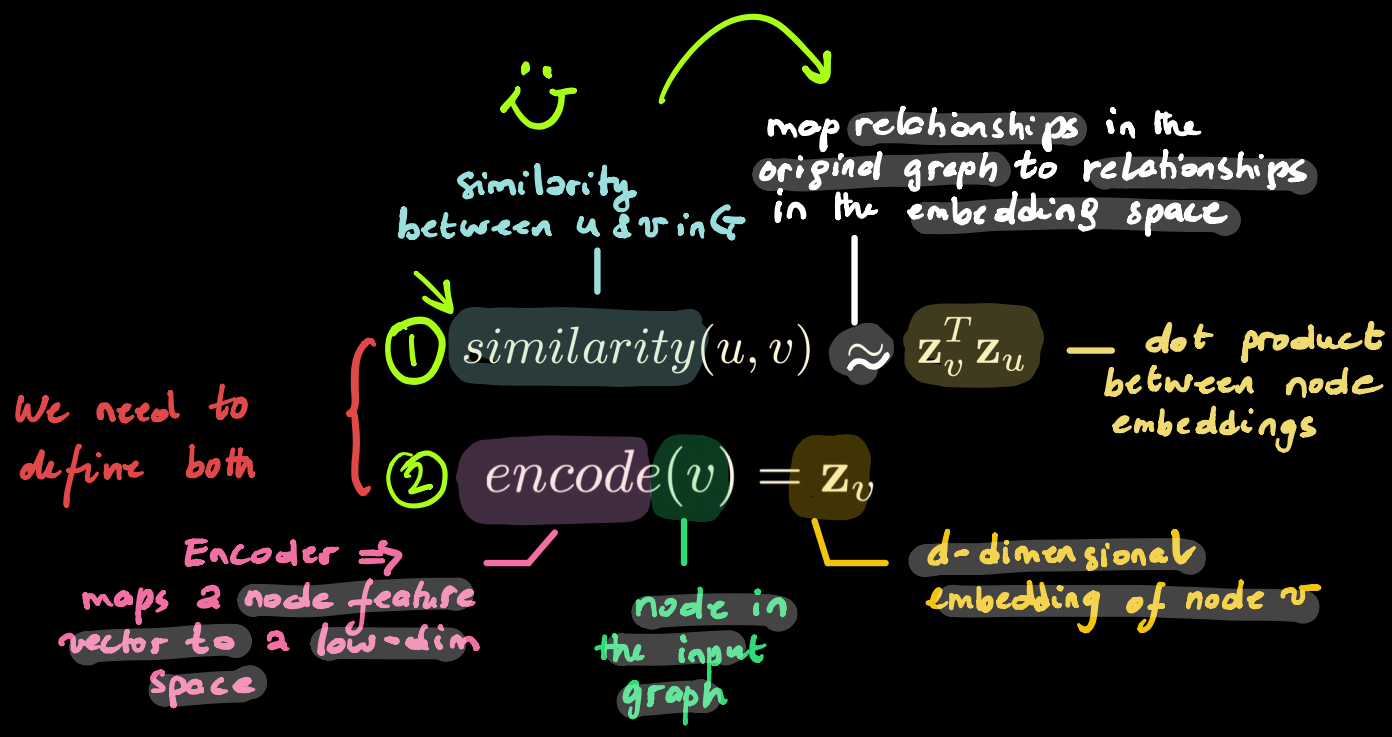
* Assume we have a graph $G = (V, E)$ (V: vertex set, E: edge set)
* A is the adjacency matrix of G.
* X is the node feature matrix.

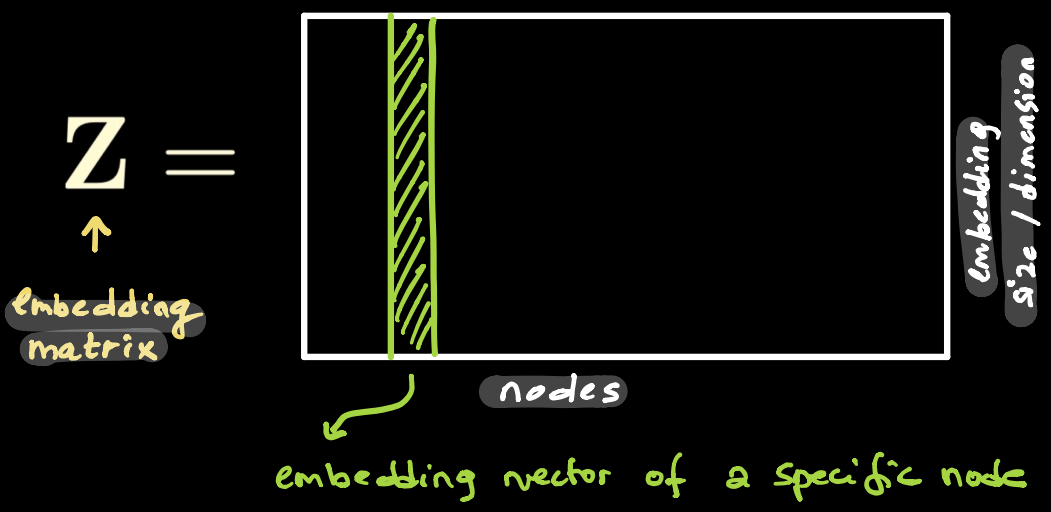
$A =$ $N \times N$ (nodes × nodes) $\quad X =$ $m \times N$ (features × nodes)

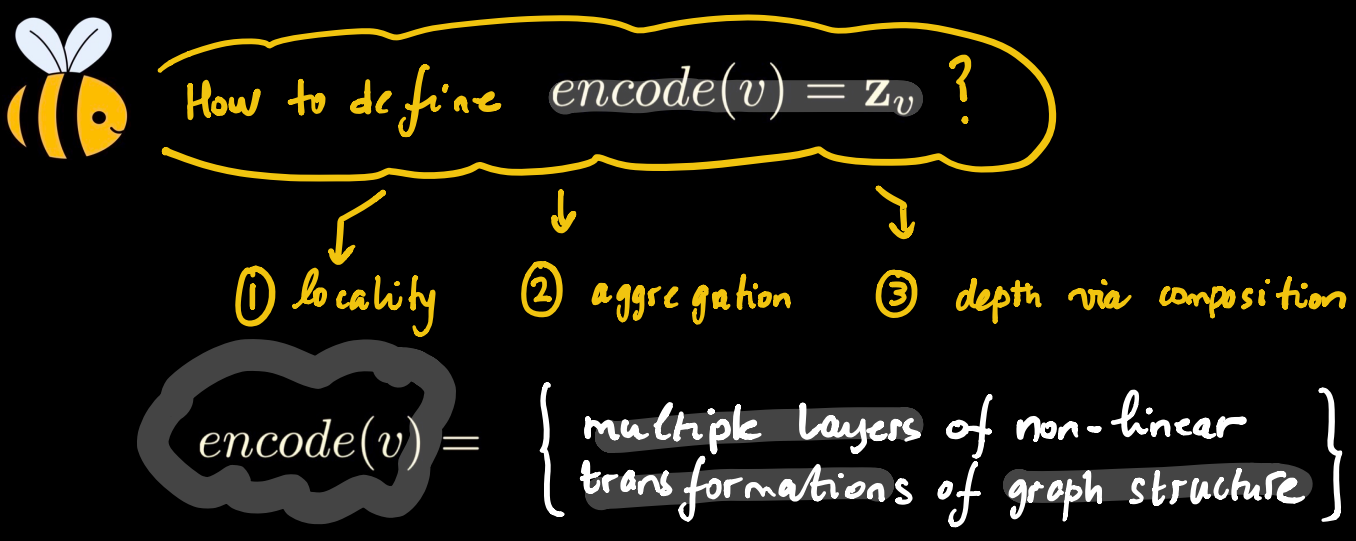
Goal : graph node embedding (encoding) → $similarity(u, v) \sim \mathbf{z}_v^T \mathbf{z}_u$

most representative of the data.



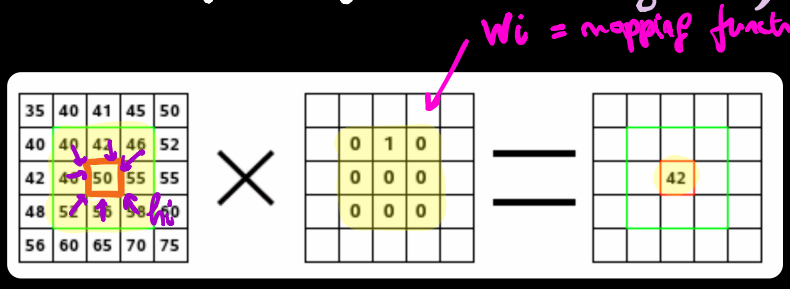

similarity between u & v in G
map relationships in the original graph to relationships in the embedding space
similarity(u, v) ≈ z_v^T z_u
dot product between node embeddings
We need to define both
encode(v) = z_v
Encoder => maps a node feature vector to a low-dim space
node in the input graph
d-dimensional embedding of node v


Z =
embedding matrix
embedding size / dimension
nodes
embedding vector of a specific node

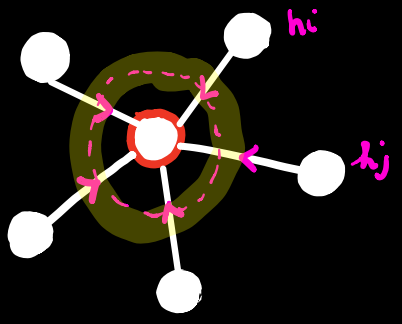

How to define encode(v) = z_v ?
① locality
② aggregation
③ depth via composition
encode(v) =
multiple layers of non-linear transformations of graph structure

RULE: Find your neighbors and define your aggregation rule to propagate information across the graph node for node feature computation.

→ Transform information ("messages" h_i) from neighbors and combine it:

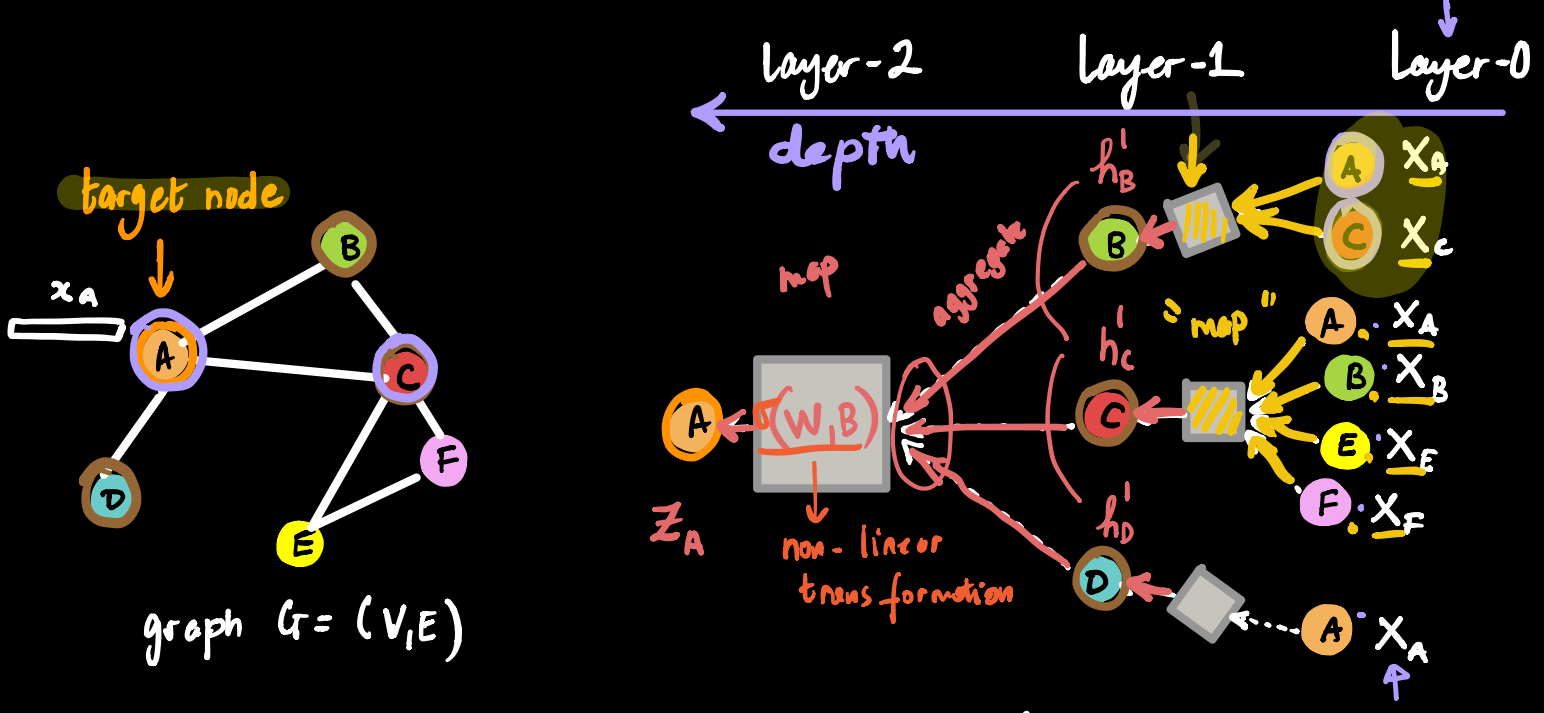


Image

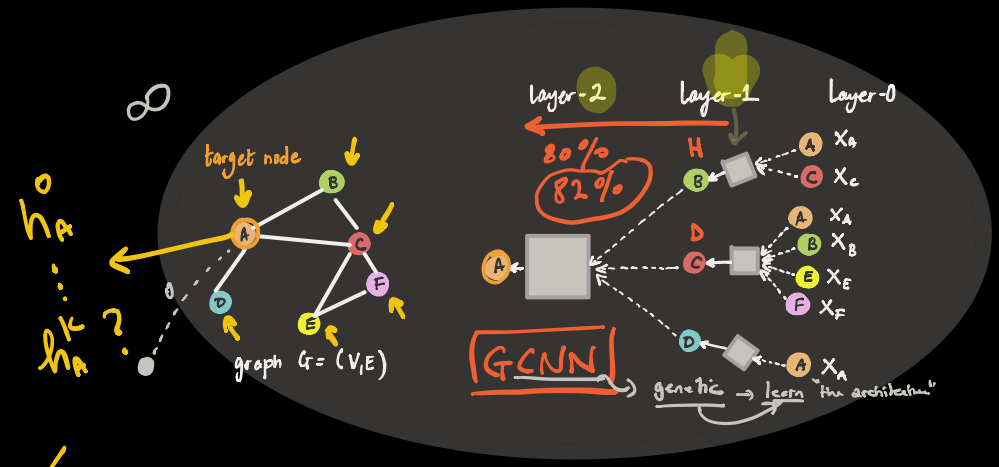


sum up transformed or mapped neighboring messages h_i by W_i → $\sum_i W_i h_i$

→ BAsic approoch
1. average messages from neighbors
2. apply neural network



* Node embeddings are defined at each layer k
* Layer-0 embedding of node u is its input feature x_u



① $\mathbf{h}_v^0 = \mathbf{x}_v$ → set initial 0-th layer embeddings to node features

② $\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, \ldots, K\}$

average of neighbor's previous layer embeddings

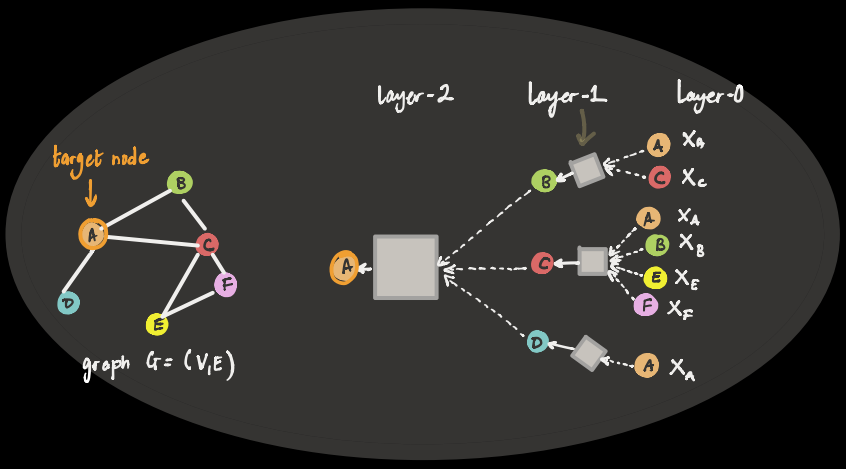
non-linearity (e.g., ReLU function)

Linear

previous layer embedding of node v

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

③ $\mathbf{z}_v = \mathbf{h}_v^K$ → embedding after K layers of neighborhood aggregation



* How to train such a model (i.e, estimate W and B)?

$$\mathcal{L}(\quad,\quad)$$

W, B

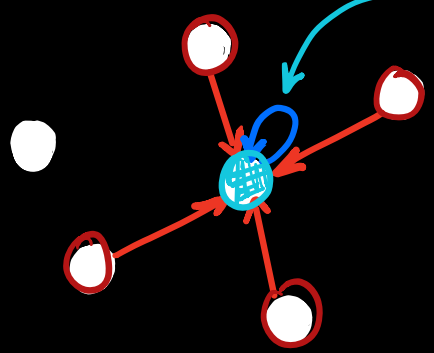
{ supervised
{ unsupervised

- Feed embeddings $\mathbf{H}^l = [\mathbf{h}_1^{l\,T} \dots \mathbf{h}_N^{l\,T}]^T$ to any loss function and run stochastic gradient descent to learn the aggregation weight matrices.

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^l\mathbf{W}_0^l + \tilde{\mathbf{A}}\mathbf{H}^l\mathbf{W}_1^l\right)$$

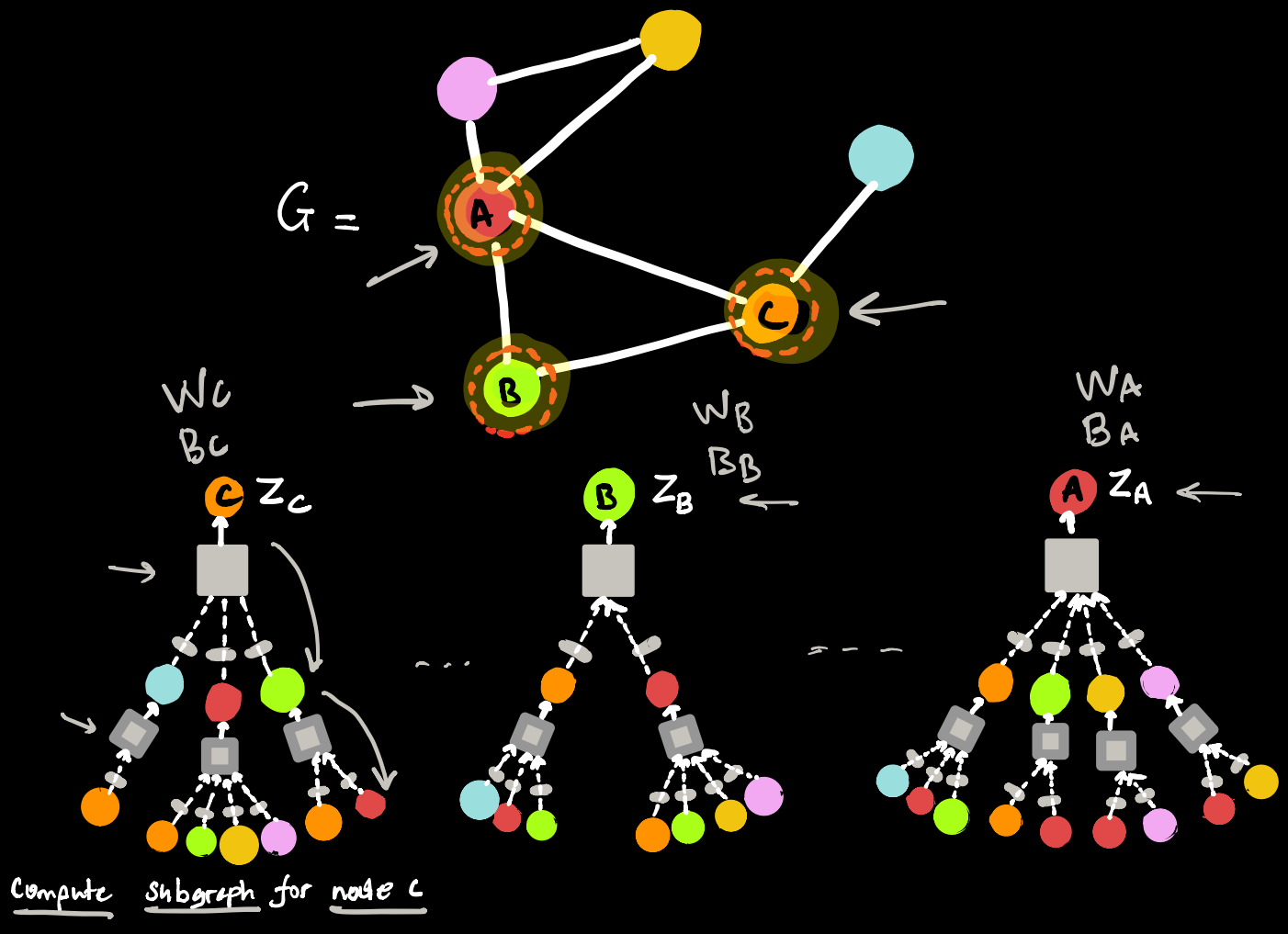
Normalized symmetric graph Laplacian

$$\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{\frac{1}{2}}$$



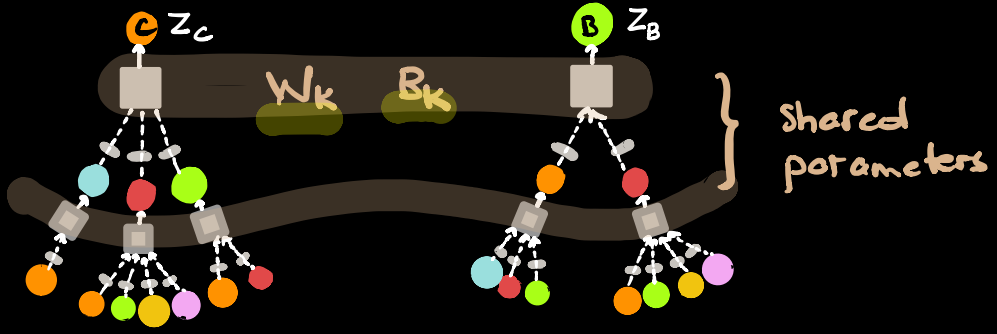
GNN model design steps

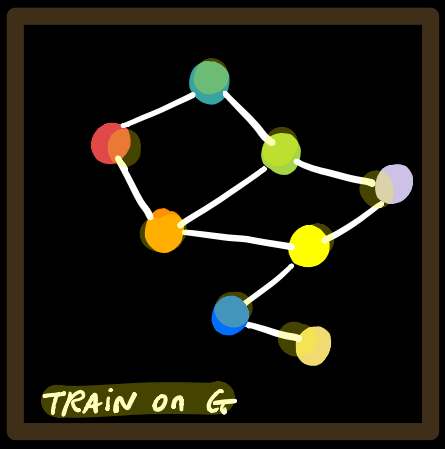
1. Define a neighborhood aggregation function (e.g., max or mean)
2. Define a loss on the embeddings
3. Train on a set of nodes (a batch of compute subgraphs)
4. Generate embeddings for nodes

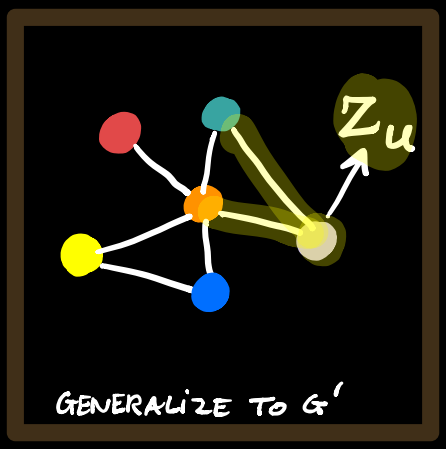


Compute subgraph for node C

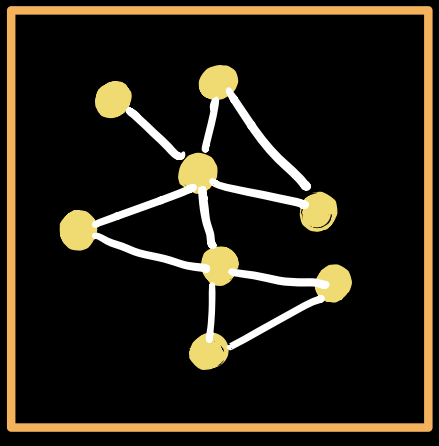
☺ RULE : sharing is less computationally expensive
↳ share parameters (W_K, B_K) across layers.



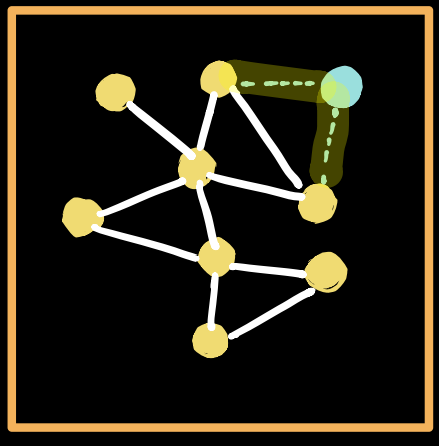




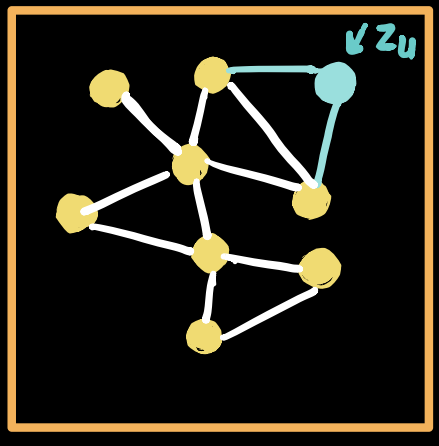
E.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B.



Training graph



New node arrives



Generate embedding for new node

generate embeddings on the fly ☺

* RECAP *

Graph Neural Networks (GNN)

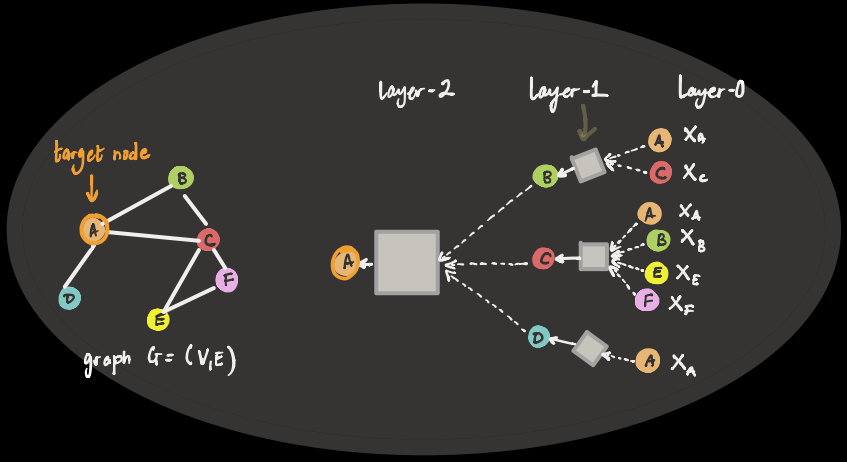
1. Locality (neighborhood) *
2. Aggregation (integration) *
3. Composition (a function of a function) (stacking layers) *

1. $\mathbf{h}_v^0 = \mathbf{x}_v$ → set initial 0-th layer embeddings to node features

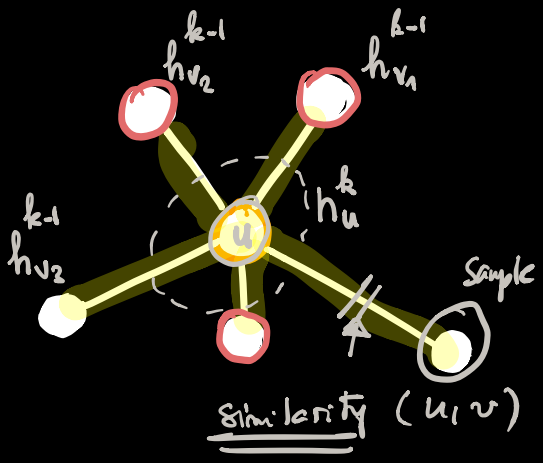
2. $\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, \ldots, K\}$

- average of neighbor's previous layer embeddings
- non-linearity (e.g., ReLU function)
- previous layer embedding of node v
- Trainable weight matrices (i.e., what we learn)

3. $\mathbf{z}_v = \mathbf{h}_v^K$ → embedding after K layers of neighborhood aggregation



"attention mechanism"



what if not all neighboors are informative ? (pay attention)

A Comprehensive Survey on Graph Neural Networks

Zonghan Wu, Shirui Pan, *Member, IEEE*, Fengwen Chen, Guodong Long, Chengqi Zhang, *Senior Member, IEEE*, Philip S. Yu, *Fellow, IEEE*

***Abstract*—Deep learning has revolutionized many machine learning tasks in recent years, ranging from image classification and video processing to speech recognition and natural language understanding. The data in these tasks are typically represented in the Euclidean space. However, there is an increasing number of applications where data are generated from non-Euclidean domains and are represented as graphs with complex relationships and interdependency between objects. The complexity of graph data has imposed significant challenges on existing machine learning algorithms. Recently, many studies on extending deep learning approaches for graph data have emerged. In this survey, we provide a comprehensive overview of graph neural networks (GNNs) in data mining and machine learning fields. We propose a new taxonomy to divide the state-of-the-art graph neural networks into four categories, namely recurrent graph neural networks, convolutional graph neural networks, graph autoencoders and spatial-temporal graph neural networks. We further discuss the applications of graph neural networks across various domains and summarize the open source codes and benchmarks of the existing algorithms on different learning tasks. Finally, we propose potential research directions in this rapidly growing field.**

***Index Terms*—Deep Learning, graph neural networks, graph convolutional networks, graph representation learning, graph autoencoder, network embedding**

I. Introduction

THE recent success of neural networks has boosted research on pattern recognition and data mining. Many machine learning tasks such as object detection [1], [2], machine translation [3], [4], and speech recognition [5], which once heavily relied on handcrafted feature engineering to extract informative feature sets, has recently been revolutionized by various end-to-end deep learning paradigms, e.g., convolutional neural networks (CNNs) [6], recurrent neural networks (RNNs) [7], and autoencoders [8]. The success of deep learning in many domains is partially attributed to the rapidly developing computational resources (e.g., GPU), the availability of big training data, and the effectiveness of deep learning to extract latent representations from Euclidean data (e.g., images, text, and video). Taking image data as an example, we can represent an image as a regular grid in the Euclidean space. A convolutional neural network (CNN) is able to exploit the shift-invariance, local connectivity, and compositionality of image data [9]. As a result, CNNs can extract local meaningful features that are shared with the entire data sets for various image analysis.

While deep learning effectively captures hidden patterns of Euclidean data, there is an increasing number of applications where data are represented in the form of graphs. For examples, in e-commence, a graph-based learning system is able to exploit the interactions between users and products to make highly accurate recommendations. In chemistry, molecules are modeled as graphs and their bioactivity needs to be identified for drug discovery. In a citation network, papers are linked to each other via citationships and they need to be categorized into different groups. The complexity of graph data has imposed significant challenges on existing machine learning algorithms. As graphs can be irregular, a graph may have a variable size of unordered nodes and nodes from a graph may have a different number of neighbors, resulting in some important operations (e.g., convolutions) being easy to compute in the image domain, but difficult to apply to the graph domain. Furthermore, a core assumption of existing machine learning algorithms is that instances are independent of each other. However, this assumption no longer holds for graph data because each instance (node) is related to others by links of various types, such as citations, friendships, and interactions.

Recently, there is increasing interest in extending deep learning approaches for graph data. Motivated by CNNs, RNNs, and autoencoders from deep learning, new generalizations and definitions of important operations have been rapidly developed over the past few years to handle the complexity of graph data. For example, a graph convolution can be generalized from a standard 2D convolution. As illustrated in Figure 1, an image can be considered as a special case of graphs where pixels are connected by adjacent pixels. Similar to 2D convolution, one may perform graph convolution by taking the weighted average of a nodes neighborhood information.

There are a limited number of existing reviews on the topic of graph neural networks (GNNs). Using the term *geometric deep learning*, Bronstein et al. [9] give an overview of deep learning methods in the non-Euclidean domain, including graphs and manifolds. Although it is the first review on GNNs, this survey mainly reviews convolutional GNNs. Hamilton et al. [10] cover a limited number of GNNs with a focus

Z. Wu, F. Chen, G. Long, C. Zhang are with Centre for Artificial Intelligence, FEIT, University of Technology Sydney, NSW 2007, Australia (E-mail: zonghan.wu-3@student.uts.edu.au; fengwen.chen@student.uts.edu.au; guodong.long@uts.edu.au; chengqi.zhang@uts.edu.au).

S. Pan is with Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia (Email: shirui.pan@monash.edu).

P. S. Yu is with Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053, USA (Email: psyu@uic.edu)

Corresponding author: Shirui Pan.

Manuscript received Dec xx, 2018; revised Dec xx, 201x.