Graph Neural Network

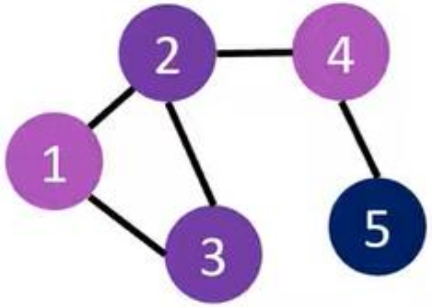
A Simple Graph

The most fundamental part of GNN is a Graph.

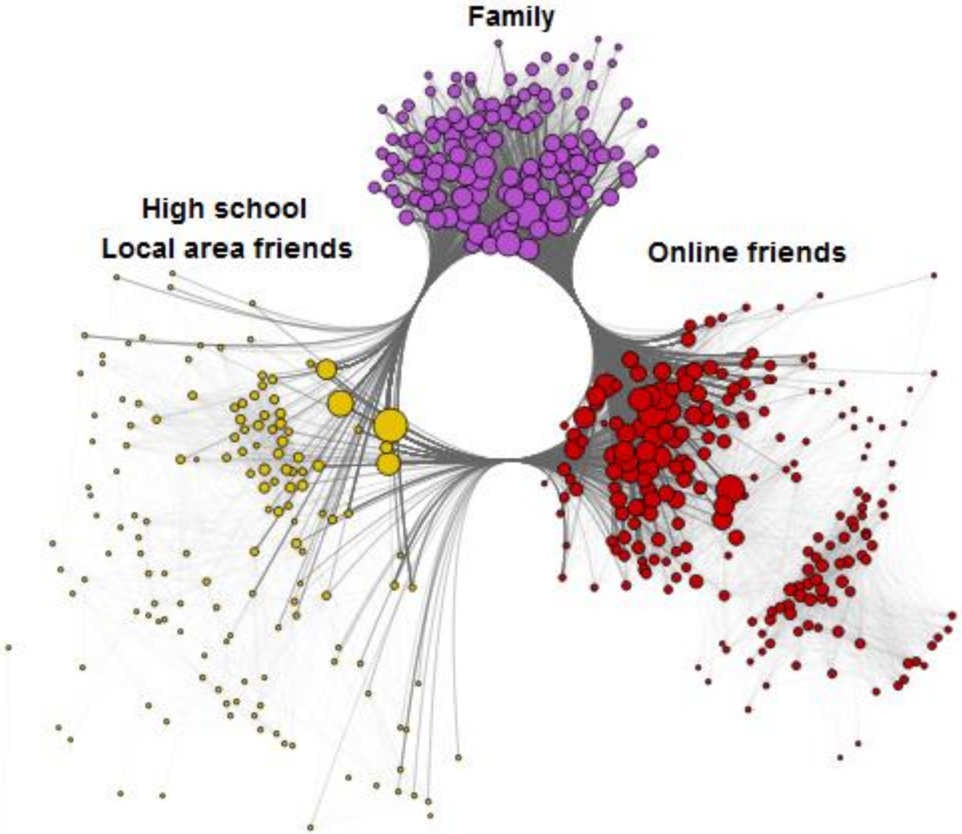
	V1	V2	...
V1	0	1	...
V2	1	0	...
V3	1	1	...
...	...	...	...

Adjacency matrix
V x V

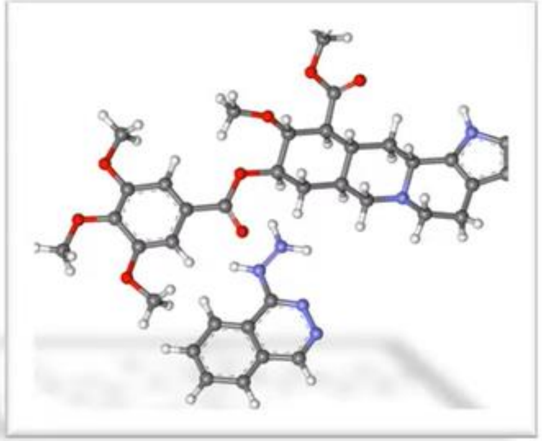
A graph can represent things like social media networks, or molecules. Think of nodes as users, and edges as connections.



$G = (V,E)$



Graph Data is everywhere



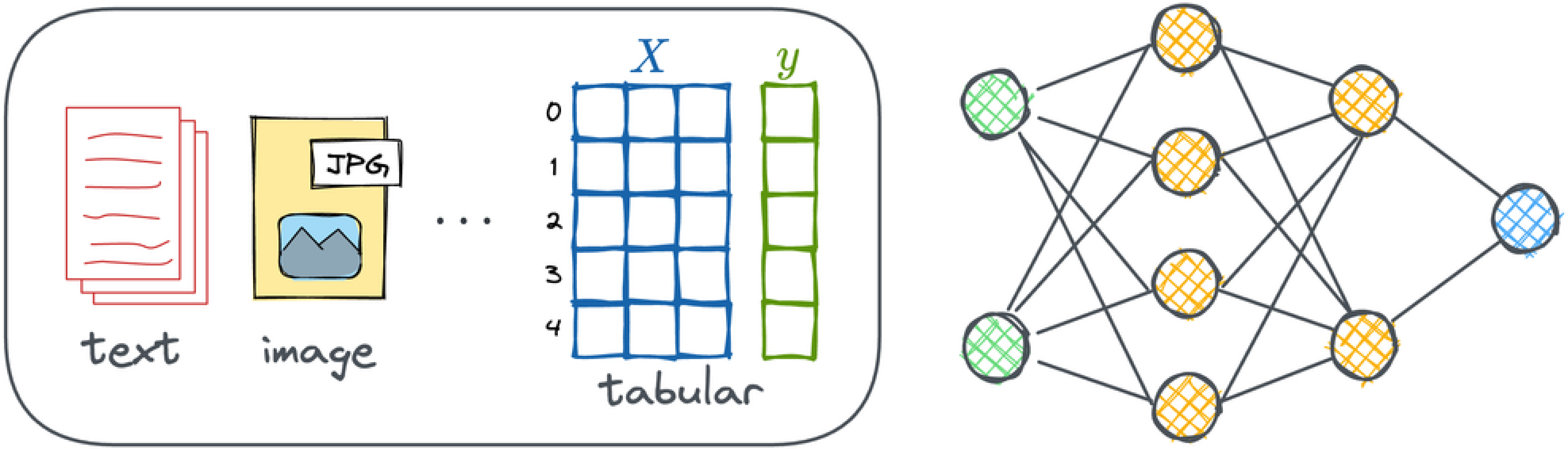
Medicine / Pharmacy

Recommender Systems

Social Networks

3D Games / Meshes

Traditional DL

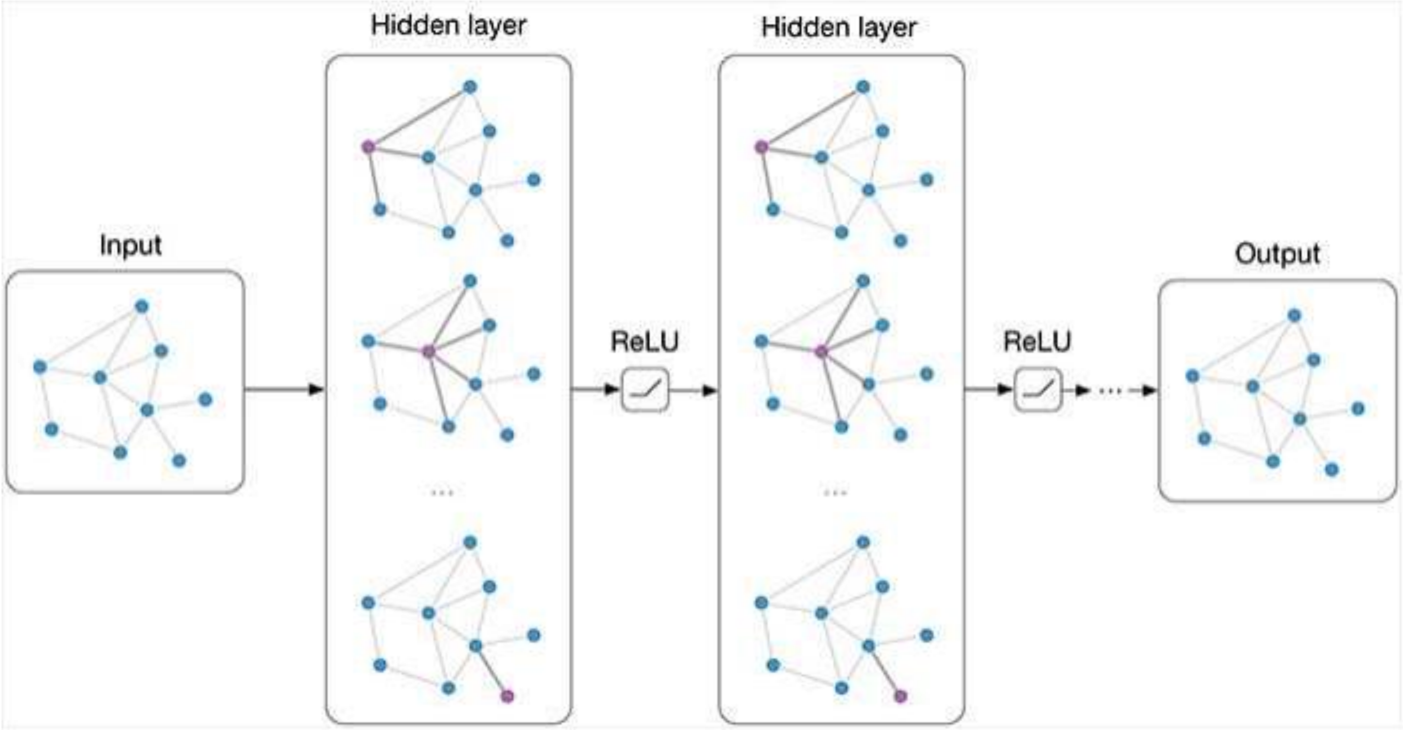


data formats that are tabular, image-based, or sequential (like language) in nature.

a significant proportion of our real-world data often exists in the form of graphs,

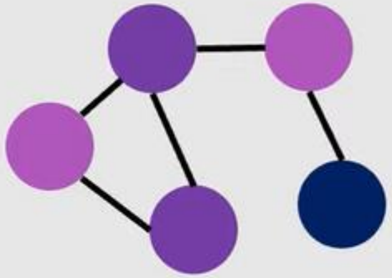
What is a Graph Neural Network (GNN)?

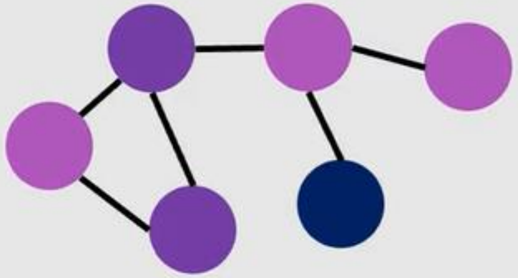
- To fill this gap, offering a way to extend deep learning techniques to graph-structured data.
- Processes and analyzes data represented as nodes (entities) and edges (relationships).
- **Components**:
- **Nodes**: Represent entities in the graph.
- **Edges**: Represent relationships or connections between nodes.
- **Features**: Attributes of nodes or edges.



Problem: Graph Data is different!

Difference 1: Size and Shape

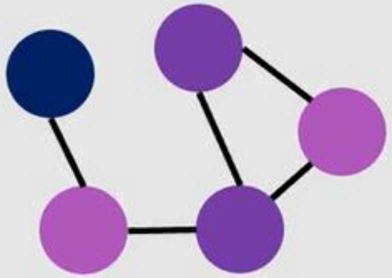




Size independent

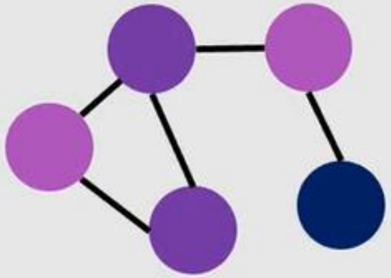
Difference 2: Isomorphism

~~Adjacency matrix as input~~



Permutation invariance

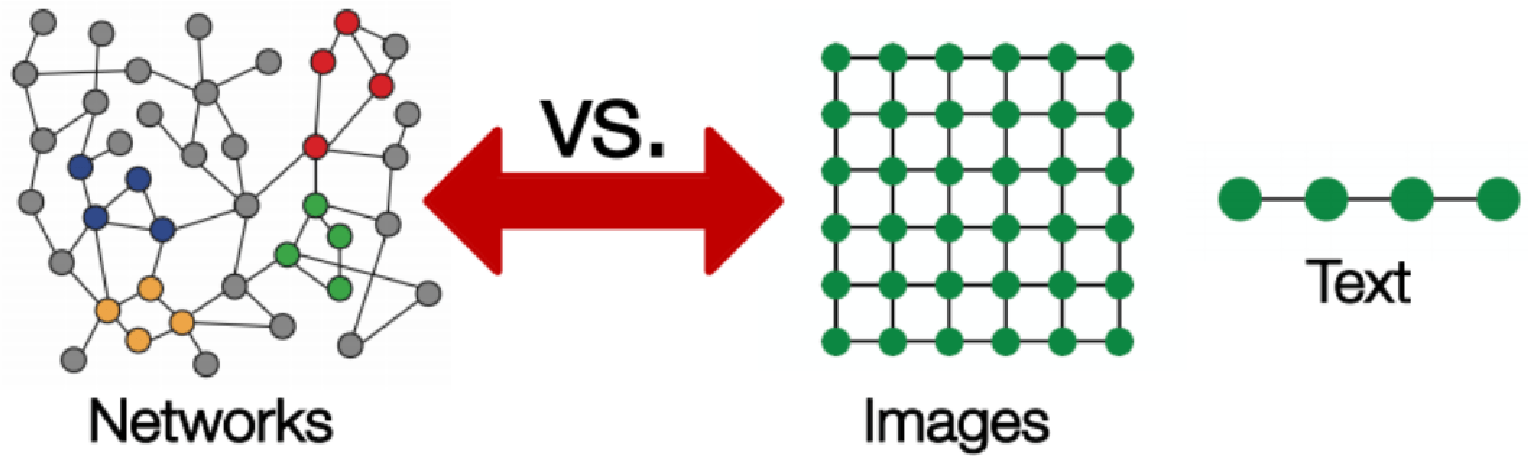
Difference 3: Grid structure



Non-euclidean space

Why is it hard to analyze a graph?

- **Complex Nature of Graph Data:**
- Graph data lacks fixed forms and has variable-sized, unordered nodes.
- Nodes can have varying numbers of neighbors.
- **Limitations of Conventional ML/DL Tools:**
- Specialized for structured data like:
- Images (fixed-size grid graphs).
- Text and speech (line graphs).
- **Dependency Issue:**
- Existing ML algorithms assume independent instances.
- Graph data violates this due to interrelated nodes with various link types.

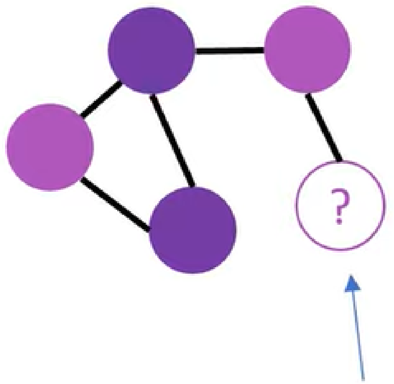


Types of Graph Neural Networks Tasks

- **Graph Classification**: Classify entire graphs into categories.
- *Example*: Social network analysis (e.g., professional vs. friendship networks).
- **Node Classification:** Predict missing node labels using neighboring node information.
- *Example*: Fraud detection in transaction networks.
- **Link Prediction**: Predict potential or missing links between nodes in an incomplete graph.
- *Example*: Friend recommendation in social networks.
- **Community Detection**: Identify clusters of nodes based on edge structure, weights, and distances.
- *Example*: Detecting communities in social media platforms.
- **Graph Embedding**: Convert graph data into vector representations while preserving structure and relationships.
- *Example*: Input for machine learning tasks like graph-based recommendation systems.

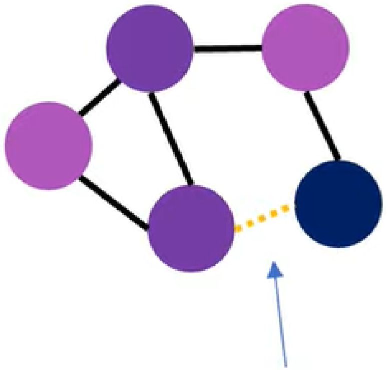
Examples for Machine Learning Problems with Graph Data

Node-level predictions



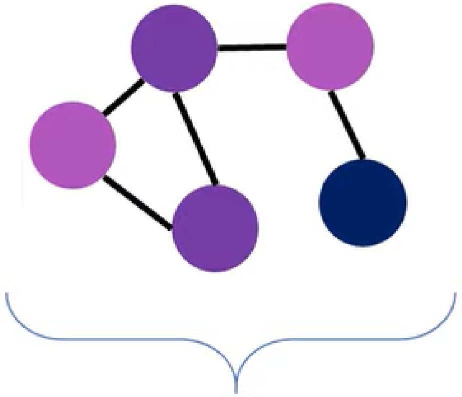
Does this person smoke?
(unlabeled node)

Edge-level predictions (Link prediction)

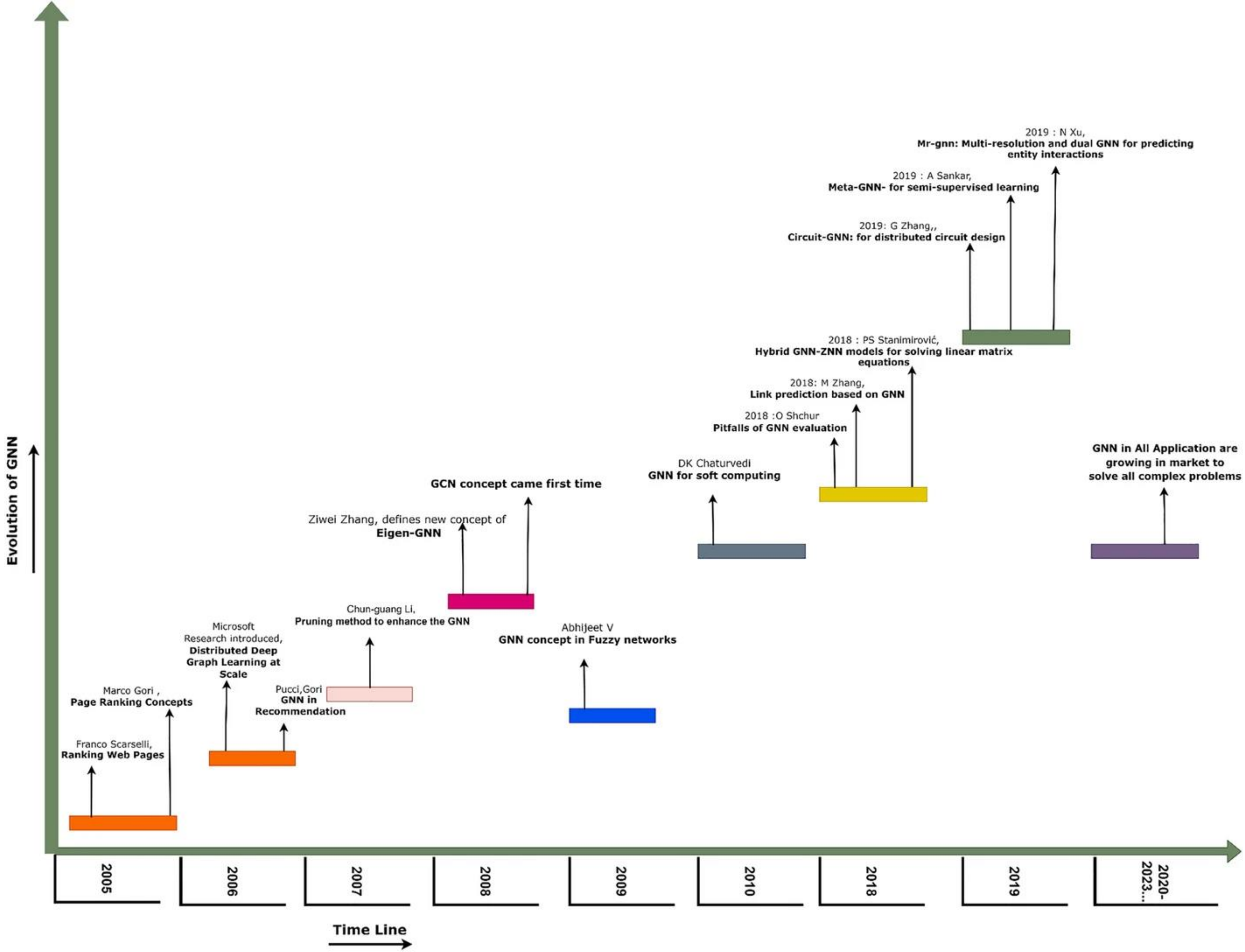


Next Netflix video?

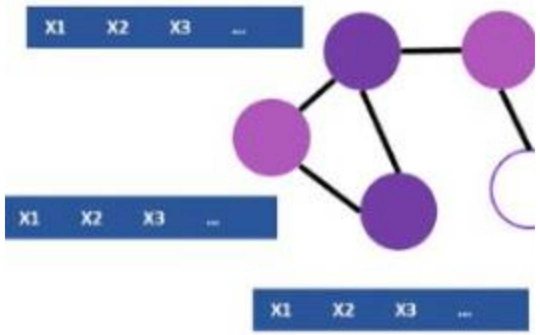
Graph-level predictions



Is this molecule a suitable drug?


2019 : N Xu,
Mr-gnn: Multi-resolution and dual GNN for predicting entity interactions
2019 : A Sankar,
Meta-GNN- for semi-supervised learning
2019: G Zhang,,
Circuit-GNN: for distributed circuit design
2018 : PS Stanimirović,
Hybrid GNN-ZNN models for solving linear matrix equations
2018: M Zhang,
Link prediction based on GNN
2018 :O Shchur
Pitfalls of GNN evaluation
DK Chaturvedi
GNN for soft computing
GNN in All Application are growing in market to solve all complex problems
GCN concept came first time
Ziwei Zhang, defines new concept of
Eigen-GNN
Chun-guang Li,
Pruning method to enhance the GNN
Abhijeet V
GNN concept in Fuzzy networks
Microsoft Research introduced,
Distributed Deep Graph Learning at Scale
Marco Gori ,
Page Ranking Concepts
Pucci,Gori
GNN in Recommendation
Franco Scarselli,
Ranking Web Pages
Evolution of GNN
2005
2006
2007
2008
2009
2010
2018
2019
2020-2023...
Time Line

Node Embedding



- Each node is represented as a vector in a **low-dimensional space** (d-dimensional), which is much smaller than the size of the graph itself (e.g., the adjacency matrix or feature matrix).
- The goal of node embedding is to learn an informative representation of each node in the graph.
- **Capturing Structural and Feature Information**:
    - Neighborhood: The structure and types of connections it has with other nodes.
    - Node Features: Any attributes or labels associated with the node.
    - Graph Context: The broader context of the graph, which helps the node embedding reflect not just the local connections but also global graph structure.
- Node embeddings are learned through a series of graph convolutional layers that aggregate information from neighboring nodes, progressively refining the embeddings.

Neighborhood: The Structure and Types of Connections

- Node embeddings capture how a node is connected to its neighbors.
- Example:
- Consider a social network with the following relationships:
    - Node A (Alice) is friends with Node B (Bob) and Node C (Charlie).
  - Node B (Bob) is friends with Node A (Alice), Node C (Charlie), and Node D (David).
  - Node C (Charlie) is friends with Node A (Alice), Node B (Bob), and Node E (Eve).
  - For **Node A (Alice)**:
  - **Neighborhood Information**: Alice's direct neighbors are **Bob** and **Charlie**. So, her node embedding will reflect the structure of these relationships.
  - Alice's embedding will consider the properties of her neighbors

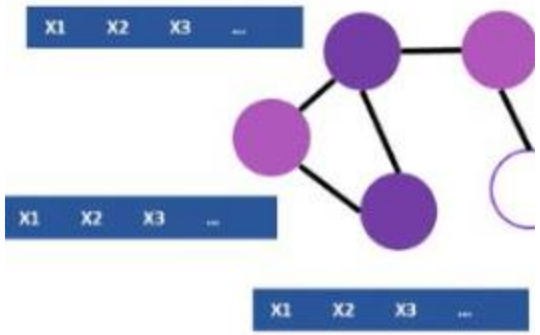
Node Features: Attributes or Labels Associated with the Node

- Example:
- Let’s assume each individual in the social network has the following features:
- **Age**
- **Interests** (e.g., books, technology, sports)
- **Occupation** (e.g., teacher, engineer)
- For **Node A (Alice)**:
- **Node Features**: Alice may have the following features:
  - **Age**: 30
  - **Interest**: Books, technology

Graph Context

- **global context** - even though a node has specific neighborhood relationships, it is also influenced by the structure and distribution of the entire graph.
- Example:
- If the network has distinct **communities** or **groups** (e.g., people interested in different types of activities like sports, literature, or technology), the **global context** might show that Alice, who is interested in books and technology, is part of a group of people with similar interests, even if some of her direct neighbors are from different groups.
- For **Node A (Alice)**:
- **Global Context**: The embedding for Alice will take into account not only her direct neighbors (Bob and Charlie) but also the broader structure of the entire network.

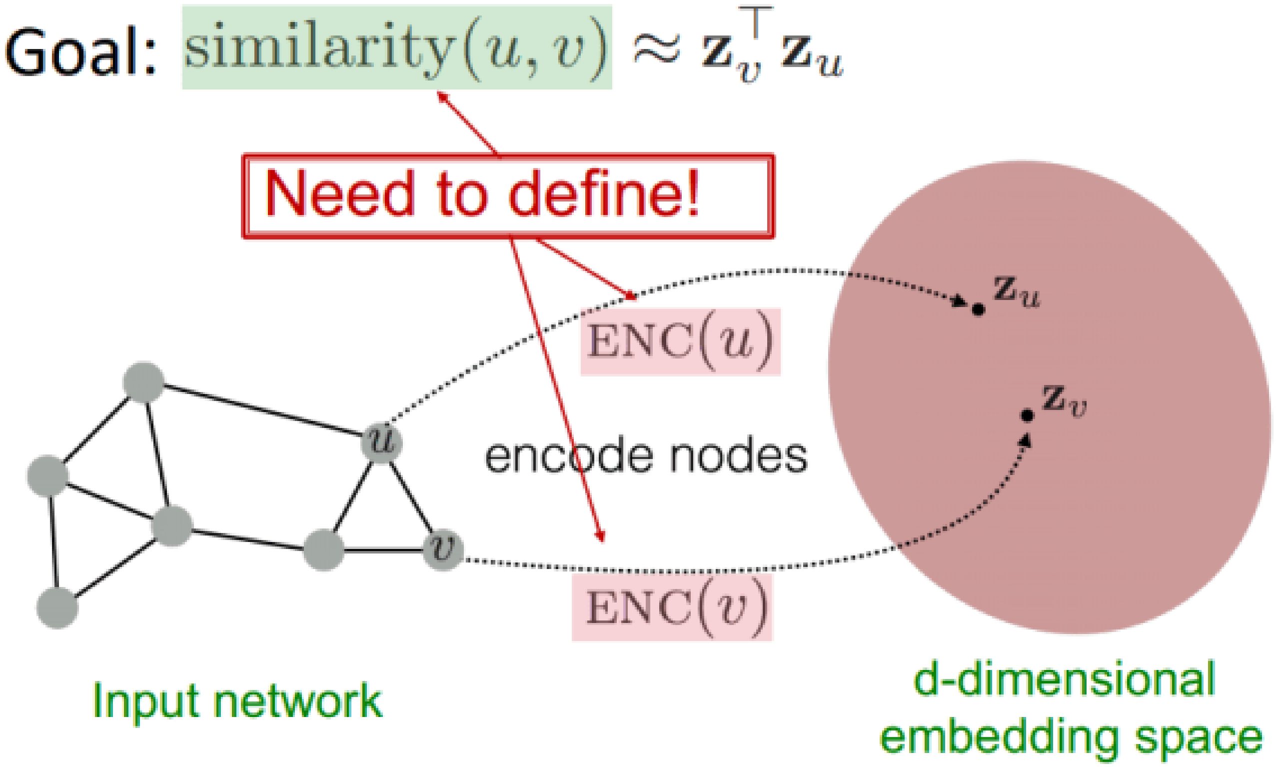
Node Embedding



- Each node is represented as a vector in a **low-dimensional space** (d-dimensional), which is much smaller than the size of the graph itself (e.g., the adjacency matrix or feature matrix).
- The goal of node embedding is to learn an informative representation of each node in the graph.
- **Capturing Structural and Feature Information**:
  - Neighborhood: The structure and types of connections it has with other nodes.
  - Node Features: Any attributes or labels associated with the node.
  - Graph Context: The broader context of the graph, which helps the node embedding reflect not just the local connections but also global graph structure.
- Node embeddings are learned through a series of graph convolutional layers that aggregate information from neighboring nodes, progressively refining the embeddings.

Similarity

Node embedding represent each node in such a way that similar nodes in terms of structure, attributes, or role in the graph are embedded close to each other in the vector space.

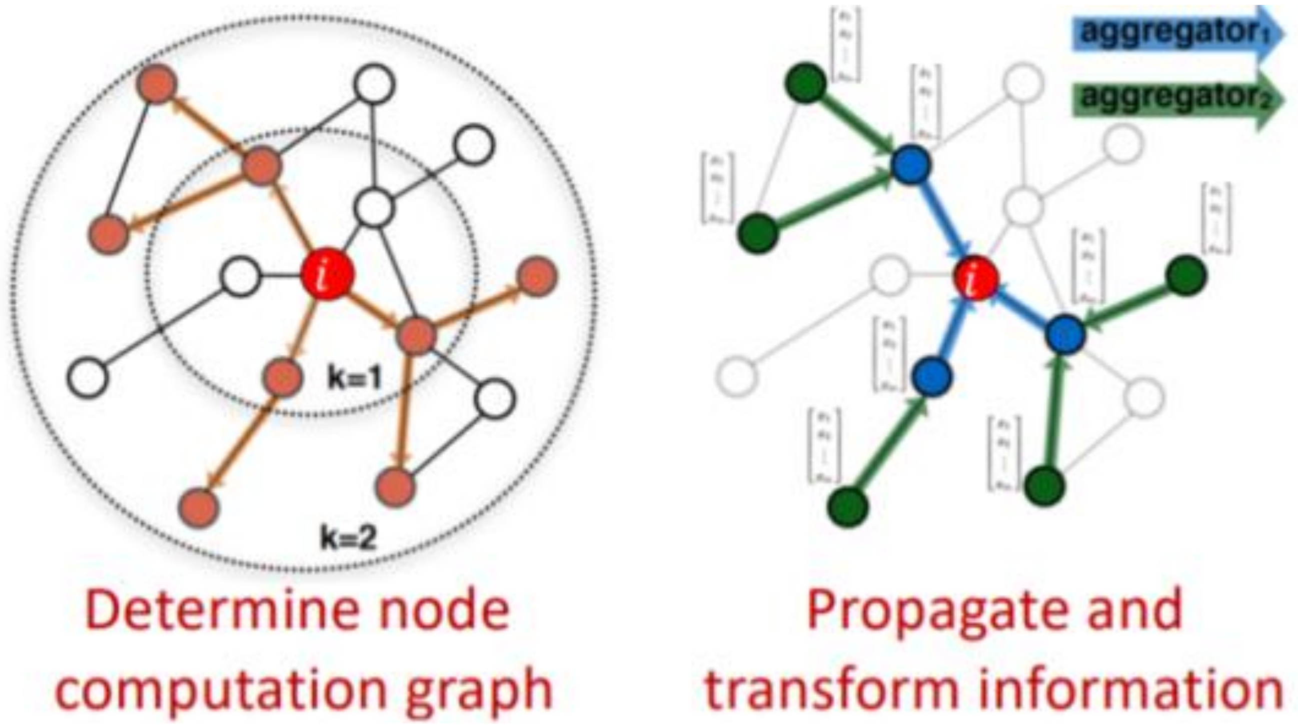


$\boldsymbol{x_u}$ and $\boldsymbol{x_v}$ are two feature vectors.
Now we'll define the encoder function ***Enc(u)*** and ***Enc(v)***, which convert the feature vectors to $\boldsymbol{z_u}$ and $\boldsymbol{z_v}$.

The encoder function should be able to perform :

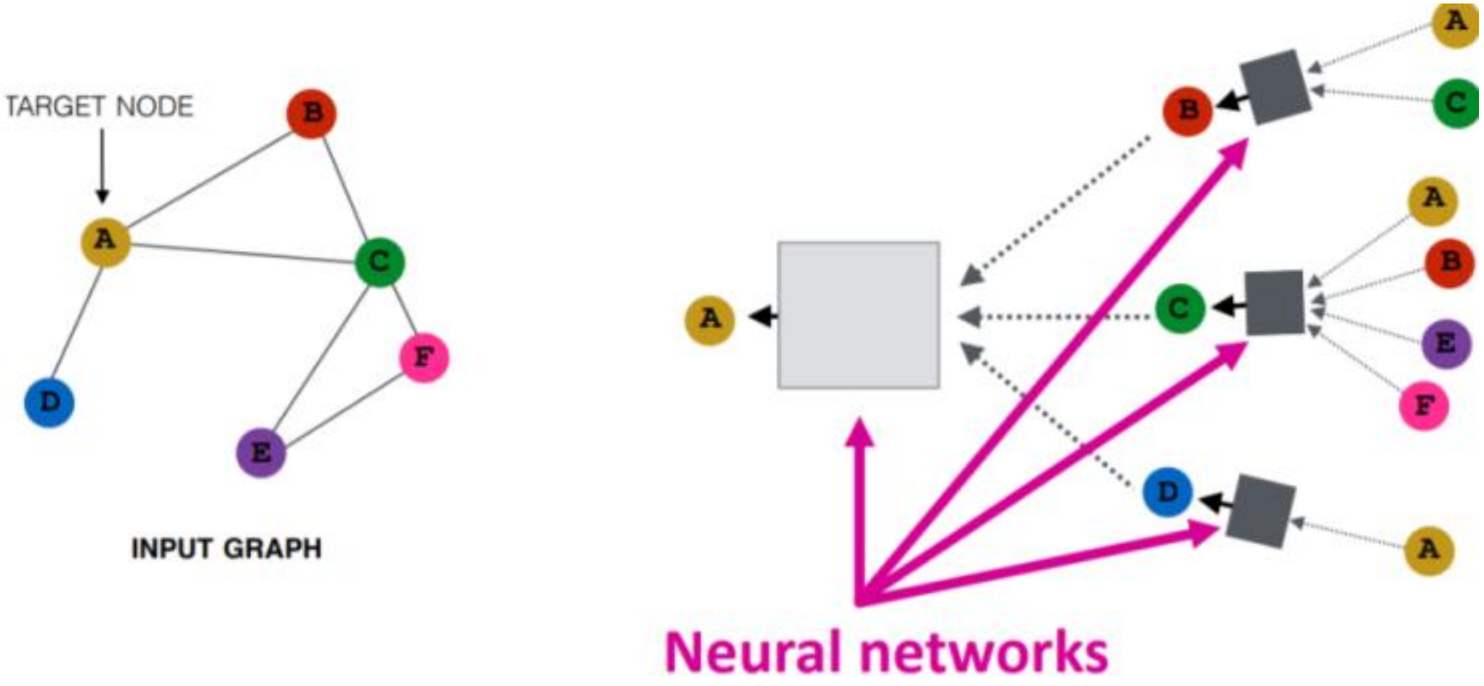
- Locality (local network neighborhoods)
- Aggregate information
- Stacking multiple layers (computation)

Locality- computational graph



how this node is connected to its neighbors and those neighbors' neighbors

Aggregation (Message Passing)

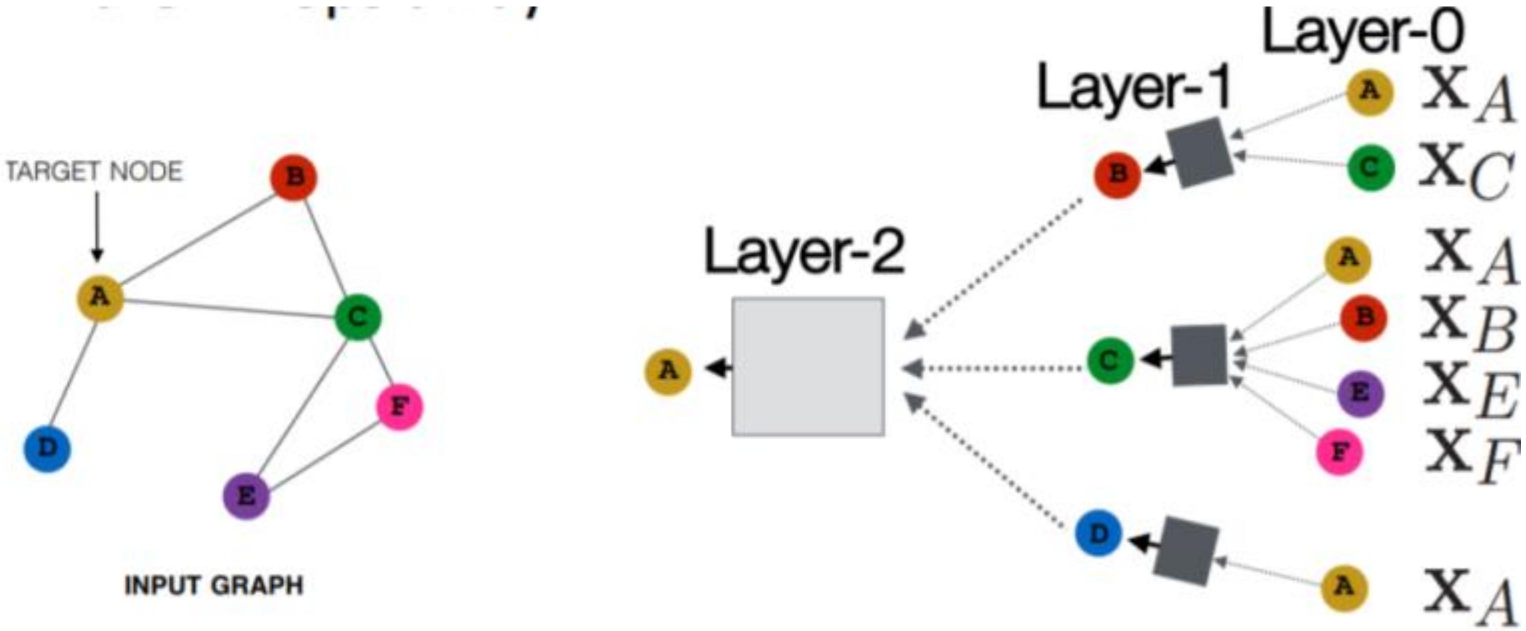


process of collecting and combining information from neighboring nodes in a computational graph.

Aggregation Functions:

- **Sum**: Add the features of the neighboring nodes.
- **Mean**: Take the average of the features of the neighboring nodes.
- **Max**: Select the maximum value of features across neighbors.

Forward propagation



Forward propagation

1. **Initial Node Embeddings:**
   - Each node v is represented by an initial feature vector $h_v^{(0)} \in \mathbb{R}^d$, where d is the dimensior of the feature space.
2. **Message Passing (Aggregation):**
   - At layer l, each node aggregates information from its neighbors $N(v)$ (neighbors of node v):

$$m_v^{(l)} = \sum_{u \in N(v)} \text{Aggregate}(h_u^{(l-1)}, A_{uv})$$

   where $m_v^{(l)}$ is the aggregated message for node v at layer l, $h_u^{(l-1)}$ is the feature vector of neighbor node u at the previous layer, and A_{uv} is the edge weight (or adjacency relation) between nodes u and v.

Forward propagation

3. **Node Update (Transformation):**
   - After aggregation, the node's feature vector is updated using a transformation function, typically a linear transformation followed by a non-linear activation function (e.g., ReLU):

     $$h_v^{(l)} = \sigma\left(W^{(l)} \cdot m_v^{(l)} + b^{(l)}\right)$$

     where $W^{(l)}$ is the weight matrix, $b^{(l)}$ is the bias, and σ is the activation function (e.g., ReLU).
4. **Multiple Layers:**
   - The process is repeated across multiple layers L of the network, with the final representation of each node reflecting both local and global graph structure:

     $$h_v^{(L)} = \text{Update function}(h_v^{(L-1)}, \{m_u^{(L-1)}\})$$
5. **Final Node Representation:**
   - After passing through all layers, the node embeddings $h_v^{(L)}$ are used for downstream tasks such as classification, regression, or link prediction.

Back propagation

2. Loss Calculation:

After obtaining the final embeddings $h_v^{(L)}$, calculate the loss $\mathcal{L}$ for the task (e.g., node classification):

$$\mathcal{L} = \sum_{v \in V} \text{Loss}(y_v, \hat{y}_v)$$

Where:

- $\hat{y}_v$ is the predicted label (or output) for node v based on its embedding $h_v^{(L)}$.

Back propagation

a. Compute Gradients for the Output Layer L:

For each node v, compute the gradient of the loss with respect to the node embedding $h_v^{(L)}$:

$$\frac{\partial \mathcal{L}}{\partial h_v^{(L)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_v} \cdot \frac{\partial \hat{y}_v}{\partial h_v^{(L)}}$$

Then, backpropagate this gradient to compute the gradients for the weight matrix $W^{(L)}$ and bias vector $b^{(L)}$ at the output layer.

b. Compute Gradients for Intermediate Layers:

For each intermediate layer l, compute the gradient of the loss with respect to the node embedding $h_v^{(l)}$:

$$\frac{\partial \mathcal{L}}{\partial h_v^{(l)}} = \sum_{u \in N(v)} \frac{\partial \mathcal{L}}{\partial h_u^{(l+1)}} \cdot \frac{\partial h_u^{(l+1)}}{\partial h_v^{(l)}}$$

This allows the gradient to be propagated backward through the layers.

c. Gradient of the Aggregation Function:

During the backpropagation, the gradient with respect to the aggregation function (e.g., summation of neighbors' embeddings) is also computed:

$$\frac{\partial \mathcal{L}}{\partial m_v^{(l)}} = \sum_{u \in N(v)} \frac{\partial \mathcal{L}}{\partial h_u^{(l+1)}} \cdot \frac{\partial h_u^{(l+1)}}{\partial m_v^{(l)}}$$

Back propagation

4. Parameter Update:

After computing the gradients for the loss with respect to the model parameters, update the weigh $W^{(l)}$ and biases $b^{(l)}$ using an optimization algorithm like **Stochastic Gradient Descent (SGD)** or **Adam**:

$$W^{(l)} \leftarrow W^{(l)} - \alpha \cdot \frac{\partial \mathcal{L}}{\partial W^{(l)}}$$

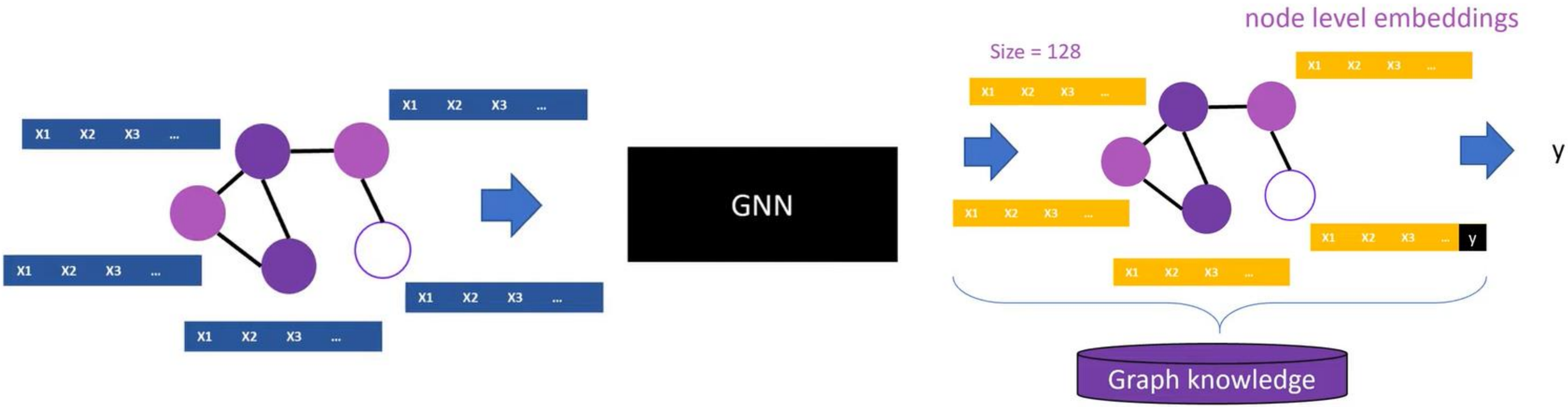
$$b^{(l)} \leftarrow b^{(l)} - \alpha \cdot \frac{\partial \mathcal{L}}{\partial b^{(l)}}$$

Where α is the learning rate, which controls how much to adjust the weights during each step.

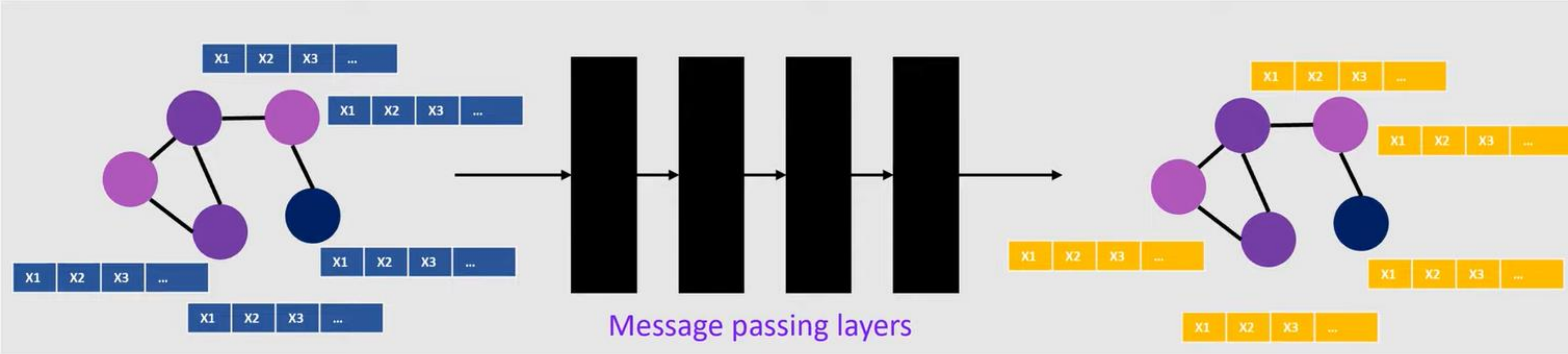
Fundamental Idea of GNNs

>> Learning a for neural networks <u>suitable representation</u> of graph data. <<

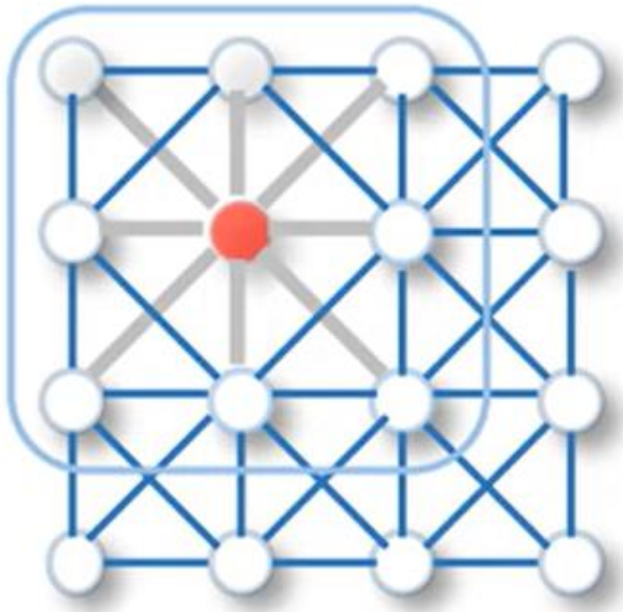
= Representation learning

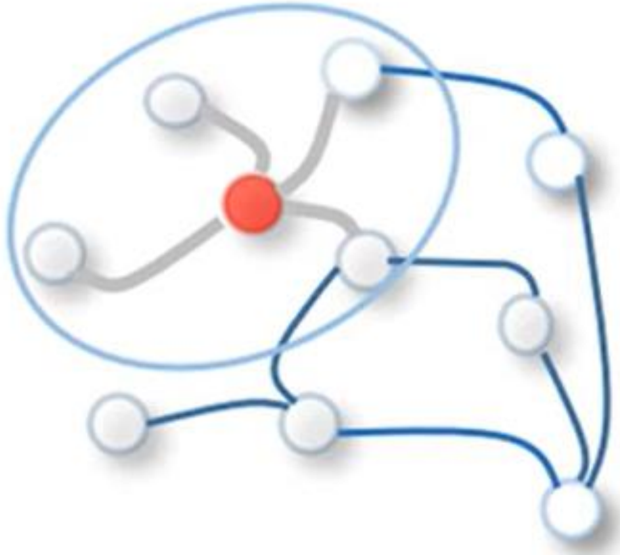


How do Graph Neural Networks work?

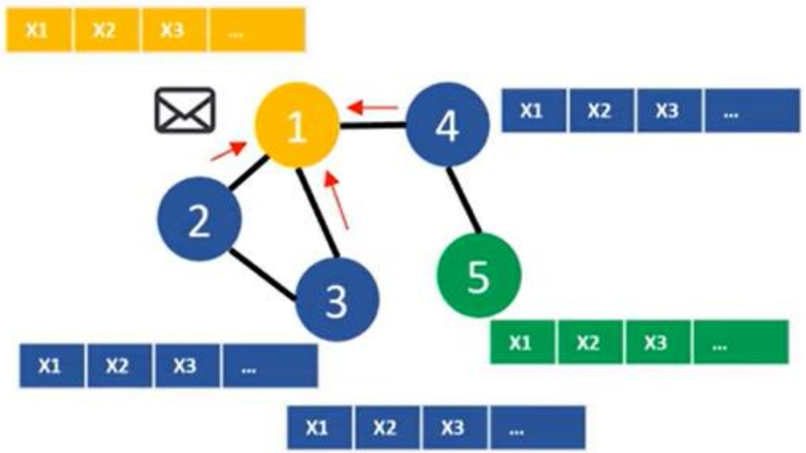


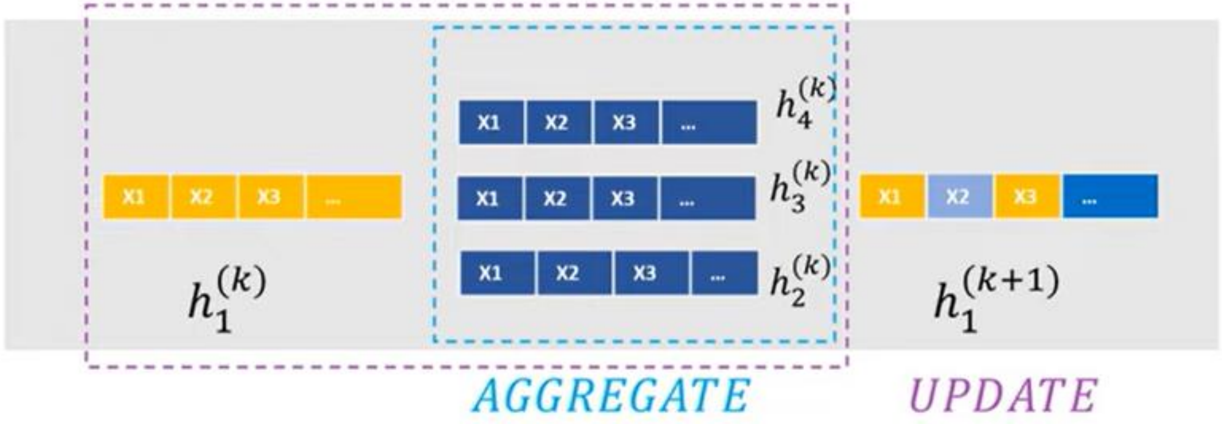
What is happening in the Message Passing Layers?



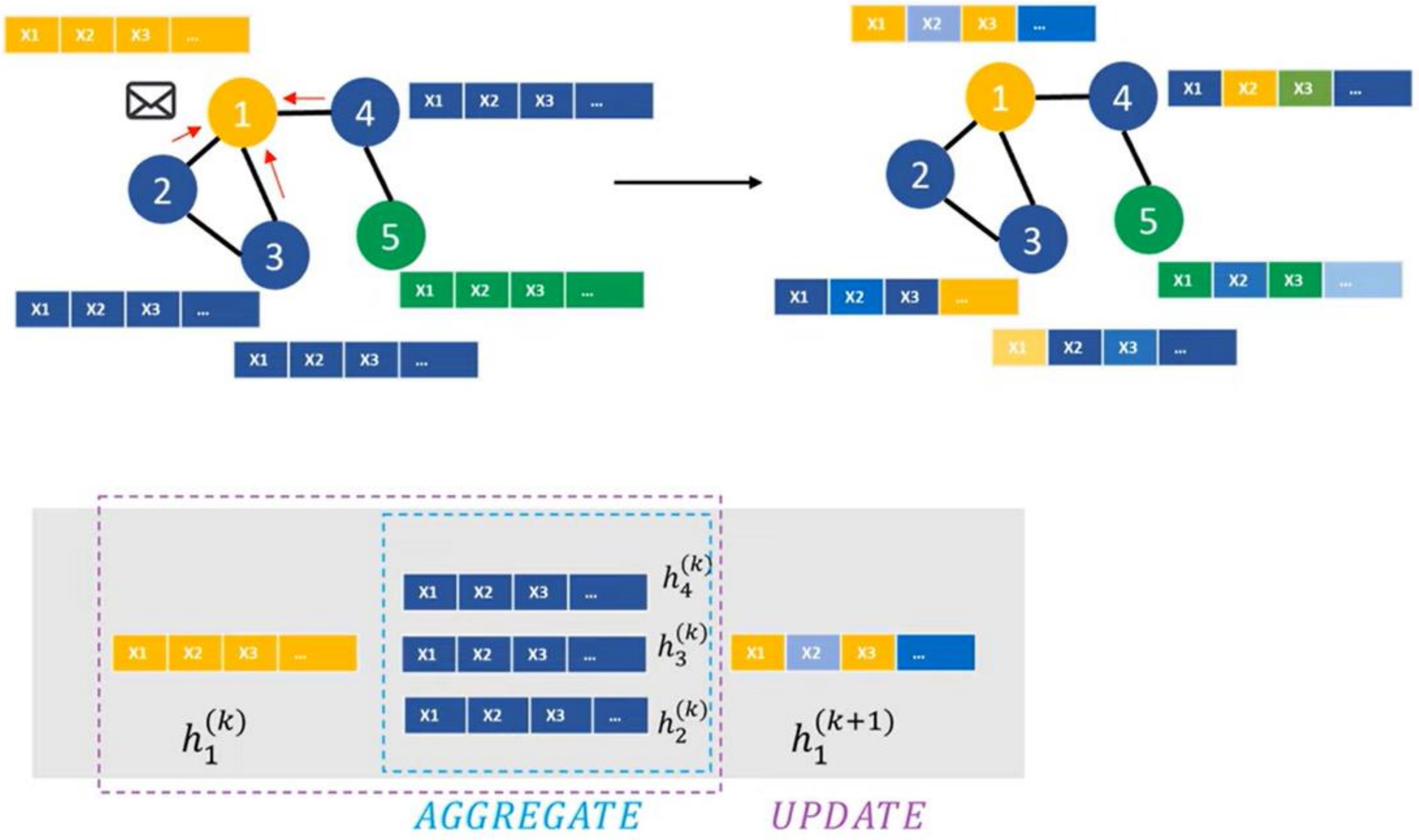


What is happening in the Message Passing Layers?

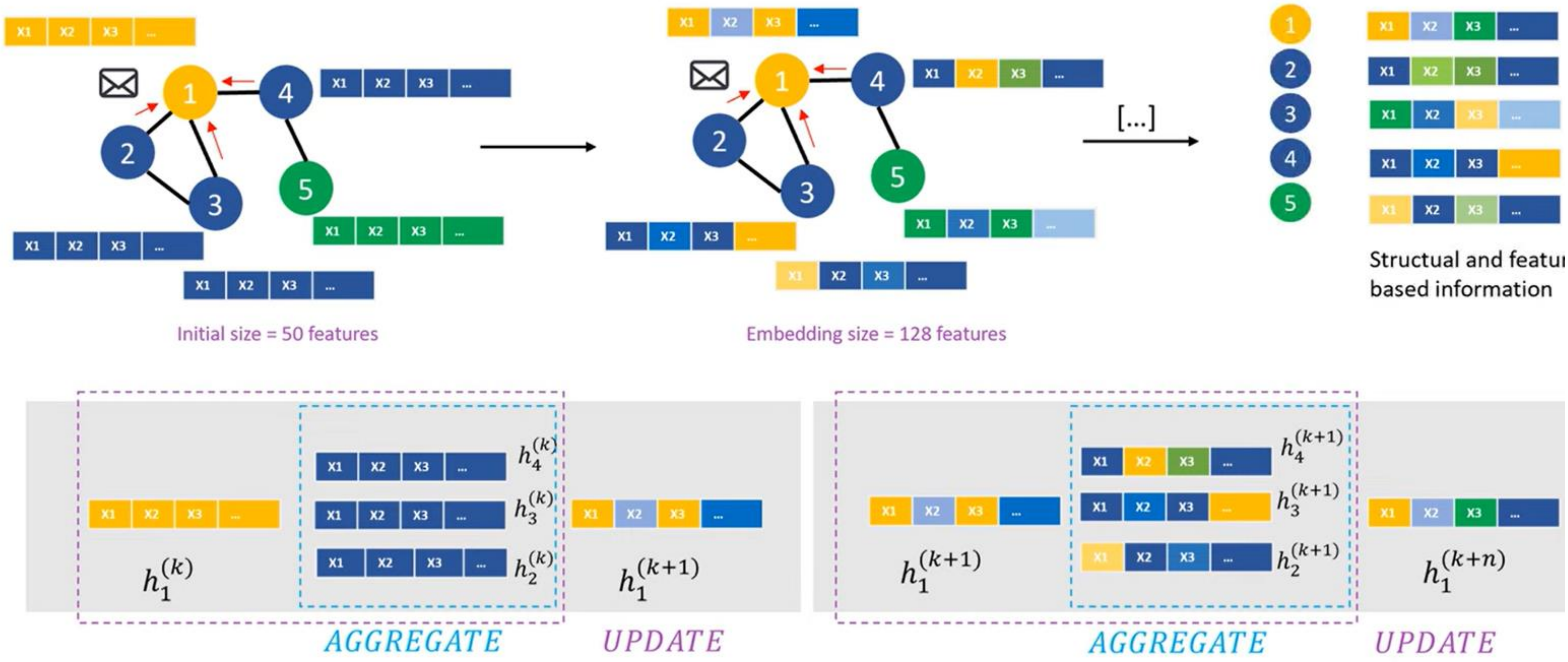




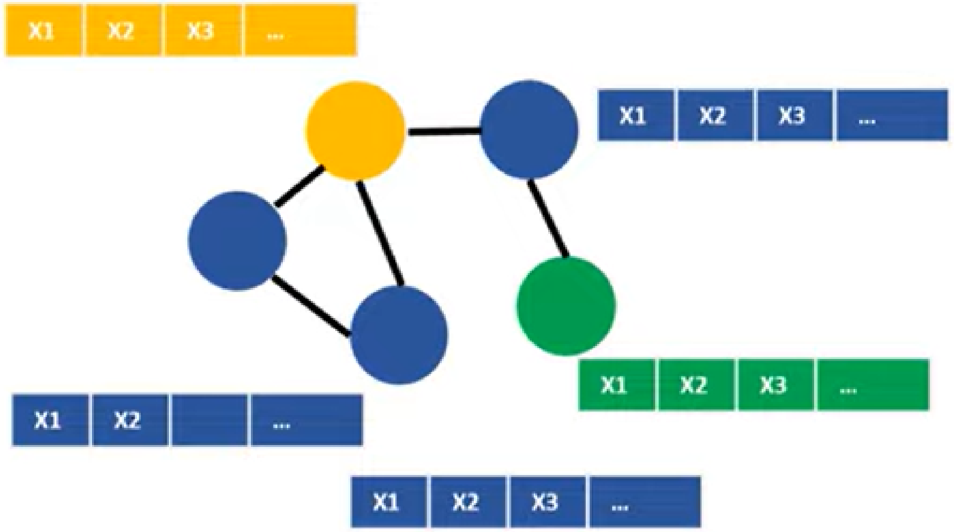
What is happening in the Message Passing Layers?



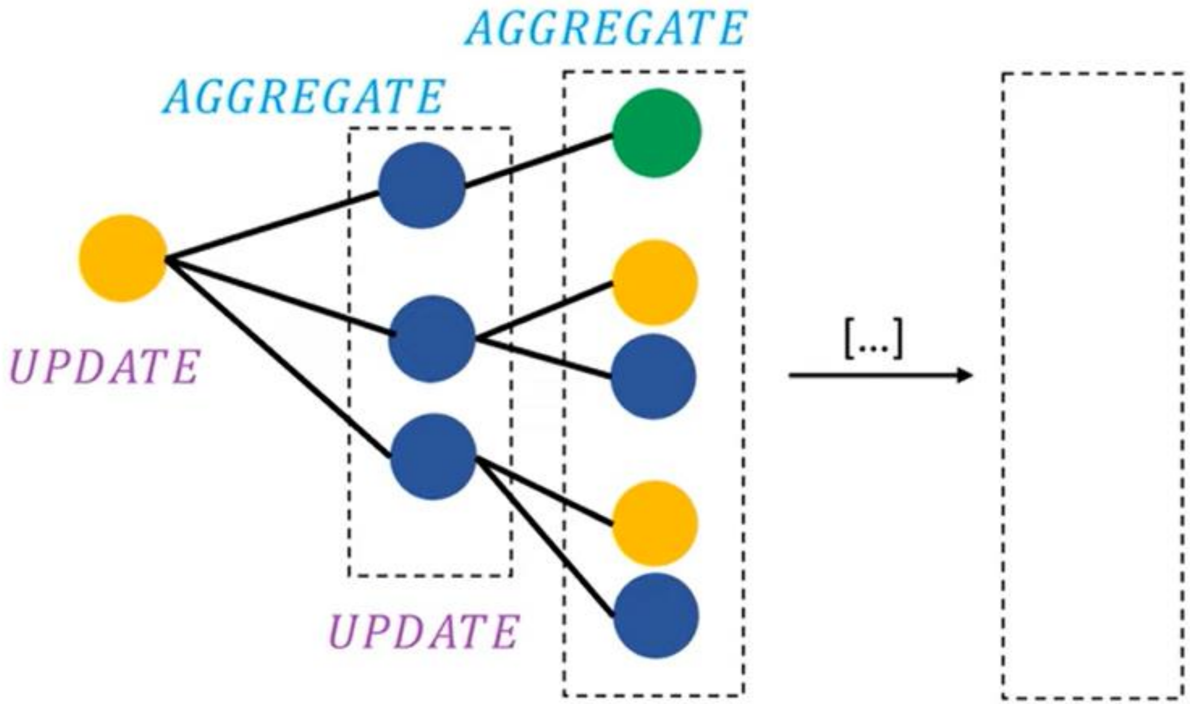
What is happening in the Message Passing Layers?



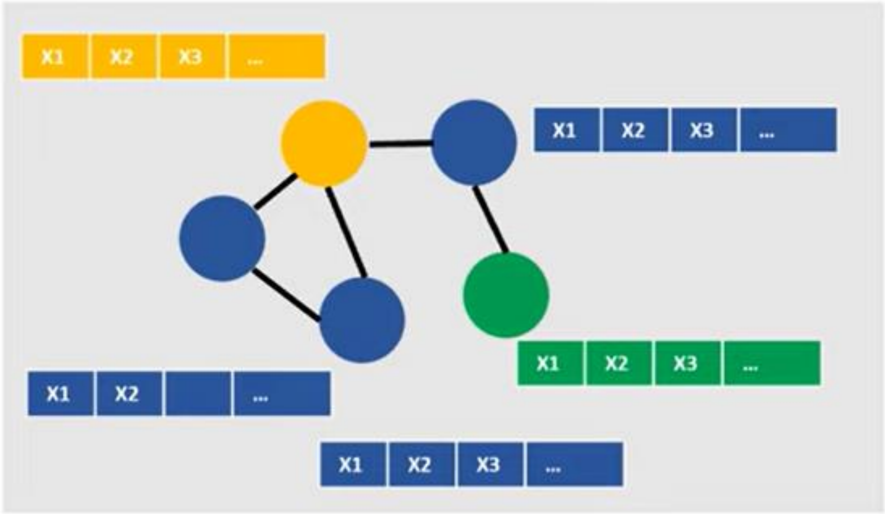
Computation Graph Representation

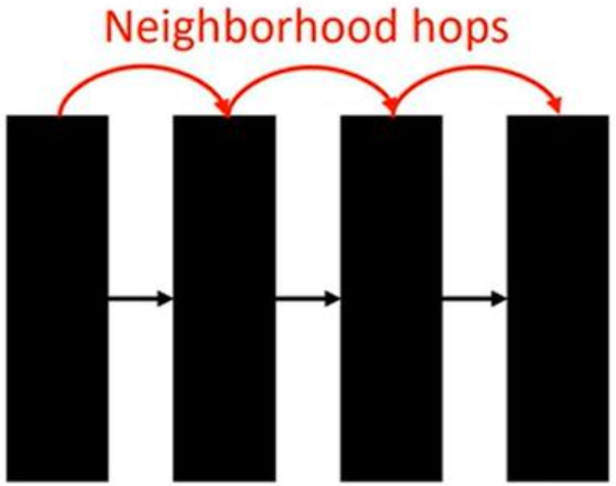


Computation Graph Representation



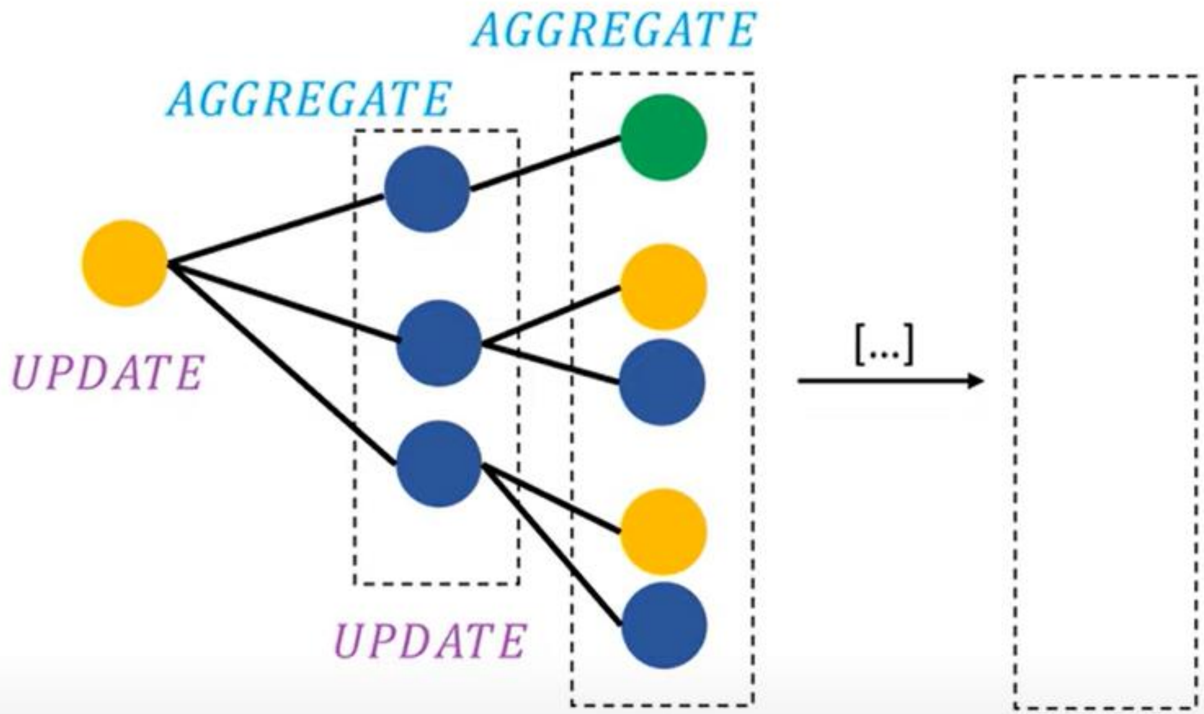
Computation graph for node v

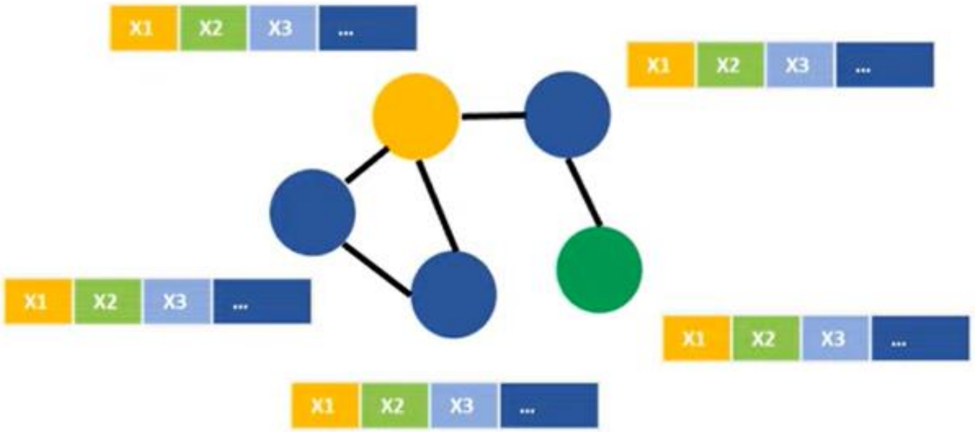




The number of MP-layers is a hyperparameter

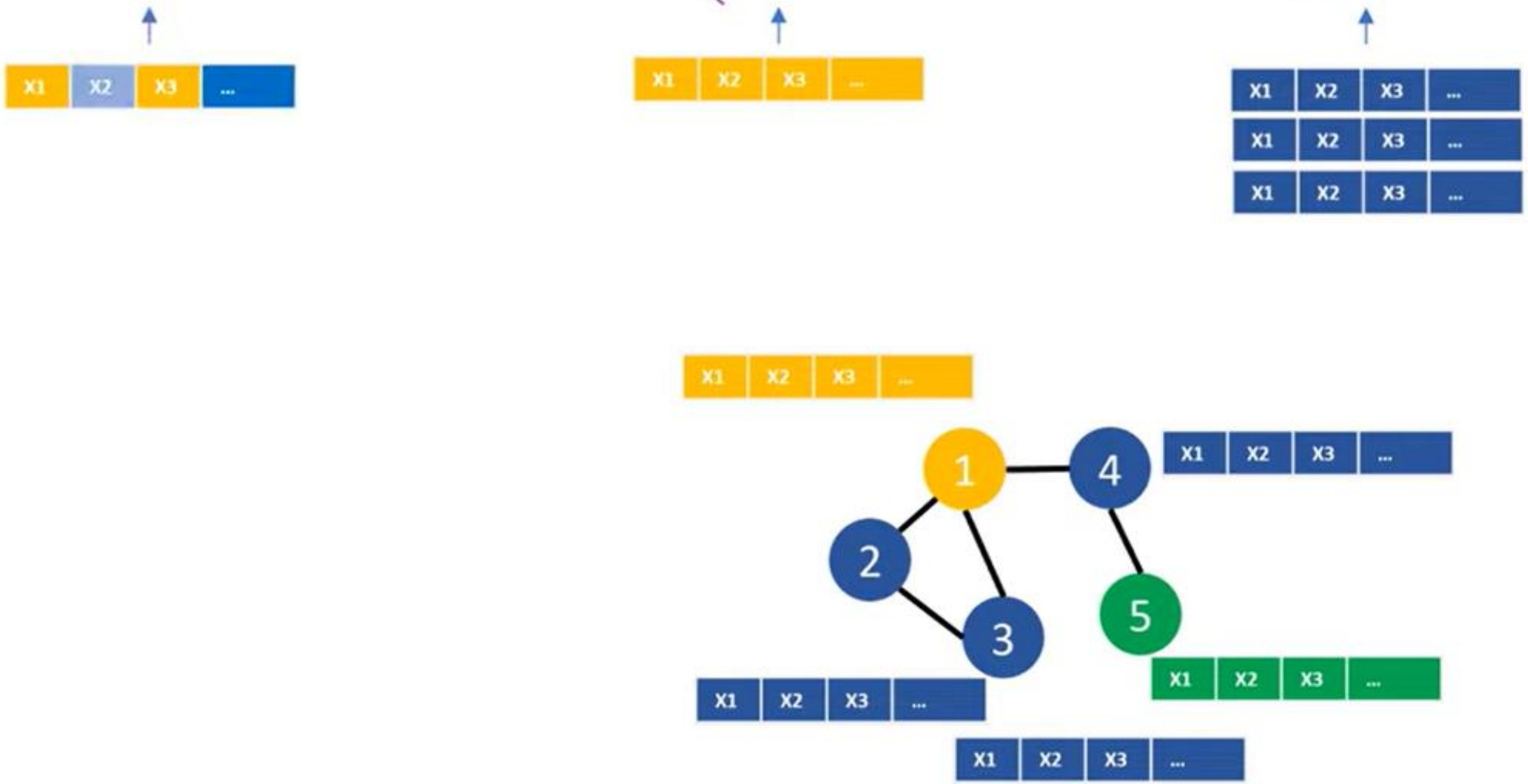
Over-smoothing in GNNs





Message Passing Update and Aggregation Functions

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



Message Passing Update and Aggregation Functions

Differentiable functions

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

- Mean
- Max
- Neural Network
- Recurrent NN

- Mean
- Max
- Normalized Sum
- Neural Network

GNN variants

AGGREGATE
(permutation invariant)

UPDATE

Graph Convolutional Networks, Kipf and Welling [2016]

Self-loop

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}\right)$$

Sum of normalized neighbor embeddings

Multi-Layer-Perceptron as Aggregator, Zaheer et al. [2017]

Aggregated message

$$\mathbf{m}_{\mathcal{N}(u)} = \boxed{\mathrm{MLP}_\theta}\left(\sum_{v \in N(u)} \mathrm{MLP}_\phi(\mathbf{h}_v)\right)$$

trainable!

Send states through a MLP

Graph Attention Networks, Veličković et al. [2017]

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v$$

Attention weights

$$\alpha_{u,v} = \frac{\exp\left(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v]\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}]\right)}$$

Gated Graph Neural Networks, Li et al. [2015]

$$\mathbf{h}_u^{(k)} = \mathrm{GRU}(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)})$$

Recurrent update of the state

Types of Graph Neural Networks

- **Graph Convolutional Networks (GCNs)**: Learn node features by aggregating information from neighboring nodes using graph convolution and activation functions.
- **Graph Auto-Encoder Networks**: Encode and reconstruct graph representations for tasks like link prediction, using an encoder-decoder structure.
- **Recurrent Graph Neural Networks (RGNNs)**: Handle multi-relational graphs and optimize diffusion patterns while reducing computational complexity.
- **Gated Graph Neural Networks (GGNNs)**: Enhance RGNNs with gating mechanisms for better handling of long-term dependencies.

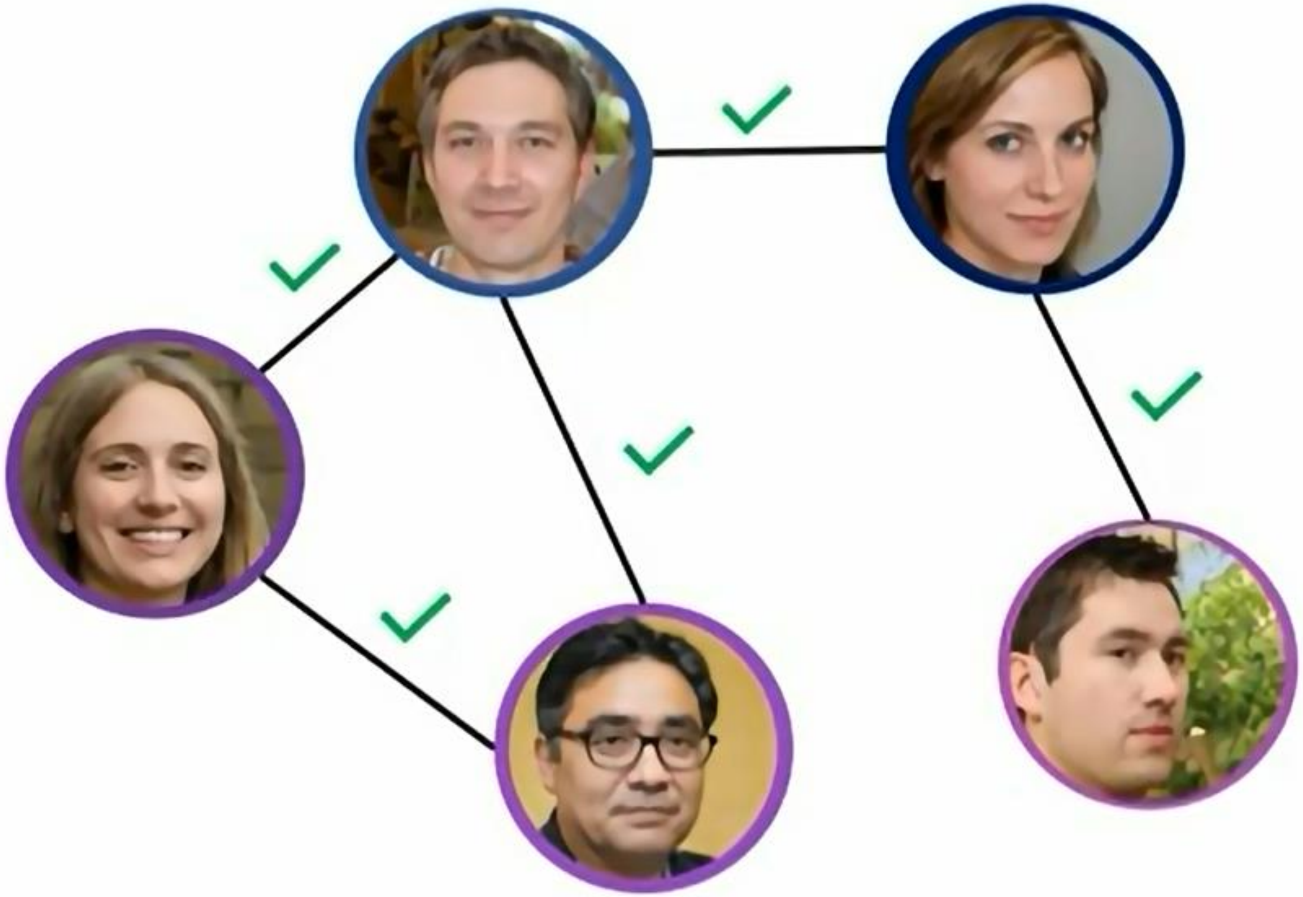
Implementation

- https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00876-4
- Code
- https://colab.research.google.com/drive/16GBgwYR2ECiXVxA1BoLxYshKczNMeEAQ?usp=sharing
- Video
- https://www.youtube.com/watch?v=0YLZXjMHA-8&list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z&index=3

Edge features

Simple binary edge features

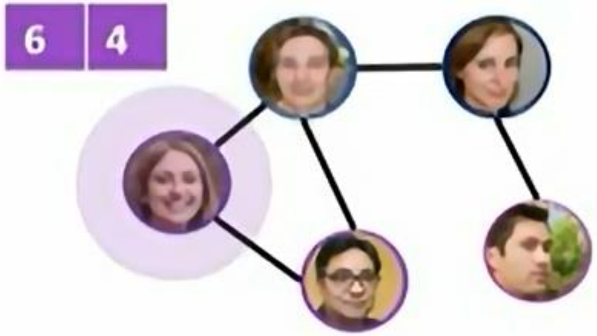
	0	1	0	1	0
	1	0	1	1	0
	0	1	0	0	1
	1	1	0	0	0
	0	0	1	0	0



The general process in GNNs

$$h_{Alice}^{(k+1)} = UPDATE\left(h_{Alice}^{(k)}, AGGREGATE_{j \in \mathrm{N}(Alice)}\, TRANSFORM(h_j^{(k)})\right)$$

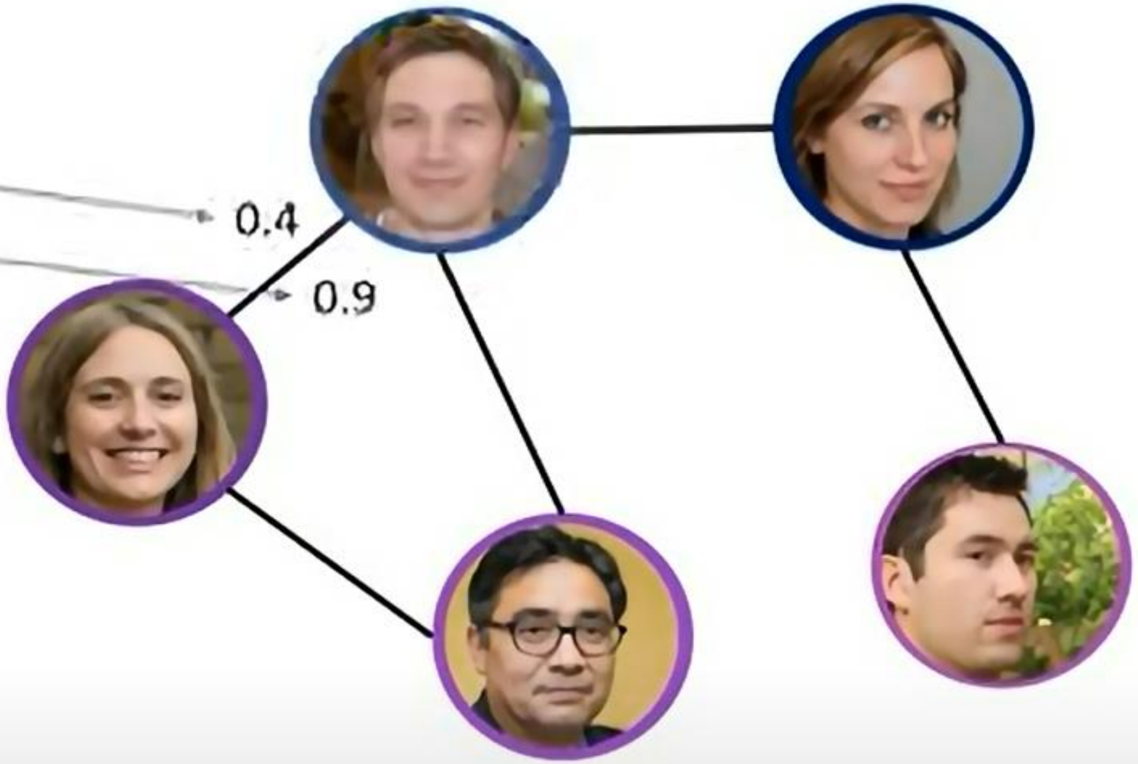
$$h_{Alice}^{(k+1)} = AGGREGATE_{j \in \mathrm{N}(Alice)}\, TRANSFORM(h_j^{(k)})$$



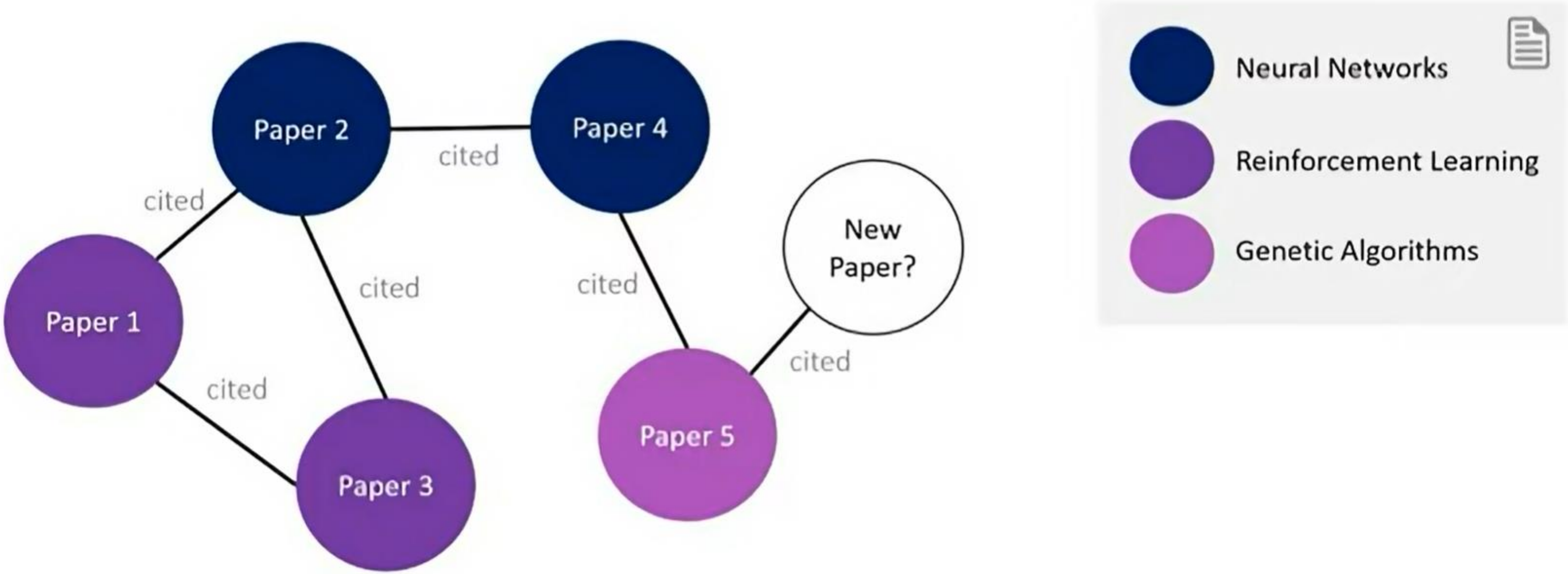
Using edge weights

	0	1	0	1	0
	1	0	1	1	0
	0	1	0	0	1
	1	1	0	0	0
	0	0	1	0	0

	0	0.4	0	0.5	0
	0.9	0	1	1	0
	0	1	0	0	0.5
	1	0.7	0	0	0
	0	0	0.1	0	0

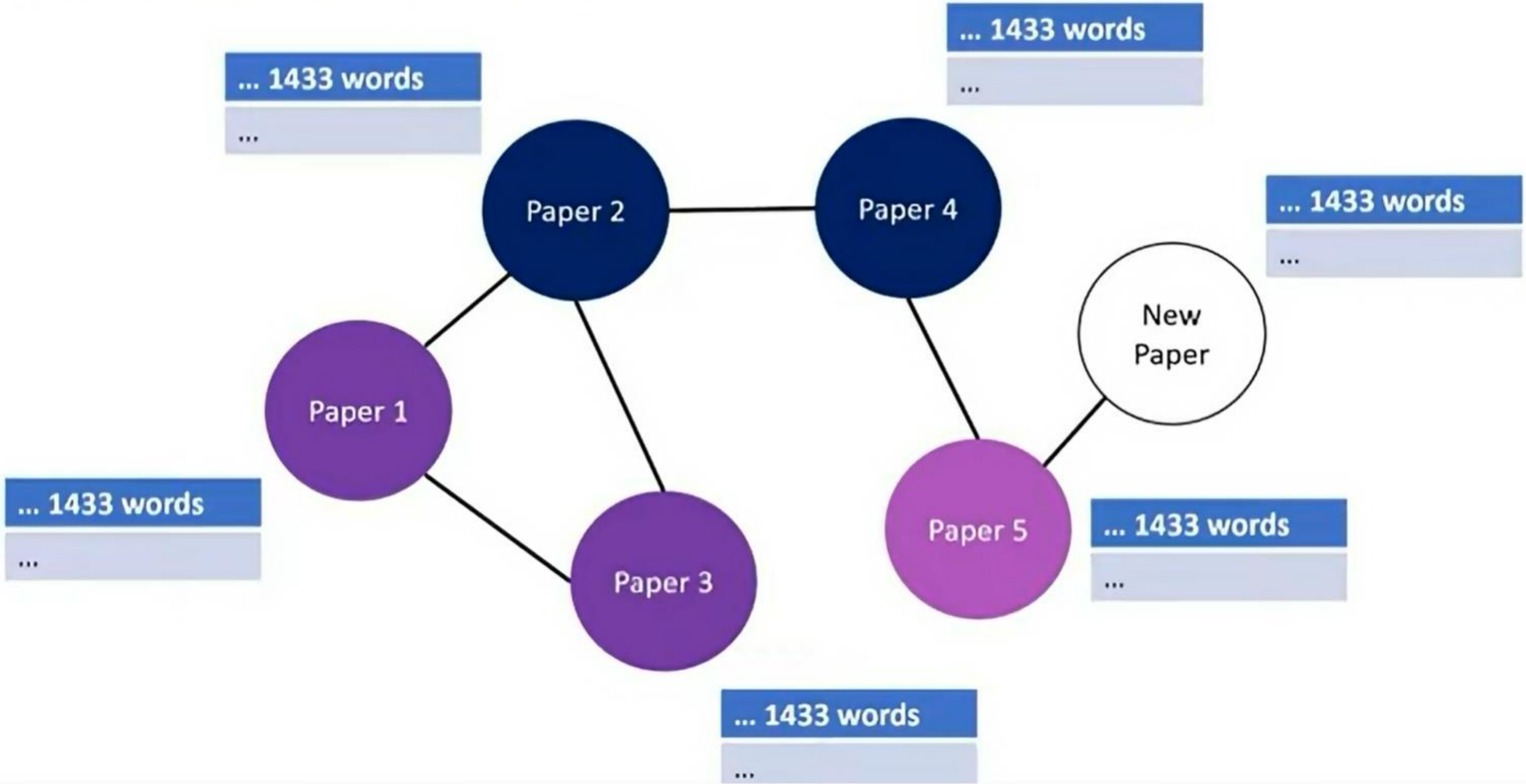


Cora Citation Dataset

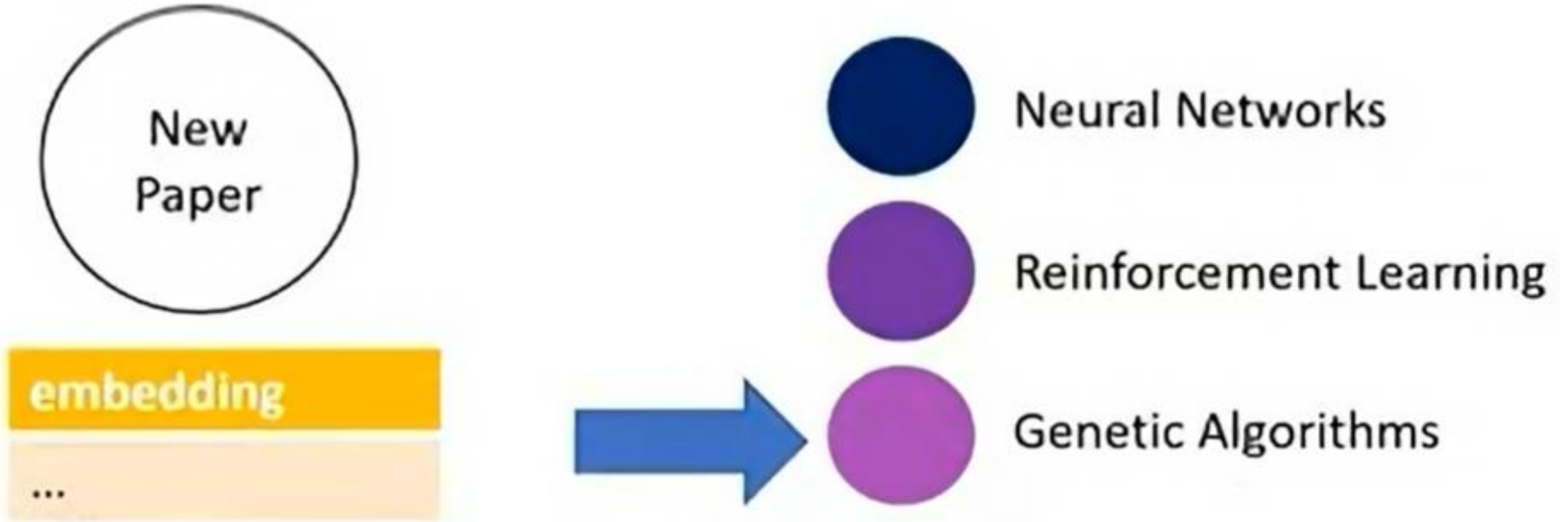


Cora Citation Dataset

Bag of words representation



Representation learning



https://colab.research.google.com/drive/1LJir3T6M6Omc2Vn2GV2cDW_GV2Yfl53_?usp=sharing