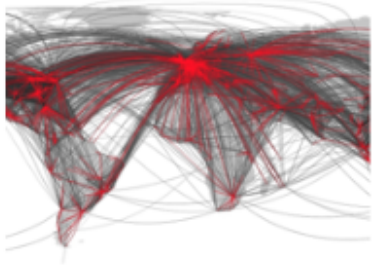
# Graph Neural Networks (GNN)

(Material based on book Hands-on Graph Neural Networks using Python, by Maxime Labonne

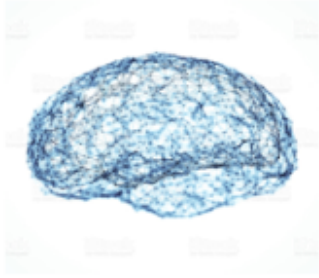
and on GNNs guide

and on A Practical Tutorial on Graph Neural Networks, ACM survey paper)

Social Graphs

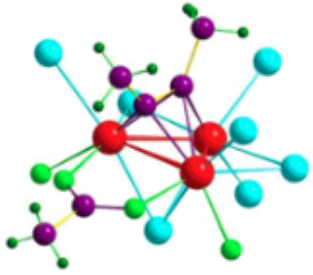


Transportation Graphs

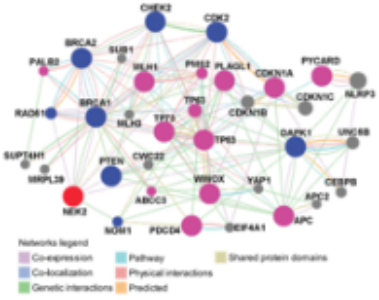


Brain Graphs

Web Graphs



Molecular Graphs



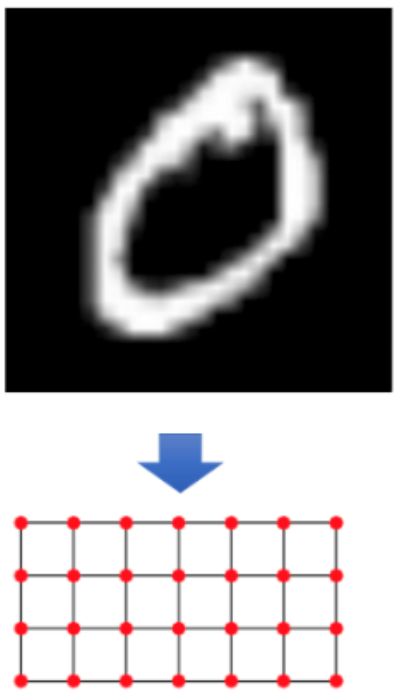
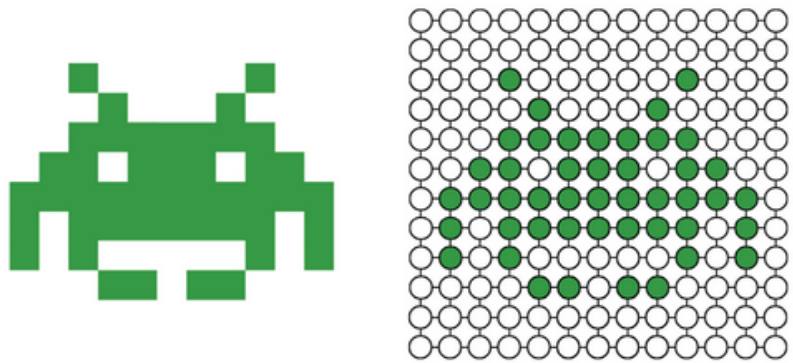
Gene Graphs

(Source: http://cse.msu.edu/ mayao4/tutorials/aaai2020/)

Other useful link: A Gentle Intro to GNNs

# Data as Graphs

Data as Graphs $\rightarrow$ Implicit representation

| Example 1 | Example 2 |
| --- | --- |
|  |  |

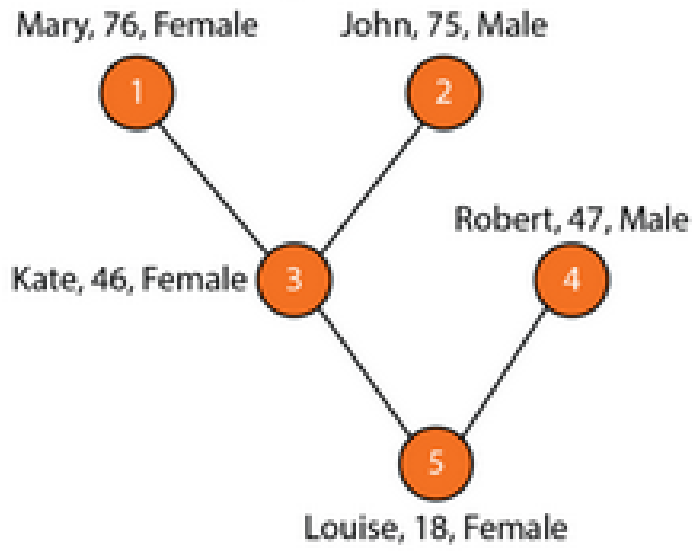
(Source: http://cse.msu.edu/ mayao4/tutorials/aaai2020/)

# Data as Graphs

**Tabular dataset**

| ID | Name | Age | Gender |
|---|---|---|---|
| 1 | Mary | 76 | Female |
| 2 | John | 75 | Male |
| 3 | Kate | 46 | Female |
| 4 | Robert | 47 | Male |
| 5 | Loise | 18 | Female |

**Graph dataset**




Tabular format does not represent relations

# Data as Graphs

Challenges in analyzing a graph:

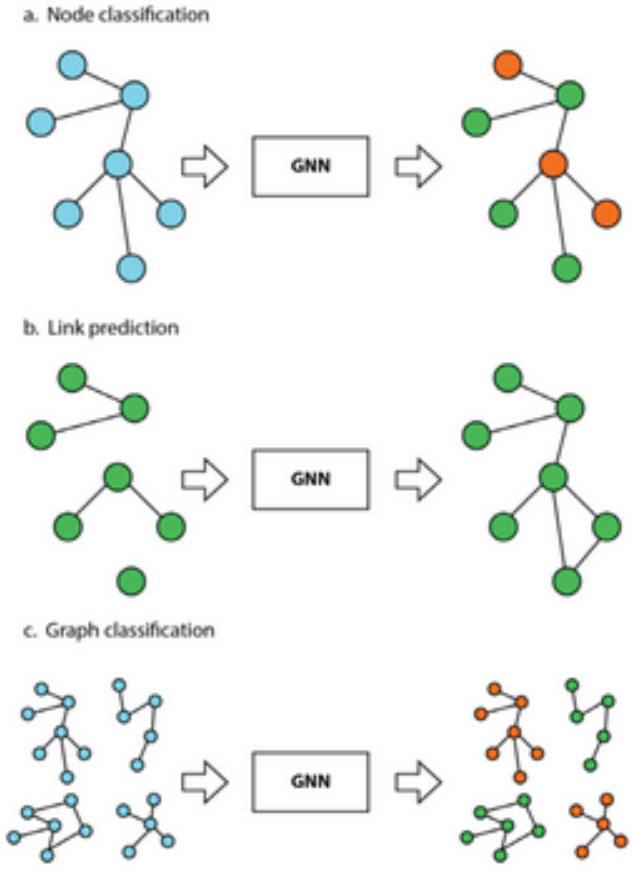
- Graph size is dynamic
- Each node can have a variable number of edges
- Standard methods used for images and texts are not suitable for graphs
- Adjacency matrix representation can be very inefficient
- There can be multiple adjency matrices to represent the same graph
- Standard convolution applied to images does not work (adaptations have been tried)

# Graph Neural Networks

- Proposed to handle graph prediction problems efficiently
- GNN: Graph-in, Graph-Out network
  - It takes the input graph comprising embeddings for edges, nodes and context
  - Generates the output graph transformed and updated embeddings
  - Used for graph-level, node-level and edge-level prediction tasks
- GNN and GCN (Graph Convolutional Network) are used interchangeably

# Graph Neural Networks: potential tasks

- Link prediction
- Node classification
- Community detection
- Ranking
- etc...



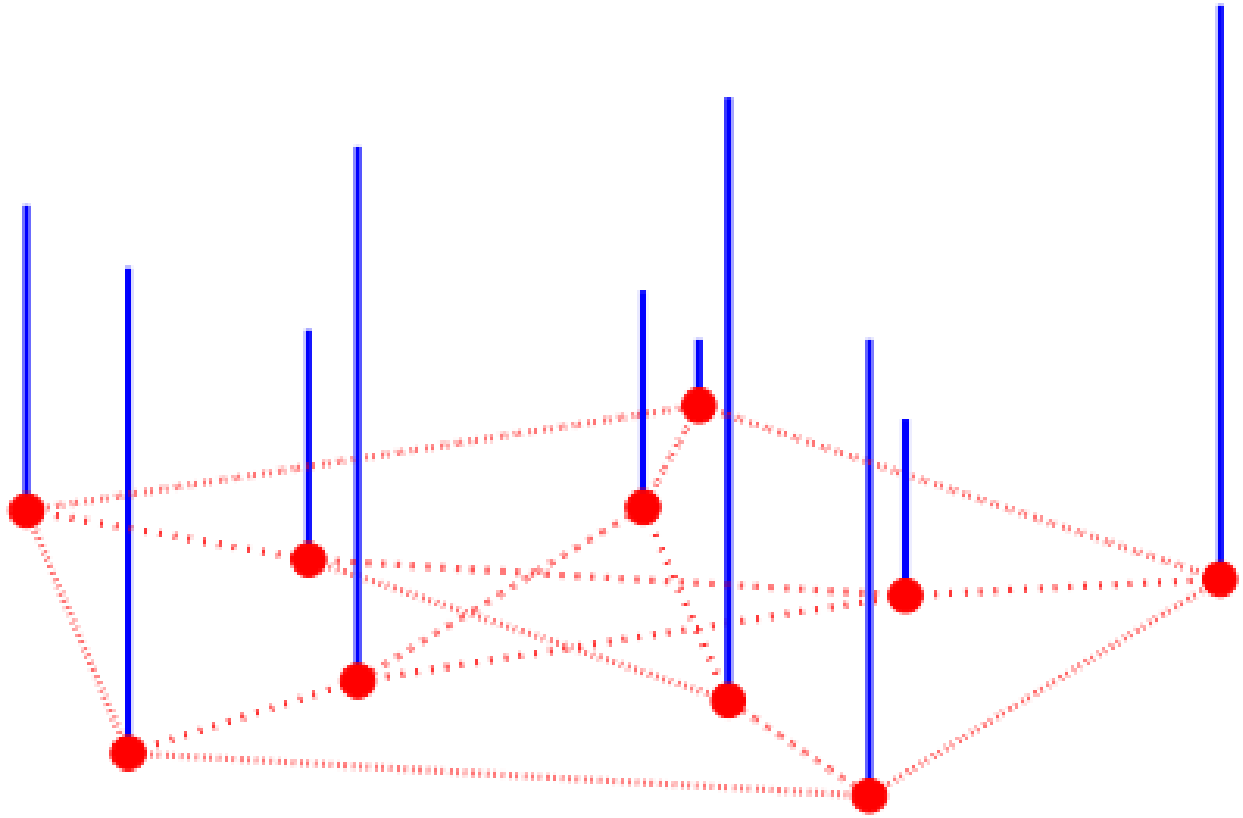
# Graph Representation

- adjacency matrix
- edge list
- adjacency list

# Graph Neural Network Architectures

- Mainly three methods
  - **Spectral**: uses Discrete Fourier Transform (DFT) to transform the graph. Perform graph convolution in the spectral domain.
    → GCN, Chebyshev GCN, Graph Attention Networks (GAT)
  - **Spatial**: applied directly on the neighborhood of each node in the graph (message passing or neighborhood aggregation).
    → GraphSAGE (Graph Sample and Aggregation), Graph Isomorphism Network (GIN)
  - **Sampling**: to handle the scalability issues, use only a subset of nodes instead of all.
    → GraphSAGE, DeepWalk

# Spectral Methods



Source: Intuition behind Laplacian matrix

# Spectral

- Uses DFT on the adjacency matrix to extract structural properties and patterns within the graph
  - Graph Spectra
  - Frequency Components
  - Graph Embeddings
  - Graph Compression
  - Graph Filtering

# Graph Spectra

Spectral properties that encode information about the graph's connectivity, community structure, and other structural characteristics. Eigenvectors represent the graph frequencies and eigenvalues, the most common structures or motifs

# Frequency components

Similar to signals, graphs can exhibit certain frequency components that correspond to patterns of connectivity or motifs within the graph. The DFT can identify these frequency components by decomposing the graph into its constituent sinusoidal components.

# Graph Embeddings

By transforming the graph's adjacency matrix using the DFT, it's possible to obtain embeddings of the graph vertices in a lower-dimensional space. These embeddings capture structural information about the graph and can be used for tasks such as graph classification, clustering, or visualization.

# Graph Compression

The DFT can be used for graph compression by retaining only the most significant frequency components while discarding the higher-frequency noise. This can lead to more compact representations of large graphs while preserving important structural information.

# Graph Filtering

Just as in signal processing, the DFT can be used for filtering operations on graphs. By selectively removing or attenuating certain frequency components, it's possible to denoise the graph or highlight specific structural patterns of interest.

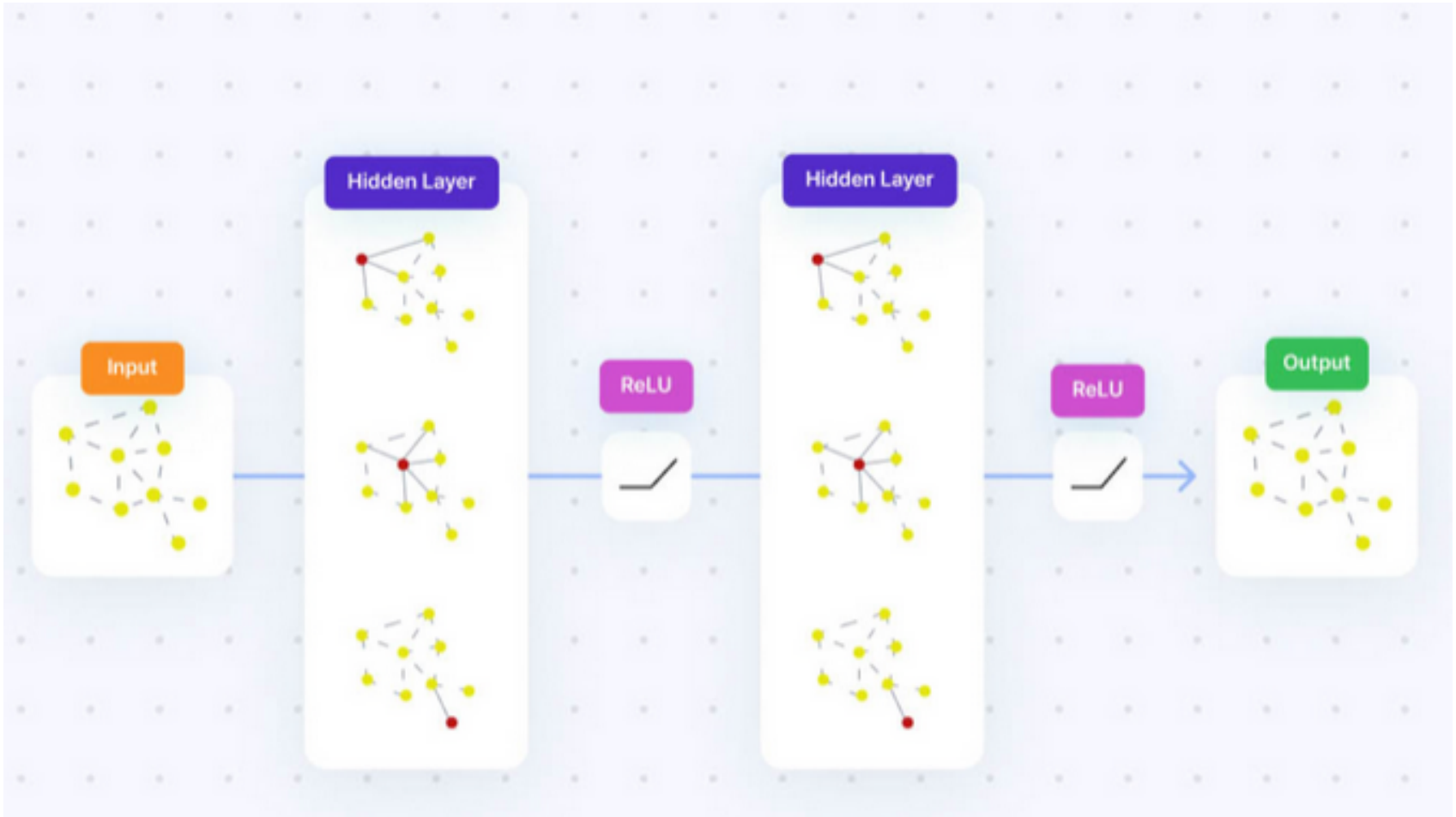
# Spectral Networks

- SCNN (Spectral-based Convolutional Neural Networks)
  - learns convolutional filters for graph prediction tasks
  - advantage: kernel is learnable
  - disadvantages:
    - computationally inefficient for large graphs because of multiple matrices multiplications
    - number of parameters in the kernel depends on the number of nodes in the graph
    - filter is applied to the whole graph, difficult to obtain local info

# Spectral Networks

- GCN (Graph Convolutional Neural Networks)
  - simple, scalable, more computationally efficient
  - simple arch: conv layer, linear layer, non-linear activation layer
  - disadvantages: do not support edge features (message passing between nodes)

Note: can also be spatial-based



# Spectral Methods Disadvantages

- not suitable for undirected graphs
- graph structure can not be updated during training
- computationally more intensive than spatial methods

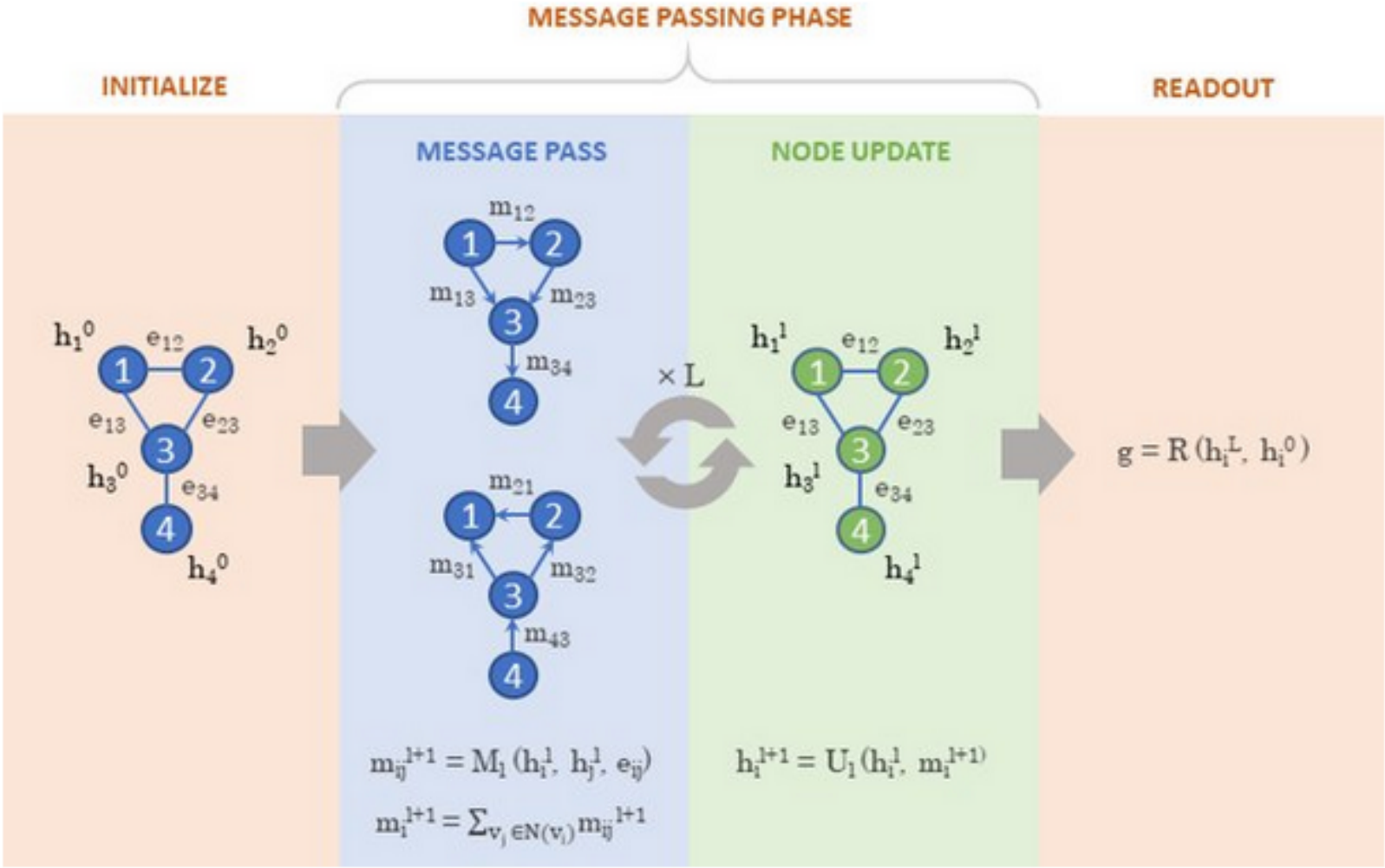
# Spatial Methods

- Spatial methods follow a standard approach of graph convolution acting directly over the graphs nodes and edges.

# Spatial Methods: Message Passing (MPNN)

- First introduced by Gilmer et al. (first publication with an application in quantum chemistry, ICML 2017)
- Used to propagate information between nodes in the graph (for example, neighborhood or individual node features)
- Goal: learn a graph representation vector via a neighborhood aggregation scheme of node states and edge states

# Spatial Methods: Message Passing (MPNN) abstract scheme



# Message Passing (MPNN) readout functions

- **Sum Readout**: The simplest form of readout function is to compute the sum of all node representations. Mathematically, the graph-level representation $h_G$ can be computed as:

$$h_G = \sum_{i=1}^{N} h_i^T$$

Where $N$ is the number of nodes in the graph, and $h_i^T$ represents the final representation of node $i$ after $T$ message passing iterations.

# Message Passing (MPNN) readout (R) functions

- **Mean Readout**: mean (average) of all node representations. Mathematically, the graph-level representation $h_G$ can be computed as:

$$h_G = \frac{1}{N} \sum_{i=1}^{N} h_i^T$$

Where $N$ is the number of nodes in the graph, and $h_i^T$ represents the final representation of node $i$ after $T$ message passing iterations.

# Message Passing (MPNN) readout functions

- **Pooling Readout**: pooling operations such as max pooling or attention-based pooling. In max pooling, the graph-level representation is computed by taking the maximum value across all dimensions of the node representations. Attention-based pooling uses learned attention weights to dynamically weight the contributions of different nodes to the graph-level representation.
- **Graph-Level MLP**: Instead of using a predefined readout function, can also use a graph-level multi-layer perceptron (MLP) to learn a more complex function that maps the node representations to the graph-level representation. This allows the model to capture more intricate relationships and patterns in the graph.

# Spatial Methods: Graph Attention Networks (GAT)

- GAT introduces the concept of attention mechanism in GNs
- In typical algorithms, the same convolutional kernel parameters are applied over all nodes of the graph
- GAT allows for different convolutional parameters $\rightarrow$ can help adjusting the degree of association between nodes and determining the corresponding importance of nodes

# Spatial Methods: Graph Attention Networks (GAT)

- Attention coefficients are calculated by passing node or edge features into an attention function
- Softmax is applied over the obtained value to give the final weights
- GAT allows for different convolutional parameters $\rightarrow$ can help adjusting the degree of association between nodes and determining the corresponding importance of nodes

# Spatial Methods: Graph Attention Networks (GAT)

Attention function:

$$a_{ij} = attention(h_i, h_j) = \frac{exp(a_{ij})}{\sum_{k \in N_i} exp(a_{ik})}$$

Update rule (aggegation process):

$$\vec{h}'_i = \sigma(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in N_i} \alpha_{ij}^k W^k \vec{h}_j)$$

# MPNN x GAT

**Message Passing Neural Networks (MPNNs)**

- Node representations are typically updated through an iterative message passing process.
- At each iteration, each node aggregates information from its neighboring nodes to compute a message, which is then used to update its representation.
- The update function in MPNNs combines the node's current representation with the aggregated messages from its neighbors.
- The update process is typically performed for a fixed number of iterations, allowing nodes to integrate information from their local neighborhood and capture both local and global structural information.

# MPNN x GAT

## Graph Attention Networks (GATs)

- Attention mechanisms are used to update node representations in a non-linear and adaptive manner.
- Instead of aggregating information from all neighboring nodes equally, GATs compute attention coefficients that determine the importance of each neighbor's contribution to the node's representation.
- The attention coefficients are computed based on learnable parameters and the similarity between the node's features and its neighbors' features.
- The updated representation of each node is a weighted sum of the representations of its neighbors, where the weights are determined by the attention coefficients.
- The attention mechanism allows GATs to dynamically focus on different parts of the graph and adaptively aggregate information from neighboring nodes based on the task requirements and the structure of the graph.

# MPNN x GAT

In summary, while both MPNNs and GATs update node representations through information aggregation, MPNNs use a fixed aggregation scheme based on message passing iterations, whereas GATs employ attention mechanisms to compute adaptive and context-aware updates based on the relationships between nodes.

# Sampling methods

- As graphs get larger, aggregating features from all neighboring nodes would be computationally inefficient.
- Sampling methods downsize the graphs allowing for a more efficient analysis and modeling

# Sampling methods: GraphSAGE

- Uniform sampling on nodes would be computationally inefficient.
- Extends the neighborhood depth k on each layer
- Learns feature information from k nodes away with every additional layer

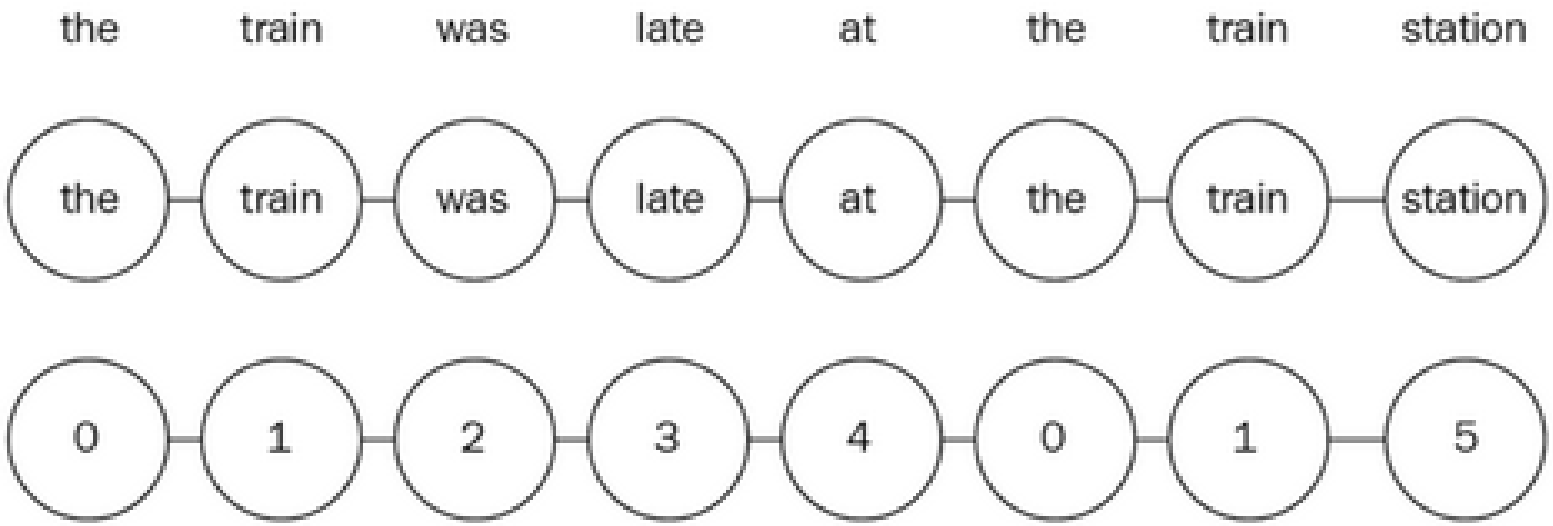

1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

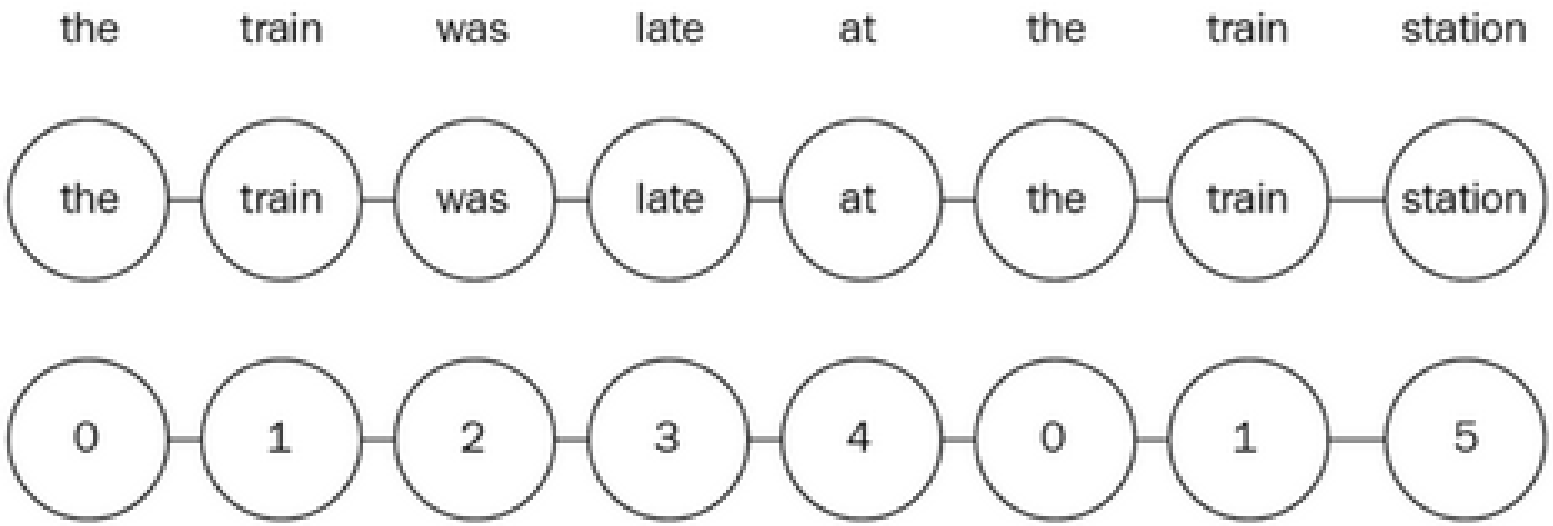
# Sampling methods: DeepWalk

- Main goal: to produce high-quality feature representations of nodes in an unsupervised way
- Heavily inspired by Word2Vec in NLP
- Use random walks to generate meaningful sequences of nodes that act like sentences



# Sampling methods: DeepWalk

- Use a skip-gram model commonly used to learn word embeddings to learn node embeddings
- skip-gram attempts to maximize the similarity of the nodes that occur in the same random walk
- A set of random vectors are generated for each node using the skip-gram method
- Gradient descent is applied to these vectors to update the node embeddings and maximize the probability of the neighboring nodes given a node by using a softmax function



# Sampling methods: DeepWalk (Perozzi et al. )

Works in 2 stages:

- Stage 1: Random Walks
  1. Repeat k times:
     - 1.1 A node is selected randomly
     - 1.2 Of all its neighboring nodes, another one is selected randomly
     - 1.3 Repeat till sequence length l is reached
- Stage 2: Skip-gram model

# To learn more

- Graph Neural Networks - tutorials and resources
- Colab notebooks and tutorials using pytorch geometric