

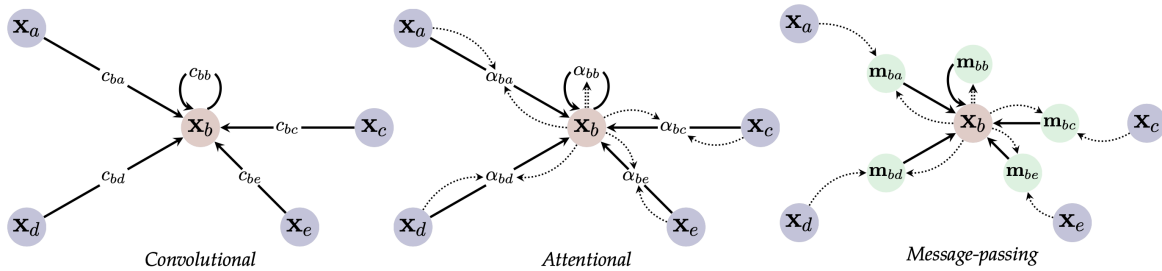
Standard architectures of GNNs

Let $h_i^{(l)} \in \mathbb{R}^d$ denote a d -dimensional embedding of node i at layer $l \in \{0, \dots, L\}$, where L is the total number of layers of the graph neural network. The initial embedding of node i is simply its feature vector,

$$h_i^{(0)} = x_i,$$

while the embedding in the final layer L can be used for a downstream prediction task,

$$z_i = h_i^{(L)}.$$



Graph convolutional network (GCM)

Recall the update procedure of the *graph convolutional network* ([Kipf & Welling, 2017](#)).

$$h_i^{(l+1)} = \sigma \left(\overline{W}^{(l)} h_i^{(l)} + W^{(l)} \sum_{j \in \Gamma_i} \frac{h_j^{(l)}}{k_i} \right)$$

Graph attention network (GAT)

In the graph convolutional network, all neighbors Γ_i of node i are treated as equally important. In contrast, the *graph attention network* ([Veličković et al., 2018](#)), inspired by the concept of cognitive attention, assigns different importance to the neighbors through a learned attention mechanism.

In particular, the update procedure is defined as a weighted sum

$$h_i^{(l+1)} = \sigma \left(W^{(l)} \sum_{j \in \Gamma_i} \alpha_{ij} h_j^{(l)} \right),$$

where $W^{(l)} \in \mathbb{R}^{d \times d}$ are learnable weight matrices and $\alpha_{ij} \in \mathbb{R}$ are the attention weights. The attention weights are computed by a learnable attention function $\text{ATTENTION}^{(l)}$ that first computes the attention coefficients $e_{ij} \in \mathbb{R}$ as

$$e_{ij} = \text{ATTENTION}^{(l)} \left(W^{(l)} h_i^{(l)} \parallel W^{(l)} h_j^{(l)} \right),$$

where $\parallel$ denotes concatenation, followed by a softmax normalization as

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{j \in \Gamma_i} \exp e_{ij}}.$$

Note that the above formulation is agnostic to the specific choice of the attention function $\text{ATTENTION}^{(l)}$. However, a single-layer or multilayer perceptron is most commonly used in practice.

To stabilize the learning process of the attention mechanism, one can use multi-head attention with heads $a \in \{1, 2, \dots\}$.

$$e_{ij}^{[a]} = \text{ATTENTION}_{[a]}^{(l)} \left(W^{(l)} h_i^{(l)} \parallel W^{(l)} h_j^{(l)} \right) \quad \& \quad \alpha_{ij}^{[a]} = \frac{\exp e_{ij}^{[a]}}{\sum_{j \in \Gamma_i} \exp e_{ij}^{[a]}}$$

Then, the update procedure can be defined as a summation over all heads a , although other choices are possible (e.g., concatenation followed by a single-layer or multilayer perceptron).

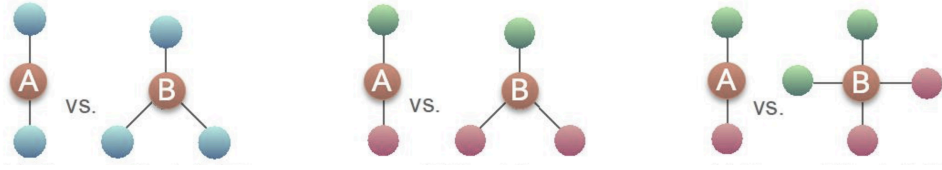
$$h_i^{(l+1)} = \sigma \left(W^{(l)} \sum_a \sum_{j \in \Gamma_i} \alpha_{ij}^{[a]} h_j^{(l)} \right)$$

Graph isomorphism network (G/IN)

A classical way to assess the expressive power of graph neural networks is through the graph isomorphism problem. Graph isomorphism remains a long-standing open problem, since no polynomial-time algorithm is known, yet the problem has not been proven to be *NP*-complete. The Weisfeiler-Lehman kernel provides a powerful heuristic approach for distinguishing non-isomorphic graphs and serves as a theoretical benchmark for evaluating the expressivity of graph neural networks.

A maximally expressive graph neural network should be an injective function that maps distinct node neighborhoods Γ to distinct embeddings h . Among common neighborhood aggregation functions, only summation is injective in general, whereas mean and maximum aggregations can fail to distinguish certain neighborhood structures. In general, summation

provides the most expressive aggregation, followed by mean, with maximum being the least expressive.

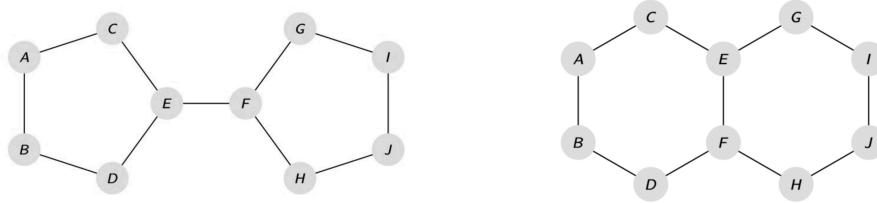


The *graph isomorphism network* ([Xu et al., 2019](#)) is specifically designed to provably match the expressive power of the Weisfeiler–Lehman kernel. The update procedure is defined as

$$h_i^{(l+1)} = \text{MLP}^{(l)} \left((1 + \epsilon^{(l)}) \cdot h_i^{(l)} + \sum_{j \in \Gamma_i} h_j^{(l)} \right),$$

where $\epsilon^{(l)} \in \mathbb{R}^+$ are learnable parameters and $\text{MLP}^{(l)}$ are multilayer perceptrons with a couple of hundred layers.

Note that certain graphs remain indistinguishable even by the graph isomorphism network or the Weisfeiler-Lehman kernel.

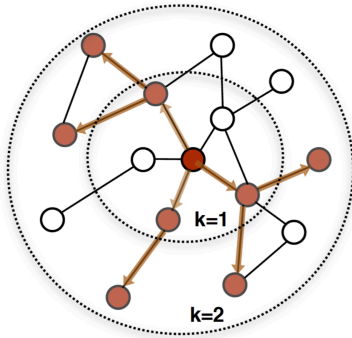


Graph "sample and aggregate" network (*GraphSAGE*)

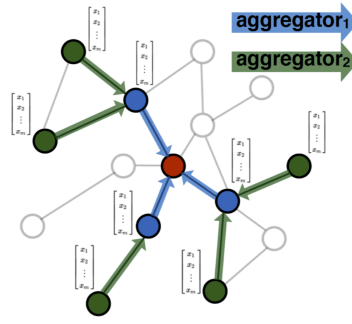
The *graph "sample and aggregate" network* ([Hamilton et al., 2017](#)) is a widely used graph neural network architecture that introduced several key design ideas. For each node i , its neighbors Γ_i are first randomly sampled and aggregated using either summation, mean, or maximum, optionally followed by a single-layer or multilayer perceptron. Alternatively, one can also apply sequence models, such as *long short-term memory (LSTM)* architecture, over a randomly reshuffled neighborhood Γ_i .

For instance,

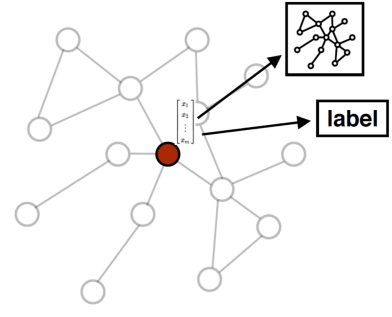
$$\text{AGGREGATE} \left(\{ \{ h_j^{(l)} \mid j \in \Gamma_i \} \} \right) = \sum_{j \in \Gamma_i} \frac{h_j^{(l)}}{k_i}.$$



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

Then, the update procedure is defined as

$$h_i^{(l+1)} = \sigma \left(W^{(l)} \left(h_i^{(l)} \parallel \text{AGGREGATE} \left(\{ \{ h_j^{(l)} \mid j \in \Gamma_i \} \} \right) \right) \right),$$

where $W^{(l)} \in \mathbb{R}^{d \times 2d}$ are learnable weight matrices.

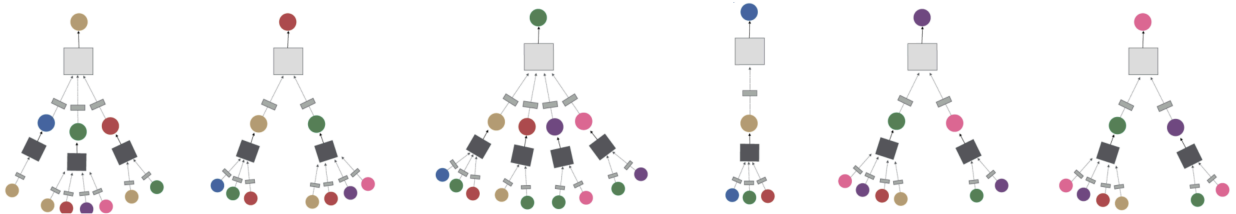
The *GraphSAGE* architecture also normalizes the embeddings at every layer as

$$h_i^{(l+1)} \leftarrow \frac{h_i^{(l+1)}}{\|h_i^{(l+1)}\|_2},$$

which can lead to improved performance in certain cases.

Message passing framework for GNNs

The *GraphSAGE* paper also introduced the *message passing framework* for graph neural networks, based on the concept of a computation graph.



1. Message computation

First, each node i sends a message $m_i^{(l)}$ to all its neighbors Γ_i . This message can simply be the node's current embedding,

$$m_i^{(l)} = \text{MESSAGE}^{(l)} \left(h_i^{(l)} \right) = h_i^{(l)},$$

or it can be transformed by a single-layer or multilayer perceptron, or other learnable function.

2. Neighborhood aggregation

Then, each node i aggregates the messages of its neighbors Γ_i . For example,

$$m_{\Gamma_i}^{(l)} = \text{AGGREGATE}^{(l)} \left(\{ \{ m_j^{(l)} \mid j \in \Gamma_i \} \} \right) = \sum_{j \in \Gamma_i} \frac{m_j^{(l)}}{k_i}.$$

3. Update function

Finally, each node i updates its embedding by combining its own message $m_i^{(l)}$ with the aggregated message from its neighbors $m_{\Gamma_i}^{(l)}$. For example,

$$h_i^{(l+1)} = \text{UPDATE}^{(l)} \left(m_i^{(l)}, m_{\Gamma_i}^{(l)} \right) = \sigma \left(W^{(l)} \left(m_i^{(l)} \parallel m_{\Gamma_i}^{(l)} \right) \right).$$

Standard graph neural network architectures can be expressed within the message passing framework, which simplifies their comparison. However, it is not always straightforward to identify which components correspond to the message, aggregation, and update functions. Even the simplest graph convolutional network can be formulated in multiple equivalent ways.