

Graph neural networks

Let A be the adjacency matrix of a simple undirected graph, and let D be the diagonal degree matrix whose entries are the node degrees.

$$A = A^T \in \{0, 1\}^{n \times n}$$

$$D = \text{diag}(k_1, \dots, k_n) \in \mathbb{N}^{n \times n}$$

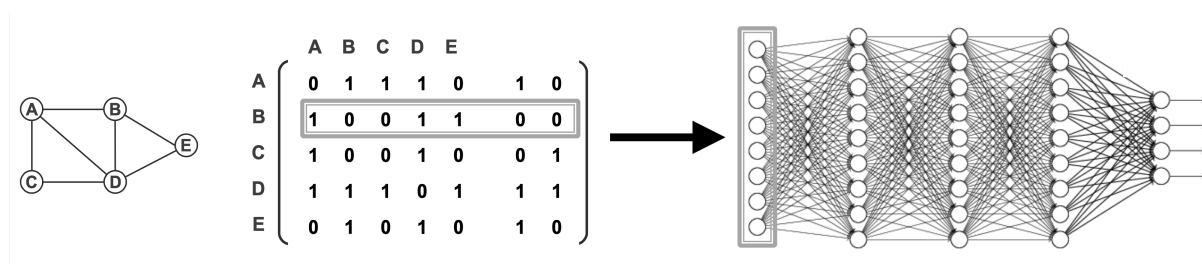
Additionally, let each node i be associated with a d -dimensional feature vector x_i . These features may represent, for example, user metadata or profile image in social networks, gene expression profiles in biological networks, or web page content in information networks.

$$X \in \mathbb{R}^{n \times d}$$

If no node features are available, x_i can be defined either as a one-hot encoding of node i (transductive setting) or as a constant vector of ones (inductive setting). Note that in the former case, the feature vectors are n -dimensional, which is practical only for very small graphs.

Naive approach to graph learning

A naive approach to graph learning is to concatenate the adjacency matrix A and the feature matrix X , thereby representing each node with an $(n + d)$ -dimensional vector that encodes both its graph structure and associated features. These vectors can then be used as input to a deep neural network f .



This approach has several significant drawbacks. Firstly, the neural network f can only be applied in a transductive setting to graphs of the same size and feature dimensionality. Secondly, it is sensitive to the ordering of nodes. Thirdly, and most importantly, each learning

instance is represented by $n + d$ features (i.e. adjacency plus metadata), while there are only n training instances (i.e. nodes), leading to an unfavorable feature-to-sample ratio.

What we need is for the neural network f to be permutation invariant and equivariant.

Permutation invariance

Consider a graph-level task in which the goal is to obtain a d -dimensional embedding of the entire graph. A neural network $f : \{0, 1\}^{n \times n} \times \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^d$ is *permutation invariant* if, for any permutation matrix $P \in \{0, 1\}^{n \times n}$ with exactly one 1 in each row and column and zeros elsewhere, it satisfies

$$f(A, X) = f(PAP^T, PX).$$

In simple terms, if we permute the nodes of the graph, the output of f must remain unchanged.

For example, a simple summation of all nodes' features $\sum_i x_i$ or, equivalently, in matrix form

$$f(A, X) = 1^T X$$

is permutation invariant, since for any permutation matrix P ,

$$f(PAP^T, PX) = 1^T PX = 1^T X = f(A, X).$$

Permutation equivariance

Consider a node-level task in which the goal is to obtain a d -dimensional embedding of each node in the graph. A neural network $f : \{0, 1\}^{n \times n} \times \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times d}$ is *permutation equivariant* if, for any permutation matrix P , it satisfies

$$Pf(A, X) = f(PAP^T, PX).$$

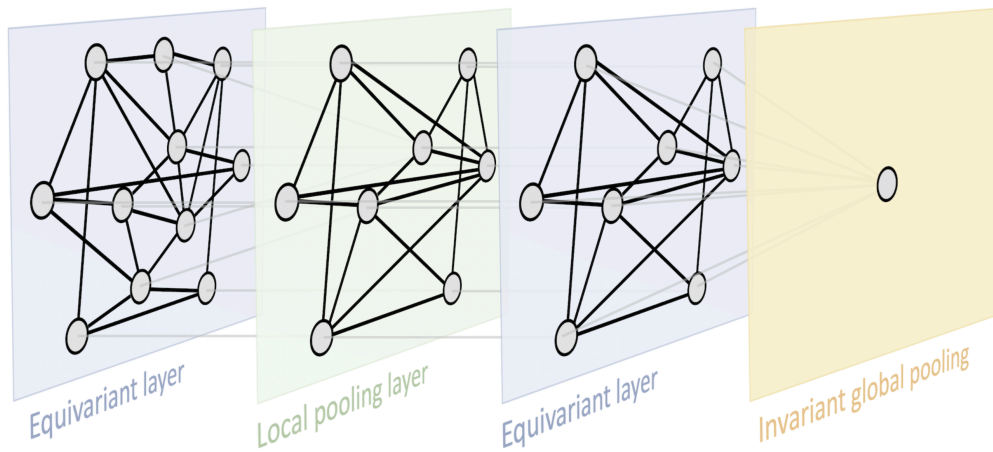
In simple terms, permuting the nodes of the graph results in the same permutation of the output of f .

For example, a simple summation of a node's neighbors' features $\sum_{j \in \Gamma_i} x_j$ or, equivalently, in matrix form

$$f(A, X) = AX$$

is permutation equivariant, since for any permutation matrix P ,

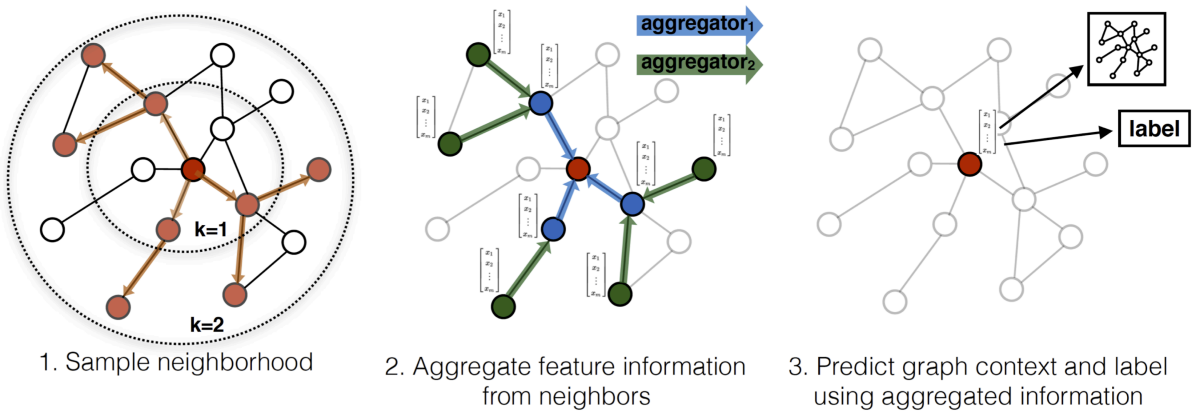
$$f(PAP^T, PX) = PAP^T PX = PAX = Pf(A, X).$$



Permutation invariance and equivariance are thus inherently built into the architecture of modern graph neural networks. Typically, a graph neural network consists of a few permutation equivariant layers, followed by a final permutation invariant layer, often referred to as a *global pooling layer*. Additionally, *local pooling layers* can be incorporated throughout the network to aggregate information hierarchically.

Graph convolutional network

The *graph convolutional network (GCN)* represents the foundational architecture of graph neural networks, as introduced in the seminal paper ([Kipf & Welling, 2017](#)).



Let $h_i^{(l)} \in \mathbb{R}^d$ denote a d -dimensional representation or embedding of node i at layer $l \in \{0, \dots, L\}$, where L is the total number of layers. In practice, graph neural networks typically use only a few layers, such as $L \in \{2, 3, 4\}$, for reasons that will become clear later.

Initial representation

Initially, the representation of node i is simply its feature vector x_i ,

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

Hidden representations

At each layer l , the representation $h_i^{(l)}$ of node i is updated as

$$h_i^{(l+1)} = \sigma \left(\overline{W}^{(l)} h_i^{(l)} + W^{(l)} \sum_{j \in \Gamma_i} \frac{h_j^{(l)}}{k_i} \right), \forall l \in \{0, \dots, L-1\}.$$

This update can be interpreted as follows.

1. Node i updates its own representation $h_i^{(l)}$ by a learnable weight matrix $\overline{W}^{(l)}$ (essentially a single-layer perceptron).
2. Node i also averages the representations of its neighbors $\sum_{j \in \Gamma_i} h_j^{(l)} / k_i$ and transforms them via a learnable weight matrix $W^{(l)}$.
3. The two contributions 1. and 2. are summed together and passed through a non-linear activation function σ , producing the updated representation $h_i^{(l+1)}$.

Final representation

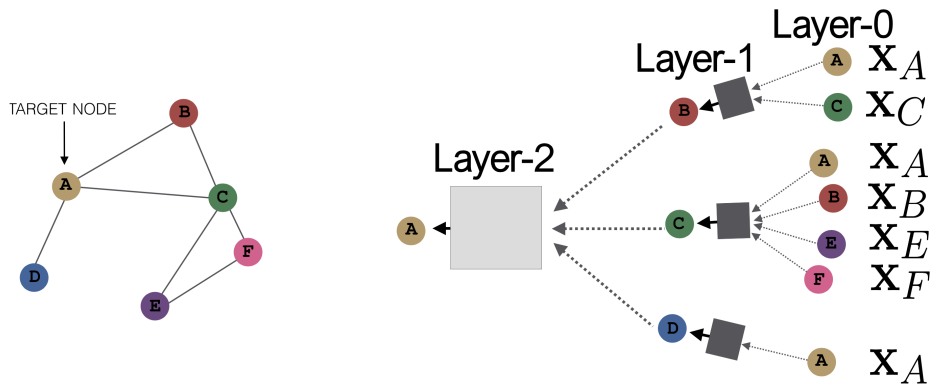
After L layers, the final embedding z_i of node i is

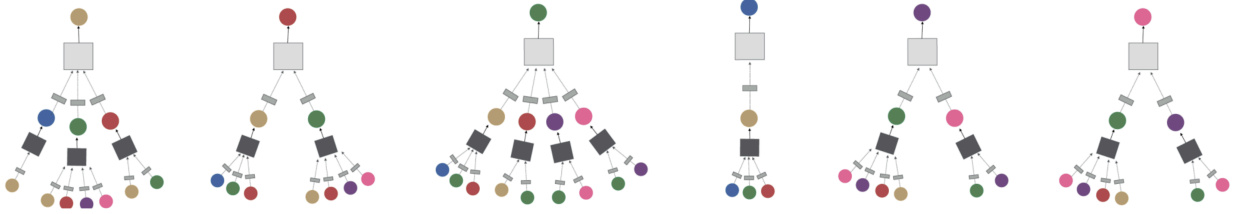
$$z_i = h_i^{(L)},$$

which can be directly optimized for a downstream learning task in an end-to-end manner.

Computation graph

The computation of the embedding z_i of node i across multiple layers of a graph neural network can be conveniently represented using a so-called *computation graph* shown below.





Matrix formulation

Let $H \in \mathbb{R}^{n \times d}$ denote the matrix of node representations at a given layer of a graph neural network. Then, the update rule of *GCN* can be expressed compactly in a matrix form as

$$H' = \sigma \left(H \bar{W}^T + D^{-1} A H W^T \right),$$

where H' denotes the updated node representations.