Graph Neural Networks

From Transformers to Message Passing

Akash Malhotra

November 24, 2025

Abstract

We develop the theory of Graph Neural Networks, beginning with their connection to transformer architectures and proceeding to the general message-passing framework. Our treatment follows Bishop (2024), emphasizing permutation equivariance as a fundamental principle and exploring applications from molecular property prediction to social network analysis.

Contents

1 Introduction

Graph-structured data pervades modern machine learning. From the molecular structure of drugs to the topology of social networks, many problems involve relationships between discrete entities that cannot be naturally represented as sequences or grids. Graph Neural Networks provide a principled framework for learning on such data.

Figure 13.1 illustrates three canonical examples: a caffeine molecule where atoms are nodes connected by chemical bonds, a rail network with cities as nodes and railway lines as edges, and the worldwide web with pages as nodes and hyperlinks as edges. In each case, both the attributes of individual nodes and their patterns of connectivity contain essential information.

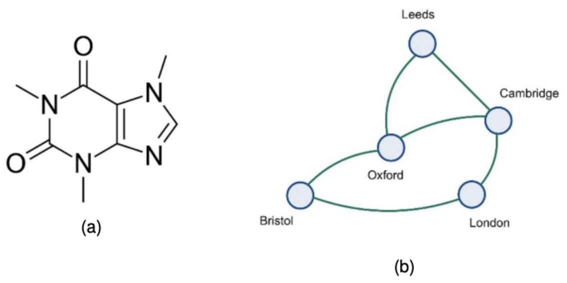

Figure 1: Three examples of graph-structured data: (a) caffeine molecule, (b) rail network, (c) worldwide web

2 From Transformers to Graphs

2.1 The Transformer as a Complete Graph

Consider the transformer architecture applied to a sequence $X = [x_1, \ldots, x_n] \in \mathbb{R}^{n\times d}$. The attention mechanism computes:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \tag{1}$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{2}$$

Every token attends to every other token. We may interpret this as defining a complete graph: each token is a node, and edges connect all pairs with weights determined by the attention scores. This fully connected structure incurs $O(n^2)$ computational cost.

2.2 The Graph Question

What if connectivity were sparse rather than complete? What if not all tokens needed to attend to all others, with the communication structure determined by domain knowledge? For sparse graphs where $|E| \ll n^2$, we might process information in $O(|E|)$ time rather than $O(n^2)$. This efficiency, combined with the ability to incorporate relational structure, motivates Graph Neural Networks.

3 Graphs and Their Representations

Definition 3.1 (Graph)**.** A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of vertices (nodes) $\mathcal{V} = \{v_1, \ldots, v_n\}$ and a set of edges (links) $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$.

We represent graphs numerically through node features $\mathbf{X} \in \mathbb{R}^{N \times D}$, where row n contains the feature vector $\mathbf{x}_n$ for node n, and an adjacency matrix $\mathbf{A} \in \{0,1\}^{N \times N}$ with

$$A_{nm} = \begin{cases} 1 & \text{if } (n,m) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

For undirected graphs, the adjacency matrix is symmetric: $A_{nm} = A_{mn}$.

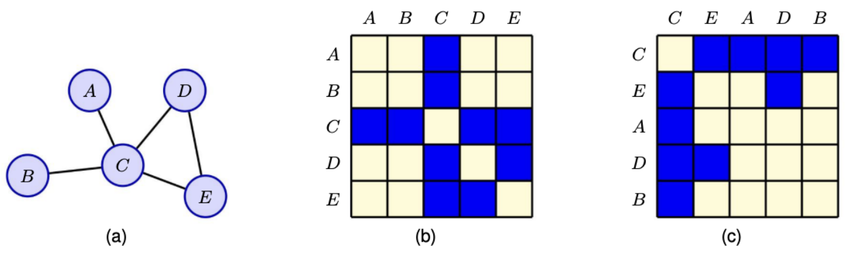

Figure 2: An example graph with five nodes and its adjacency matrix for two different node orderings, showing that the adjacency matrix depends on the arbitrary choice of node ordering.

The neighborhood of node n is defined as $\mathcal{N}(n) = \{m : A_{nm} = 1\}$, the set of nodes directly connected to n.

4 Permutation Symmetry

Graphs possess no canonical node ordering. Unlike sequences with their temporal structure or images with their spatial grid, the vertices in a graph may be labeled arbitrarily. This seemingly innocuous observation has profound consequences for learning on graphs.

4.1 Permutation Matrices

A permutation matrix $\mathbf{P}$ reorders nodes. Under permutation, the node features transform as $\tilde{\mathbf{X}} = \mathbf{P}\mathbf{X}$ and the adjacency matrix as $\tilde{\mathbf{A}} = \mathbf{P}\mathbf{A}\mathbf{P}^T$. For example, the permutation $(A, B, C, D, E) \to (C, E, A, D, B)$ corresponds to:

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

4.2 Invariance and Equivariance

Definition 4.1 (Permutation Invariance)**.** A function $y = f(\mathbf{X}, \mathbf{A})$ is permutation invariant if

$$y(\tilde{\mathbf{X}}, \tilde{\mathbf{A}}) = y(\mathbf{X}, \mathbf{A})$$

for all permutations. The output remains unchanged under node reordering.

Definition 4.2 (Permutation Equivariance)**.** A function $\mathbf{y} = f(\mathbf{X}, \mathbf{A})$ outputting one value per node is permutation equivariant if

$$\mathbf{y}(\tilde{\mathbf{X}}, \tilde{\mathbf{A}}) = \mathbf{P}\mathbf{y}(\mathbf{X}, \mathbf{A})$$

for all permutations. The output transforms in the same manner as the input.

Graph-level predictions require invariance, while node-level predictions require equivariance. These symmetries constitute essential inductive biases that we build into our architectures.

5 Neural Message Passing

5.1 Inspiration from Convolution

Convolutional neural networks provide a template. An image is a special case of graph-structured data: pixels are nodes, and edges connect spatially adjacent pixels. A 3×3 convolutional filter computing the value at position i in layer $l + 1$ from layer l takes the form

$$z_i^{(l+1)} = f\left(\sum_j w_j z_j^{(l)} + b\right) \tag{13.8}$$

where the sum runs over the 9-pixel patch, and weights w_j and bias b are shared across all patches. The same function is applied across multiple patches, ensuring parameter sharing (and thus not carrying an index i).

This operation is not equivariant under arbitrary node relabeling because the weight vector has positionally ordered elements. However, we can achieve equivariance by treating the node's own contribution separately:

$$z_i^{(l+1)} = f\left(w_{\text{neigh}} \sum_{j \in \mathcal{N}(i)} z_j^{(l)} + w_{\text{self}} z_i^{(l)} + b\right) \tag{3}$$

The aggregation $\sum_{j \in \mathcal{N}(i)} z_j^{(l)}$ is invariant to permutation of the neighbor labels, rendering the complete operation equivariant.

5.2 The Message-Passing Framework

We generalize this idea to arbitrary graphs. Each node n maintains an embedding vector $\mathbf{h}_n^{(l)} \in \mathbb{R}^D$ at layer l, initialized with $\mathbf{h}_n^{(0)} = \mathbf{x}_n$. A message-passing layer consists of two stages:

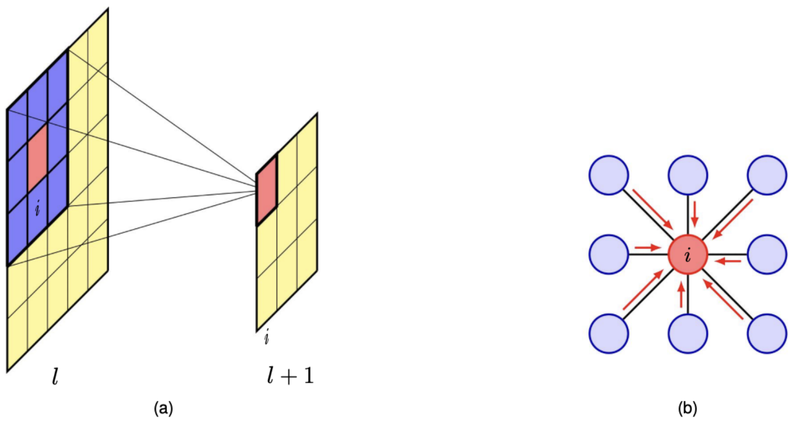

Figure 3: A convolutional filter for images can be represented as a graph-structured computation. (a) A filter computed by node i in layer $l+1$ is a function of activation values in layer l over a patch of pixels. (b) The same computation structure expressed as a graph showing messages flowing into node i from its neighbors.

Aggregation. Information from neighbors is gathered:

$$\mathbf{z}_n^{(l)} = \text{Aggregate}\left(\{\mathbf{h}_m^{(l)} : m \in \mathcal{N}(n)\}\right)$$

Update. The node's embedding is revised:

$$\mathbf{h}_n^{(l+1)} = \text{Update}\left(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l)}\right)$$

These operations are applied synchronously to all nodes, and the process iterates through L layers.

Algorithm 1 Simple Message-Passing Neural Network

```
1: **Input:** Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, initial embeddings $\{\mathbf{h}_n^{(0)} = \mathbf{x}_n\}$
2: **Output:** Final embeddings $\{\mathbf{h}_n^{(L)}\}$
3: for $l = 0, \ldots, L-1$ do
4:     $\mathbf{z}_n^{(l)} \leftarrow \text{Aggregate}\left(\{\mathbf{h}_m^{(l)} : m \in \mathcal{N}(n)\}\right)$ for all $n$
5:     $\mathbf{h}_n^{(l+1)} \leftarrow \text{Update}\left(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l)}\right)$ for all $n$
6: end for
7: return $\{\mathbf{h}_n^{(L)}\}$
```

5.3 Aggregation Functions

The aggregation function must be permutation invariant (neighbor order is arbitrary) and differentiable. Several choices arise:

Summation. The simplest aggregation:

$$\text{Aggregate}\left(\{\mathbf{h}_m^{(l)} : m \in \mathcal{N}(n)\}\right) = \sum_{m \in \mathcal{N}(n)} \mathbf{h}_m^{(l)}$$

This preserves multiset structure, distinguishes different neighborhood sizes, and requires no parameters.

Mean. Normalization by neighborhood size:

$$\text{Aggregate}\left(\{\mathbf{h}_m^{(l)} : m \in \mathcal{N}(n)\}\right) = \frac{1}{|\mathcal{N}(n)|} \sum_{m \in \mathcal{N}(n)} \mathbf{h}_m^{(l)}$$

This discards structural information but provides stability across varying node degrees.

Symmetric normalization. Following Kipf & Welling (2016):

$$\text{Aggregate}\left(\{\mathbf{h}_m^{(l)} : m \in \mathcal{N}(n)\}\right) = \sum_{m \in \mathcal{N}(n)} \frac{\mathbf{h}_m^{(l)}}{\sqrt{|\mathcal{N}(n)||\mathcal{N}(m)|}}$$

This accounts for degrees of both source and target nodes.

Maximum. Element-wise maximum:

$$\text{Aggregate}\left(\{\mathbf{h}_m^{(l)} : m \in \mathcal{N}(n)\}\right) = \max_{m \in \mathcal{N}(n)} \mathbf{h}_m^{(l)}$$

This focuses on salient features but loses count information.

MLP-based. A universal approximator (Zaheer et al., 2017):

$$\text{Aggregate}\left(\{\mathbf{h}_m^{(l)} : m \in \mathcal{N}(n)\}\right) = \text{MLP}_\theta\left(\sum_{m \in \mathcal{N}(n)} \text{MLP}_\phi(\mathbf{h}_m^{(l)})\right)$$

where both MLPs are shared across nodes. This can approximate any permutation-invariant function mapping sets of embeddings to single embeddings.

5.4 Update Functions

The update operator combines aggregated neighbor information with the node's own state. A simple form, analogous to the CNN case:

$$\text{Update}\left(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l)}\right) = f\left(\mathbf{W}_{\text{self}}\mathbf{h}_n^{(l)} + \mathbf{W}_{\text{neigh}}\mathbf{z}_n^{(l)} + \mathbf{b}\right) \tag{4}$$

where $f(\cdot)$ is a nonlinear activation and $\mathbf{W}_{\text{self}}$, $\mathbf{W}_{\text{neigh}}$, $\mathbf{b}$ are learnable parameters.

If we use summation for aggregation and share weights $\mathbf{W}_{\text{self}} = \mathbf{W}_{\text{neigh}}$:

$$\mathbf{h}_n^{(l+1)} = f\left(\mathbf{W}_{\text{neigh}} \sum_{m \in \mathcal{N}(n) \cup \{n\}} \mathbf{h}_m^{(l)} + \mathbf{b}\right) \tag{5}$$

5.5 Matrix Formulation

We may express the network as a sequence of transformations:

$$\mathbf{H}^{(1)} = \mathbf{F}\left(\mathbf{X}, \mathbf{A}, \mathbf{W}^{(1)}\right) \tag{6}$$

$$\mathbf{H}^{(2)} = \mathbf{F}\left(\mathbf{H}^{(1)}, \mathbf{A}, \mathbf{W}^{(2)}\right) \tag{7}$$

$$\vdots \tag{8}$$

$$\mathbf{H}^{(L)} = \mathbf{F}\left(\mathbf{H}^{(L-1)}, \mathbf{A}, \mathbf{W}^{(L)}\right) \tag{9}$$

where $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times D}$ contains all node embeddings at layer l, and $\mathbf{W}^{(l)}$ represents the learnable parameters.

Under permutation $\mathbf{P}$, the transformation satisfies:

$$\mathbf{PH}^{(l)} = \mathbf{F}\left(\mathbf{PH}^{(l-1)}, \mathbf{PAP}^{T}, \mathbf{W}^{(l)}\right)$$

ensuring the complete network computes an equivariant transformation.

5.6 Receptive Fields

Each layer expands the receptive field by one hop. After L layers, node n has effectively aggregated information from all nodes within distance L in the graph.

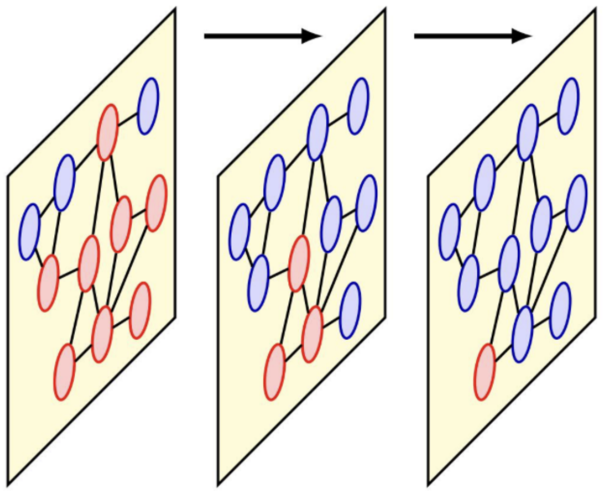

Figure 4: Schematic illustration of information flow through successive layers of a graph neural network. In the third layer a single node is highlighted in red. It receives information from its two neighbors in the previous layer and those in turn receive information from their neighbors in the first layer. As with convolutional neural networks, the effective receptive field grows with the number of processing layers.

For large, sparse graphs, many layers may be required before each output depends on all inputs. Some architectures introduce auxiliary "super-nodes" connected to all vertices to accelerate information propagation.

6 Learning Tasks

6.1 Node Classification

After L layers, we obtain node embeddings $\{\mathbf{h}_n^{(L)}\}$. For classification into C classes, we apply a softmax readout layer:

$$y_{ni} = \frac{\exp(\mathbf{w}_i^T \mathbf{h}_n^{(L)})}{\sum_j \exp(\mathbf{w}_j^T \mathbf{h}_n^{(L)})}$$

where $\{\mathbf{w}_i\}$ are learnable weight vectors. Training minimizes cross-entropy:

$$\mathcal{L} = - \sum_{n \in \mathcal{V}_{\text{train}}} \sum_{i=1}^{C} t_{ni} \log y_{ni}$$

where $\{t_{ni}\}$ are one-hot encoded targets.

We distinguish three types of nodes:

1. $\mathcal{V}_{\text{train}}$: labeled nodes included in the training loss
2. $\mathcal{V}_{\text{trans}}$: unlabeled transductive nodes participating in message-passing during training, with labels predicted at inference
3. $\mathcal{V}_{\text{induct}}$: inductive nodes unavailable during training, with labels predicted at inference

Transductive learning uses a fixed graph with some unlabeled nodes, a form of semi-supervised learning common in citation networks and social networks. *Inductive learning* trains on one set of graphs and tests on entirely different graphs, as in molecular property prediction.

6.2 Edge Classification

For edge-level predictions, we combine node embeddings. Link prediction uses:

$$p(n, m) = \sigma\left(\mathbf{h}_n^T \mathbf{h}_m\right)$$

to define the probability of an edge between nodes n and m.

6.3 Graph Classification

Graph-level predictions require permutation-invariant pooling. The simplest approach sums all node embeddings:

$$\mathbf{y} = f\left(\sum_{n \in \mathcal{V}} \mathbf{h}_n^{(L)}\right)$$

where f may be a linear transformation or neural network. Mean, maximum, or minimum aggregations are also valid. For classification, we apply cross-entropy loss; for regression (e.g., molecular solubility), sum-of-squares loss.

7 Extensions

7.1 Graph Attention

Attention mechanisms enable adaptive weighting of neighbors. The aggregation becomes:

$$\mathbf{z}_n^{(l)} = \sum_{m \in \mathcal{N}(n)} A_{nm} \mathbf{h}_m^{(l)}$$

where attention coefficients satisfy $A_{nm} \geq 0$ and $\sum_{m \in \mathcal{N}(n)} A_{nm} = 1$.

A bilinear form computes attention (Veličković et al., 2017):

$$A_{nm} = \frac{\exp(\mathbf{h}_n^T \mathbf{W} \mathbf{h}_m)}{\sum_{m' \in \mathcal{N}(n)} \exp(\mathbf{h}_n^T \mathbf{W} \mathbf{h}_{m'})}$$

More generally, an MLP combines embeddings:

$$A_{nm} = \frac{\exp\{\mathrm{MLP}(\mathbf{h}_n, \mathbf{h}_m)\}}{\sum_{m' \in \mathcal{N}(n)} \exp\{\mathrm{MLP}(\mathbf{h}_n, \mathbf{h}_{m'})\}}$$

The MLP is shared across all nodes, preserving equivariance. Multi-head attention employs H independent attention mechanisms combined via concatenation.

7.2 Edge and Graph Embeddings

Some applications require embeddings for edges and the entire graph. We maintain edge embeddings $\mathbf{e}_{nm}^{(l)}$ and graph embedding $\mathbf{g}^{(l)}$, with update equations (Battaglia et al., 2018):

$$\mathbf{e}_{nm}^{(l+1)} = \mathrm{Update}_{\mathrm{edge}}\left(\mathbf{e}_{nm}^{(l)}, \mathbf{h}_n^{(l)}, \mathbf{h}_m^{(l)}, \mathbf{g}^{(l)}\right) \tag{10}$$

$$\mathbf{z}_n^{(l+1)} = \mathrm{Aggregate}_{\mathrm{node}}\left(\{\mathbf{e}_{nm}^{(l+1)} : m \in \mathcal{N}(n)\}\right) \tag{11}$$

$$\mathbf{h}_n^{(l+1)} = \mathrm{Update}_{\mathrm{node}}\left(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l+1)}, \mathbf{g}^{(l)}\right) \tag{12}$$

$$\mathbf{g}^{(l+1)} = \mathrm{Update}_{\mathrm{graph}}\left(\mathbf{g}^{(l)}, \{\mathbf{h}_n^{(l+1)}\}, \{\mathbf{e}_{nm}^{(l+1)}\}\right) \tag{13}$$

Algorithm 2 GNN with Node, Edge, and Graph Embeddings

```
1: Input: Graph $\mathcal{G}$, initial embeddings $\{\mathbf{h}_n^{(0)}\}$, $\{\mathbf{e}_{nm}^{(0)}\}$, $\mathbf{g}^{(0)}$
2: Output: Final embeddings $\{\mathbf{h}_n^{(L)}\}$, $\{\mathbf{e}_{nm}^{(L)}\}$, $\mathbf{g}^{(L)}$
3: for $l = 0, \ldots, L-1$ do
4:     Update all edges: $\mathbf{e}_{nm}^{(l+1)} \leftarrow \mathrm{Update}_{\mathrm{edge}}(\mathbf{e}_{nm}^{(l)}, \mathbf{h}_n^{(l)}, \mathbf{h}_m^{(l)}, \mathbf{g}^{(l)})$
5:     Aggregate for nodes: $\mathbf{z}_n^{(l+1)} \leftarrow \mathrm{Aggregate}_{\mathrm{node}}(\{\mathbf{e}_{nm}^{(l+1)} : m \in \mathcal{N}(n)\})$
6:     Update all nodes: $\mathbf{h}_n^{(l+1)} \leftarrow \mathrm{Update}_{\mathrm{node}}(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l+1)}, \mathbf{g}^{(l)})$
7:     Update graph: $\mathbf{g}^{(l+1)} \leftarrow \mathrm{Update}_{\mathrm{graph}}(\mathbf{g}^{(l)}, \{\mathbf{h}_n^{(l+1)}\}, \{\mathbf{e}_{nm}^{(l+1)}\})$
8: end for
9: return $\{\mathbf{h}_n^{(L)}\}$, $\{\mathbf{e}_{nm}^{(L)}\}$, $\mathbf{g}^{(L)}$
```

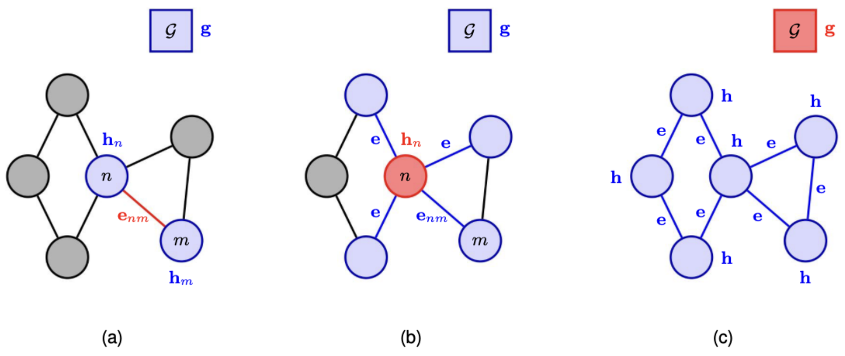

Figure 5: Illustration of the general graph message-passing updates showing (a) edge updates, (b) node updates, and (c) global graph updates. In each case the variable being updated is shown in red and the variables that contribute to that update are shown in red and blue.

7.3 Over-Smoothing

A significant challenge arises in deep networks: after many message-passing iterations, node embeddings tend to converge, effectively limiting network depth. This *over-smoothing* phenomenon occurs because repeated aggregation causes all nodes to incorporate information from the entire graph, particularly in small-world networks.

Residual connections mitigate this issue:

$$\mathbf{h}_n^{(l+1)} = \text{Update}_{\text{node}}\left(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l+1)}, \mathbf{g}^{(l)}\right) + \mathbf{h}_n^{(l)}$$

Alternatively, *Jumping Knowledge Networks* aggregate representations from all layers:

$$\mathbf{y}_n = f\left(\mathbf{h}_n^{(1)} \oplus \mathbf{h}_n^{(2)} \oplus \cdots \oplus \mathbf{h}_n^{(L)}\right)$$

where $\oplus$ denotes concatenation, or max-pooling may be used instead.

7.4 Deep Sets

An instructive special case occurs when the graph has no edges ($\mathbf{A} = 0$). Using MLP-based aggregation:

$$\text{Aggregate}\left(\{\mathbf{h}_m^{(l)}\}\right) = \text{MLP}_\theta\left(\sum_{m\in\mathcal{V}} \text{MLP}_\phi(\mathbf{h}_m^{(l)})\right)$$

(where the sum excludes $\mathbf{h}_n^{(l)}$ itself), we obtain a framework for learning functions over unstructured sets, known as *deep sets*. Graph neural networks thus generalize set-based learning to incorporate relational structure.

8 Geometric Equivariance

When predicting molecular properties, physical symmetries constrain the answer. A molecule's solubility, for instance, is invariant under translations, rotations, and reflections in three-dimensional space.

We can incorporate coordinate embeddings $\mathbf{r}_n^{(l)} \in \mathbb{R}^3$ for each atom. The update equations (Satorras, Hoogeboom, and Welling, 2021):

$$\mathbf{e}_{nm}^{(l+1)} = \text{Update}_{\text{edge}}\left(\mathbf{e}_{nm}^{(l)}, \mathbf{h}_n^{(l)}, \mathbf{h}_m^{(l)}, \|\mathbf{r}_n^{(l)} - \mathbf{r}_m^{(l)}\|^2\right) \tag{14}$$

$$\mathbf{r}_n^{(l+1)} = \mathbf{r}_n^{(l)} + C \sum_{(n,m)\in\mathcal{E}} \left(\mathbf{r}_n^{(l)} - \mathbf{r}_m^{(l)}\right) \phi\left(\mathbf{e}_{nm}^{(l+1)}\right) \tag{15}$$

$$\mathbf{z}_n^{(l+1)} = \text{Aggregate}_{\text{node}}\left(\{\mathbf{e}_{nm}^{(l+1)} : m \in \mathcal{N}(n)\}\right) \tag{16}$$

$$\mathbf{h}_n^{(l+1)} = \text{Update}_{\text{node}}\left(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l+1)}\right) \tag{17}$$

The squared distance $\|\mathbf{r}_n - \mathbf{r}_m\|^2$ is invariant to rigid transformations, while the coordinate update using relative differences $(\mathbf{r}_n - \mathbf{r}_m)$ is equivariant. This encodes geometric symmetry into the architecture, forming the basis of *geometric deep learning.*

9 Applications

Graph neural networks find application across diverse domains.

Molecular property prediction. Atoms are nodes, chemical bonds are edges. Tasks include predicting toxicity, binding affinity, and solubility. The graph structure encodes chemical connectivity, while node features represent atom types. Molecular properties are invariant to spatial transformations, motivating equivariant architectures.

Social networks. Users are nodes, relationships are edges. Node classification predicts user interests or identifies fake accounts. Link prediction recommends friendships. Community detection discovers clusters. These are typically transductive problems on fixed, large graphs.

Citation networks. Papers are nodes, citations are edges. Semi-supervised node classification assigns topics to papers. This canonical transductive setting has few labeled examples and many unlabeled ones.

Transportation networks. Road segments or stations are nodes, connections are edges. Spatial-temporal GNNs combine graph structure (space) with recurrent or temporal convolutional networks (time) to predict traffic flow, taxi demand, or congestion.

Protein structure. Amino acids or atoms are nodes, edges based on spatial proximity or chemical bonds. Applications include structure prediction (AlphaFold uses equivariant GNNs), function prediction, and drug-target interaction.

Recommendation systems. Users and items form a bipartite graph. GNNs learn embeddings capturing collaborative filtering signals, incorporating side information and graph structure for improved recommendations.

10 Conclusion

Graph neural networks provide a principled framework for learning on relational data. The key principles are:

Permutation symmetry. Graphs have no canonical node ordering. Invariance and equivariance are fundamental inductive biases.

Message passing. The unifying computational pattern: aggregate information from neighbors, update node states, iterate. Different GNN architectures correspond to different choices of aggregation and update functions.

Locality and receptive fields. Information propagates one hop per layer, with the effective receptive field growing with depth. Over-smoothing limits depth; residual connections and layer aggregation provide mitigation.

Hierarchy of predictions. Node-level, edge-level, and graph-level tasks require equivariant or invariant operations as appropriate.

Connections. Transformers are complete graphs. Deep sets are graphs without edges. Geometric deep learning extends to continuous symmetries.

The field continues to evolve, with active research on expressiveness, scalability, interpretability, and applications across science and engineering.

Further Reading

Foundational papers:

- Gilmer et al. (2017). Neural Message Passing for Quantum Chemistry
- Kipf & Welling (2017). Semi-Supervised Classification with Graph Convolutional Networks
- Hamilton et al. (2017). Inductive Representation Learning on Large Graphs
- Veličković et al. (2017). Graph Attention Networks
- Xu et al. (2019). How Powerful are Graph Neural Networks?
- Battaglia et al. (2018). Relational Inductive Biases, Deep Learning, and Graph Networks

Textbook:

- Bishop, C. M. (2024). *Deep Learning: Foundations and Concepts*, Chapter 13

Software:

- PyTorch Geometric (PyG): `https://pytorch-geometric.readthedocs.io/`
- Deep Graph Library (DGL): `https://www.dgl.ai/`