Graph Mining and Graph Neural Networks

Lecture Notes

Clement Wang

2025–2026

These notes cover the foundations of Graph Neural Networks, from the message-passing framework to the GCN and GraphSAGE architectures, with a spectral analysis and a discussion of practical limitations. Applications to recommender systems, routing, and board games are also presented.

Contents

1 Introduction: Motivation and Setup **3**
- 1.1 Why Graphs? . . . 3
- 1.2 Notation . . . 3
- 1.3 The Supervised Learning Framework . . . 4
- 1.4 Prediction Tasks on Graphs . . . 5
  - 1.4.1 Node-Level Prediction . . . 5
  - 1.4.2 Graph-Level Prediction . . . 5
  - 1.4.3 Edge-Level Prediction (Link Prediction) . . . 5

2 From MLPs to Message Passing **5**
- 2.1 Multi-Layer Perceptrons and Their Limits on Graphs . . . 5
- 2.2 Local Aggregation: The Key Idea . . . 5
- 2.3 The Message Passing Framework . . . 6
- 2.4 Node Representations and Layer Stacking . . . 6
- 2.5 Computation Tree and Receptive Field . . . 6

3 Specific Architectures **7**
- 3.1 Graph Convolutional Network (GCN) . . . 7
- 3.2 GraphSAGE . . . 8
- 3.3 Comparison of GCN and GraphSAGE . . . 8

4 Training GNNs **8**
- 4.1 Node Classification . . . 8
- 4.2 Transductive vs. Inductive Learning . . . 8
- 4.3 Graph-Level Prediction . . . 9
- 4.4 Link Prediction . . . 9

5 Spectral View of Graph Neural Networks **9**
- 5.1 Graph Signals . . . 9
- 5.2 The Graph Laplacian . . . 10
- 5.3 Graph Fourier Transform . . . 10
- 5.4 Spectral Graph Convolution . . . 11

5.5 Polynomial Filters and Locality . . . 11
5.6 GCN as a First-Order Polynomial Filter . . . 11

6 Limitations and Extensions **12**
6.1 Oversmoothing . . . 12
6.2 Mini-Batch GNNs for Scalability . . . 12
6.2.1 Neighborhood Sampling . . . 13
6.2.2 Subgraph-Based Methods . . . 13
6.2.3 Layer-Wise Sampling . . . 14
6.3 Heterophily . . . 14
6.4 Long-Range Dependencies . . . 15
6.5 Attention-Based GNNs . . . 15
6.5.1 Graph Attention Networks (GAT) . . . 15
6.5.2 Graph Transformers . . . 16

7 Applications **16**
7.1 Board Games: AlphaZero with GNNs . . . 16
7.2 Routing and ETA Prediction: Google Maps . . . 16
7.3 Recommender Systems: PinSage . . . 17

1 Introduction: Motivation and Setup

1.1 Why Graphs?

Many real-world datasets have an inherent relational structure: entities are linked by pairwise relationships that carry predictive signal. Representative examples include:

- **Citation networks.** Scientific papers (nodes) connected by citations (edges); predict the topic label of each paper.
- **Molecular graphs.** Atoms (nodes) connected by chemical bonds (edges); predict molecular properties such as toxicity or solubility.
- **Social networks.** Users (nodes) connected by friendship links (edges); detect communities or recommend content.
- **Road networks.** Road segments (nodes) connected at intersections (edges); predict travel time or congestion.
- **Knowledge graphs.** Entities (nodes) connected by typed relations (edges); predict missing relations (knowledge-base completion).

In all these settings, the edge structure encodes information that cannot be recovered from node features alone. The central observation motivating GNNs is that the most effective deep learning architectures match the inductive structure of the data:

Data type	**Structure**	**Architecture**
Images	Regular spatial grid	CNN
Text	Ordered token sequence	Transformer, RNN
Graphs	Irregular node-edge set	GNN

CNNs exploit translation symmetry by reusing the same convolutional kernel at every spatial position. GNNs exploit the edge set by aggregating information from graph neighbors—the graph-structured analogue of a local convolutional filter.

1.2 Notation

We fix notation used throughout. We write $G = (V, E)$ for an undirected simple graph with $n = |V|$ nodes and $m = |E|$ edges, $A \in \{0,1\}^{n\times n}$ for its adjacency matrix, and $D = \mathrm{diag}(d_1, \ldots, d_n)$ for the degree matrix.

Definition 1.1 (Node Feature Matrix)**.** Each node $v \in V$ is associated with a *feature vector* $x_v \in \mathbb{R}^{d_0}$. The *feature matrix* is

$$X = \begin{bmatrix} - & x_1^\top & - \\ & \vdots & \\ - & x_n^\top & - \end{bmatrix} \in \mathbb{R}^{n\times d_0}.$$

We adopt the convention $H^{(0)} = X$ and $h_v^{(0)} = x_v$.

Definition 1.2 (Neighborhood)**.** The *neighborhood* of node v is $\mathcal{N}(v) = \{u \in V : (u, v) \in E\}$ with $d_v = |\mathcal{N}(v)|$. The *augmented neighborhood* (including the node itself) is $\widetilde{\mathcal{N}}(v) = \mathcal{N}(v) \cup \{v\}$ with $\tilde{d}_v = d_v + 1$.

Definition 1.3 (Graph Matrices)**.** We define the following matrices, all in $\mathbb{R}^{n\times n}$:

$\widetilde{A} = A + I_n$	(augmented adjacency: adds self-loops)
$\widetilde{D} = \mathrm{diag}(\tilde{d}_1, \ldots, \tilde{d}_n)$	(augmented degree matrix)
$\widehat{A} = \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}$	(symmetrically normalized augmented adjacency)
$L = D - A$	(unnormalized Laplacian)
$L_{\mathrm{sym}} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}AD^{-1/2}$	(symmetrically normalized Laplacian)

Notation summary

Symbol	Type	Meaning
$G = (V, E)$	graph	undirected simple graph
$n = \lvert V\rvert$, $m = \lvert E\rvert$	integers	order and size
A	$\{0,1\}^{n\times n}$	adjacency matrix
D	$\mathbb{R}^{n\times n}$	degree matrix $D_{vv} = d_v$
$\widetilde{A} = A + I$	$\{0,1\}^{n\times n}$	augmented adjacency
$\widetilde{D}$	$\mathbb{R}^{n\times n}$	augmented degree matrix
$\widehat{A}$	$\mathbb{R}^{n\times n}$	normalized augmented adjacency
$L = D - A$	$\mathbb{R}^{n\times n}$	unnormalized Laplacian
x_v	$\mathbb{R}^{d_0}$	feature vector of node v
X	$\mathbb{R}^{n\times d_0}$	feature matrix (rows $= x_v^\top$)
$h_v^{(l)}$	$\mathbb{R}^{d_l}$	embedding of v at layer l
$H^{(l)}$	$\mathbb{R}^{n\times d_l}$	embedding matrix at layer l
$W^{(l)}$	$\mathbb{R}^{d_l\times d_{l+1}}$	weight matrix at layer l
K	integer	total number of GNN layers
f_θ	$(G, X) \to \hat{Y}$	learnable GNN (parameters θ)
θ	—	all learnable parameters

1.3 The Supervised Learning Framework

A *dataset* consists of one or more graphs, each with node features and target labels. The learner observes a labeled training set and minimizes a loss over it by gradient descent.

Definition 1.4 (Empirical Risk Minimization on Graphs)**.** Let $f_\theta : (G, X) \mapsto \hat{Y}$ be a parametric model with parameters $\theta \in \Theta$. Given labeled nodes $V_L \subset V$ with targets $y_v \in \mathcal{Y}$, the *empirical risk* is

$$\widehat{\mathcal{L}}(\theta) = \frac{1}{|V_L|} \sum_{v\in V_L} \ell(f_\theta(G, X)_v,\, y_v)\,,$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ is a per-sample loss.

Common choices of loss:

- **Multi-class classification** ($y_v \in \{1, \ldots, C\}$): cross-entropy $\ell(\hat{y}_v, y_v) = -\log \hat{y}_{v,y_v}$, where $\hat{y}_v = \mathrm{softmax}(z_v) \in \Delta^{C-1}$ and $z_v \in \mathbb{R}^C$ are the output logits.
- **Binary classification**: binary cross-entropy.
- **Regression** ($y_v \in \mathbb{R}$): squared loss $\ell(\hat{y}_v, y_v) = (\hat{y}_v - y_v)^2$.

Optimization. On graphs, the full loss $\widehat{\mathcal{L}}(\theta)$ is typically computed over the entire graph at once, so plain *gradient descent* is used:

$$\theta \leftarrow \theta - \eta\, \nabla_\theta \widehat{\mathcal{L}}(\theta),$$

where $\eta > 0$ is the learning rate. In practice, any first-order optimizer can replace the vanilla update; Adam [1] is the most common choice, as it adapts per-parameter learning rates using estimates of the first and second moments of the gradient. Gradients are computed by backpropagation through the GNN computation graph.

1.4 Prediction Tasks on Graphs

There are three standard prediction settings, corresponding to different output granularities.

1.4.1 Node-Level Prediction

Definition 1.5 (Node Classification)**.** Given $G = (V, E)$, features $X \in \mathbb{R}^{n \times d_0}$, and labels $\{y_v\}_{v \in V_L}$ with $y_v \in \{1, \dots, C\}$, learn $f_\theta : (G, X) \to \mathbb{R}^{n \times C}$ minimizing $\frac{1}{|V_L|} \sum_{v \in V_L} \ell(f_\theta(G, X)_v, y_v)$.

Example 1.6. Cora [2] is a citation network of 2708 papers with 5429 citation edges. Each paper has a bag-of-words feature vector ($d_0 = 1433$) and a topic label ($C = 7$). Training labels are provided for 140 nodes; the task is to predict labels for the remaining nodes using the full graph structure.

1.4.2 Graph-Level Prediction

Definition 1.7 (Graph Classification)**.** Given a dataset $\{(G_i, X_i, y_i)\}_{i=1}^N$ of labeled graphs with $y_i \in \{1, \dots, C\}$, learn $f_\theta : (G, X) \to \mathbb{R}^C$ that predicts a class distribution for an entire graph.

Example 1.8. In molecular property prediction, each molecule is a graph (atoms as nodes, bonds as edges). The Tox21 and ZINC benchmarks [3] provide binary toxicity labels and continuous property values respectively.

1.4.3 Edge-Level Prediction (Link Prediction)

Definition 1.9 (Link Prediction)**.** Given $G = (V, E)$ and features X, learn a scoring function $f_\theta : (G, X, u, v) \to [0, 1]$ estimating the probability that edge (u, v) should exist (or will exist in the future).

Example 1.10. In a knowledge graph, link prediction asks whether relation r holds between entities u and v. In a recommender system, it asks whether user u will interact with item v.

2 From MLPs to Message Passing

2.1 Multi-Layer Perceptrons and Their Limits on Graphs

A standard MLP processes each node feature independently:

$$h_v^{(l+1)} = \sigma\Big(W^{(l)} h_v^{(l)} + b^{(l)}\Big), \tag{1}$$

where σ is a nonlinearity (e.g., ReLU), $W^{(l)} \in \mathbb{R}^{d_{l+1} \times d_l}$, and $b^{(l)} \in \mathbb{R}^{d_{l+1}}$. Applied row-wise to $H^{(l)}$, this gives $H^{(l+1)} = \sigma(H^{(l)}(W^{(l)})^\top + \mathbf{1}(b^{(l)})^\top)$.

The MLP is *permutation equivariant* in the sense that permuting the rows of X permutes the rows of the output in the same way. However, it entirely ignores the edge set E: the same embedding $h_v^{(K)}$ is produced regardless of who v's neighbors are and what features they carry.

2.2 Local Aggregation: The Key Idea

The CNN's success on images comes from two principles:

1. **Locality**: each output pixel depends only on a small spatial neighborhood (the receptive field).
2. **Weight sharing**: the same filter is applied at every position.

On a regular grid, the neighborhood is always a $k \times k$ patch, so defining a local, weight-shared filter is straightforward. On a graph, the neighborhood $\mathcal{N}(v)$ is *irregular*: different nodes have different numbers of neighbors, and neighbors have no canonical order. This calls for a *permutation-invariant* aggregation over a variable-size set.

2.3 The Message Passing Framework

Definition 2.1 (Message Passing Layer)**.** A *message passing layer* computes new node embeddings in two steps. Given embeddings $\{h_v^{(l)}\}_{v \in V}$ at layer l:

$$m_v^{(l)} = \mathrm{AGG}^{(l)}\Big(\{h_u^{(l)} : u \in \mathcal{N}(v)\}\Big) \tag{2}$$

$$h_v^{(l+1)} = \mathrm{UPDATE}^{(l)}\Big(h_v^{(l)},\, m_v^{(l)}\Big). \tag{3}$$

The functions $\mathrm{AGG}^{(l)}$ and $\mathrm{UPDATE}^{(l)}$ are learnable; the same functions are shared across all nodes (weight sharing).

The aggregation must be order-independent, since $\mathcal{N}(v)$ has no canonical ordering. This requires *permutation invariance*.

Definition 2.2 (Permutation Invariance)**.** A function f on multisets of vectors in $\mathbb{R}^d$ is *permutation invariant* if for every finite multiset $S = \{z_1, \ldots, z_k\} \subset \mathbb{R}^d$ and every permutation π of $\{1, \ldots, k\}$,

$$f(\{z_1, \ldots, z_k\}) = f(\{z_{\pi(1)}, \ldots, z_{\pi(k)}\}).$$

Proposition 2.3 (Standard Aggregators)**.** *The following functions on multisets of vectors in $\mathbb{R}^d$ are permutation invariant:*

$$\sum_{u \in \mathcal{N}(v)} h_u^{(l)}, \qquad \frac{1}{d_v} \sum_{u \in \mathcal{N}(v)} h_u^{(l)}, \qquad \max_{u \in \mathcal{N}(v)} h_u^{(l)} \text{ (coordinate-wise).}$$

2.4 Node Representations and Layer Stacking

Starting from $H^{(0)} = X$, a K-layer GNN produces a sequence of representation matrices:

$$H^{(0)} \xrightarrow{\text{Layer 1}} H^{(1)} \xrightarrow{\text{Layer 2}} \cdots \xrightarrow{\text{Layer } K} H^{(K)}.$$

At layer l, the *embedding* of node v is the row vector $h_v^{(l)} \in \mathbb{R}^{d_l}$. Each layer refines the embeddings by incorporating information from the graph neighborhood.

Remark 2.4 (Depth vs. width)**.** Increasing K expands the receptive field (see below) but can cause over-smoothing (Section 6.1). Increasing d_l increases the representational capacity within each layer.

2.5 Computation Tree and Receptive Field

Definition 2.5 (Computation Neighborhood)**.** For a K-layer GNN and a node $v \in V$, the *depth-k computation neighborhood* is

$$\mathcal{N}^{(0)}(v) = \{v\}, \qquad \mathcal{N}^{(k)}(v) = \mathcal{N}^{(k-1)}(v) \cup \bigcup_{u \in \mathcal{N}^{(k-1)}(v)} \mathcal{N}(u), \quad k \geq 1.$$

Proposition 2.6 (Receptive Field)**.** *After K message-passing layers, the embedding $h_v^{(K)}$ depends only on the node features $\{x_u : u \in \mathcal{N}^{(K)}(v)\}$, i.e., features within the K-hop neighborhood of v.*

Proof. By induction on K. Base case: $h_v^{(0)} = x_v$ depends only on $\{x_u : u \in \mathcal{N}^{(0)}(v)\} = \{x_v\}$. Inductive step: Equation (3) shows that $h_v^{(l+1)}$ depends on $h_v^{(l)}$ and $m_v^{(l)}$; by (2), $m_v^{(l)}$ depends on $\{h_u^{(l)} : u \in \mathcal{N}(v)\}$. By the inductive hypothesis, $h_u^{(l)}$ depends on $\{x_w : w \in \mathcal{N}^{(l)}(u)\}$. Unioning over $u \in \mathcal{N}(v)$ gives $\mathcal{N}^{(l+1)}(v)$. □

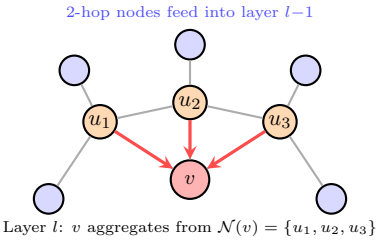


Figure 1: Message passing at node v. Red arrows show messages from 1-hop neighbors $\{u_1, u_2, u_3\}$ flowing into v at layer l. The blue (2-hop) nodes feed the same layer applied to u_1, u_2, u_3.

3 Specific Architectures

3.1 Graph Convolutional Network (GCN)

The Graph Convolutional Network of Kipf and Welling [4] is the most widely used message-passing architecture. Its derivation from spectral graph convolution is presented in Section 5.6; here we give the spatial formulation directly.

Definition 3.1 (GCN Layer – Matrix Form)**.**

$$H^{(l+1)} = \sigma\Big(\widehat{A}\, H^{(l)}\, W^{(l)}\Big),$$

where $\widehat{A} = \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}$ (Definition 1.3), $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ is a learnable weight matrix, and σ is a nonlinearity (e.g. ReLU) applied element-wise.

Proposition 3.2 (GCN Layer – Node Form)**.** *Definition 3.1 is equivalent to the following node-level update:*

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \widetilde{\mathcal{N}}(v)} \frac{1}{\sqrt{\tilde{d}_v\, \tilde{d}_u}}\, h_u^{(l)}\, W^{(l)}\right).$$

Proof. The $(v, \cdot)$ row of $\widehat{A}H^{(l)}W^{(l)}$ is $\sum_u \widehat{A}_{vu} h_u^{(l)} W^{(l)}$. By definition, $\widehat{A}_{vu} = (\widetilde{D}^{-1/2})_{vv}\widetilde{A}_{vu}(\widetilde{D}^{-1/2})_{uu} = \widetilde{A}_{vu}/\sqrt{\tilde{d}_v \tilde{d}_u}$. Since $\widetilde{A}_{vu} = 1$ iff $u \in \widetilde{\mathcal{N}}(v)$ (and zero otherwise), the sum reduces to the stated formula. □

Remark 3.3 (Bias term)**.** Like in an MLP, a bias $b^{(l)}$ can be added inside the nonlinearity. We omit it to keep formulas simpler, but it is present in practice.

Remark 3.4 (Role of the normalization)**.** Without normalization (i.e., using A directly), high-degree nodes dominate the aggregation and the norm of $h_v^{(l)}$ scales with d_v, destabilizing training. The symmetric normalization $\widehat{A}$ ensures spectral radius at most 1 (Proposition 5.13), giving numerically stable forward passes.

Two-Layer GCN for Node Classification The standard pipeline for node classification combines two GCN layers with a softmax output:

$$H^{(1)} = \sigma\Big(\widehat{A}\, X\, W^{(0)}\Big),$$
$$Z = \mathrm{softmax}\Big(\widehat{A}\, H^{(1)}\, W^{(1)}\Big).$$

Here $W^{(0)} \in \mathbb{R}^{d_0 \times d_1}$ and $W^{(1)} \in \mathbb{R}^{d_1 \times C}$ are the two weight matrices; $Z \in \mathbb{R}^{n \times C}$ contains the predicted class probabilities for all nodes. The loss is the cross-entropy on labeled nodes: $\mathcal{L}(\theta) = -\sum_{v \in V_L} \log Z_{v, y_v}$.

3.2 GraphSAGE

GraphSAGE [5] was designed for *inductive* learning on large graphs (Section 4.2). Unlike GCN, it maintains separate weight matrices for the node's own embedding and for the aggregated neighborhood message.

Definition 3.5 (GraphSAGE Layer)**.**

$$m_v^{(l)} = \mathrm{AGG}\Big(\{h_u^{(l)} : u \in \mathcal{N}(v)\}\Big), \quad \mathrm{AGG} \in \{\text{mean, max, LSTM}\},$$
$$h_v^{(l+1)} = \sigma\Big(\mathrm{CONCAT}\Big(h_v^{(l)}, m_v^{(l)}\Big)\, W^{(l)}\Big).$$

Remark 3.6 (Bias term)**.** Usually, a bias term $b^{(l)}$ is added inside the nonlinearity, but as for GCN, it is omitted in the formula for simplicity.

Remark 3.7 (Weight matrix split)**.** Partitioning $W^{(l)} \in \mathbb{R}^{2d_l \times d_{l+1}}$ into two blocks $W_{\text{self}}^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ and $W_{\text{neigh}}^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ according to the concatenation, and using the mean aggregator, gives:

$$h_v^{(l+1)} = \sigma\left(h_v^{(l)}\, W_{\text{self}}^{(l)} + \frac{1}{d_v} \sum_{u \in \mathcal{N}(v)} h_u^{(l)}\, W_{\text{neigh}}^{(l)}\right).$$

Compared with GCN, the key differences are: (i) separate weights for self and neighbors; (ii) no symmetric degree normalization; (iii) a learned aggregator rather than a fixed weighted sum.

3.3 Comparison of GCN and GraphSAGE

Property	**GCN**	**GraphSAGE**
Aggregation	Sym. normalized sum	Mean / max / LSTM
Self-connection	Via $\widetilde{A} = A + I$	Via explicit CONCAT
Weight matrix	Shared: $W^{(l)}$	Separate: $W_{\text{self}}^{(l)}, W_{\text{neigh}}^{(l)}$
Normalization	Sym. $\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}$	$1/d_v$

4 Training GNNs

4.1 Node Classification

For node classification, labels are observed for a subset $V_L \subset V$ of training nodes. Crucially, *all* nodes (labeled or not) participate in message passing; only the labeled nodes contribute to the loss:

$$\mathcal{L}(\theta) = \sum_{v \in V_L} \ell(\hat{y}_v,\, y_v), \qquad \hat{y}_v = \mathrm{softmax}(z_v) \in \mathbb{R}^C.$$

This is a *semi-supervised* setting: unlabeled nodes contribute through the graph structure but not through the loss.

4.2 Transductive vs. Inductive Learning

The same node classification task can be approached in two fundamentally different training regimes.

Definition 4.1 (Transductive Setting)**.** The *full graph* $G = (V, E)$ with all node features X is given as input. The node set is split into disjoint subsets $V = V_{\text{train}} \cup V_{\text{val}} \cup V_{\text{test}}$. Labels y_v are observed only for $v \in V_{\text{train}}$. At test time, predictions are made for $v \in V_{\text{test}}$ using the same fixed graph. Message passing uses the full adjacency A.

Definition 4.2 (Inductive Setting)**.** Only the subgraph $G[V_{\text{train}}]$ induced by training nodes is observed during training. At inference, new nodes (possibly from entirely new graphs) appear, together with their edges. Message passing at inference uses the augmented graph containing both training and test nodes.

Remark 4.3 (Practical consequence)**.** In the transductive setting, test node features flow into training node representations through message passing—a form of data leakage that is permissible since we assume that we know the features of the test nodes but not the labels. In the inductive setting, no test information is available at training time; the model must generalize to unseen nodes and graphs. GraphSAGE was designed for the inductive setting.

4.3 Graph-Level Prediction

For graph classification, the model must produce a *single* embedding for each graph. This requires a *readout* function that aggregates all node embeddings.

Definition 4.4 (Graph Readout)**.** After K message-passing layers producing $\{h_v^{(K)}\}_{v \in V}$, the *graph embedding* is

$$h_G = \text{READOUT}\Big(\{h_v^{(K)} : v \in V\}\Big) \in \mathbb{R}^{d_K}.$$

READOUT must be permutation invariant (Definition 2.2). Common choices: sum, mean, or max pooling over node embeddings.

The graph-level prediction is then $\hat{y}_G = \text{MLP}(h_G) \in \mathbb{R}^C$. Training minimizes $\mathcal{L}(\theta) = \sum_{i=1}^{N} \ell(\hat{y}_{G_i}, y_i)$ over the training graphs.

4.4 Link Prediction

For link prediction, the model computes node embeddings and then scores candidate pairs (u, v) with an *edge decoder*.

Dot-product decoder. The simplest decoder scores edge (u, v) by the inner product of node embeddings:

$$\hat{y}_{uv} = \sigma(h_u^\top h_v),$$

where σ is the sigmoid function. This encourages nodes that should be connected to have similar embeddings.

MLP decoder. A more expressive decoder concatenates the embeddings and applies an MLP:

$$\hat{y}_{uv} = \text{MLP}([h_u;\, h_v]) \in [0, 1].$$

Training. Because most pairs $(u, v) \notin E$ (negative examples), training uses *negative sampling*: for each positive edge $(u, v) \in E$, sample a small number of non-edges $(u, v') \notin E$ uniformly and train with binary cross-entropy. Evaluation uses AUC-ROC or Hits@k.

5 Spectral View of Graph Neural Networks

The spectral perspective reveals what GCN actually computes and connects it to classical signal processing on graphs.

5.1 Graph Signals

Definition 5.1 (Graph Signal)**.** A *graph signal* on $G = (V, E)$ is a function $x : V \to \mathbb{R}$, represented as a vector $x \in \mathbb{R}^n$ (with x_v the value at node v). A *multi-dimensional graph signal* is a matrix $X \in \mathbb{R}^{n \times d}$.

Intuitively, x assigns a numerical value to each node; the edge set determines what it means for x to be *smooth* across the graph.

5.2 The Graph Laplacian

Definition 5.2 (Graph Laplacian)**.** The *unnormalized graph Laplacian* is $L = D - A \in \mathbb{R}^{n \times n}$.

Proposition 5.3 (Quadratic Form)**.** *For any $x \in \mathbb{R}^n$:*

$$x^\top L x = \sum_{(u,v) \in E} (x_u - x_v)^2.$$

Proof.

$$x^\top L x = x^\top D x - x^\top A x = \sum_v d_v x_v^2 - \sum_{(u,v) \in E} 2 x_u x_v = \sum_{(u,v) \in E} (x_u^2 - 2 x_u x_v + x_v^2) = \sum_{(u,v) \in E} (x_u - x_v)^2. \quad \square$$

Corollary 5.4 (Positive Semi-Definiteness)**.** *L is positive semi-definite. Its smallest eigenvalue is $\lambda_1 = 0$, attained by the constant vector $\mathbf{1}$.*

Proof. $x^\top L x = \sum_{(u,v) \in E} (x_u - x_v)^2 \geq 0$ for all x, so $L \succeq 0$. Setting $x = \mathbf{1}$ gives $x^\top L x = 0$, so 0 is an eigenvalue. $\square$

The quadratic form measures the *total variation* of x across edges. A signal with small $x^\top L x$ changes slowly across edges (smooth); one with large $x^\top L x$ oscillates rapidly (rough).

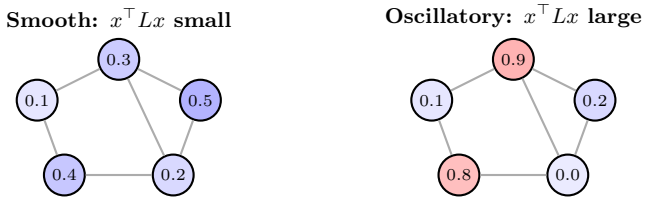


Figure 2: Left: a smooth graph signal—neighboring nodes have similar values, so $x^\top L x$ is small. Right: an oscillatory signal—neighboring nodes have very different values, so $x^\top L x$ is large.

5.3 Graph Fourier Transform

Since L is real symmetric and positive semi-definite, it admits an eigendecomposition:

$$L = U \Lambda U^\top, \qquad \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n), \quad 0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n,$$

where $U = [u_1 | \cdots | u_n] \in \mathbb{R}^{n \times n}$ is orthogonal ($U^\top U = I$).

Definition 5.5 (Graph Fourier Transform)**.** The *graph Fourier transform* (GFT) of a signal $x \in \mathbb{R}^n$ is

$$\hat{x} = U^\top x \in \mathbb{R}^n.$$

The inverse GFT is $x = U\hat{x}$. By orthogonality of U, the transform preserves ℓ^2-energy: $\|x\|_2^2 = \|\hat{x}\|_2^2$ (Parseval's identity).

Remark 5.6 (Spectral interpretation of the Laplacian)**.** The smoothness measure decomposes as

$$x^\top L x = \hat{x}^\top \Lambda \hat{x} = \sum_{i=1}^n \lambda_i \hat{x}_i^2.$$

Thus λ_i is the "frequency" of eigenvector u_i: a signal $x = u_i$ has total variation exactly λ_i. Small eigenvalues correspond to smooth (low-frequency) modes; large eigenvalues to oscillatory (high-frequency) modes.

Remark 5.7 (Connection to classical Fourier analysis)**.** On the cycle graph C_n (a ring of n nodes), the eigenvectors of L are the discrete complex exponentials $u_k = n^{-1/2}[\omega^{0k}, \omega^{1k}, \ldots, \omega^{(n-1)k}]^\top$ where $\omega = e^{2\pi i/n}$, and the GFT recovers the discrete Fourier transform.

5.4 Spectral Graph Convolution

Classical convolution on a regular domain is multiplication in the Fourier domain. Bruna et al. [6] define graph convolution analogously.

Definition 5.8 (Spectral Graph Convolution)**.** Given a learnable filter $g_\theta : \mathbb{R} \to \mathbb{R}$ applied pointwise to eigenvalues, the *spectral convolution* of signal x with filter g_θ is

$$y = g_\theta(L)\, x = U\, g_\theta(\Lambda)\, U^\top x, \qquad g_\theta(\Lambda) = \mathrm{diag}(g_\theta(\lambda_1), \ldots, g_\theta(\lambda_n))\,.$$

In the Fourier domain, this is simply $\hat{y}_i = g_\theta(\lambda_i)\hat{x}_i$.

Remark 5.9 (Limitations of direct spectral convolution)**.** This formulation requires (i) the full eigendecomposition of L, which costs $O(n^3)$ time and $O(n^2)$ space; (ii) n free parameters (one per frequency), which is expensive and does not generalize across graphs of different sizes; (iii) the resulting filter is not necessarily spatially localised.

5.5 Polynomial Filters and Locality

Definition 5.10 (Polynomial Filter)**.** A *polynomial filter* of degree K is defined by

$$g_\theta(\lambda) = \sum_{k=0}^{K} \theta_k \lambda^k, \qquad \theta = (\theta_0, \ldots, \theta_K) \in \mathbb{R}^{K+1}.$$

Then $g_\theta(L)x = \sum_{k=0}^{K} \theta_k L^k x$, and the filter is applied by repeated sparse matrix-vector multiplication, avoiding the eigendecomposition.

Proposition 5.11 (Locality of Polynomial Filters)**.** *If g_θ is a polynomial of degree K, then $(g_\theta(L)x)_v$ depends only on $\{x_u : d(u, v) \le K\}$, where $d(u, v)$ is the shortest-path distance.*

Proof. It suffices to show $(L^k x)_v$ depends only on K-hop neighbors. The sparsity pattern of L equals that of A (plus the diagonal). For any $k \ge 1$, $(L^k)_{vu} \neq 0$ only if there exists a walk of length k from v to u, which requires $d(u, v) \le k$. By induction on k, the claim follows. □

Remark 5.12 (Chebyshev expansion of the Laplacian)**.** Defferrard et al. [7] parameterise g_θ using Chebyshev polynomials T_k: $g_\theta(\lambda) = \sum_{k=0}^{K} \theta_k T_k(\tilde{\lambda})$, where $\tilde{\lambda} = 2\lambda/\lambda_{\max} - 1 \in [-1, 1]$. This ensures numerical stability and avoids computing L^k directly via the three-term recurrence $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$.

5.6 GCN as a First-Order Polynomial Filter

Kipf and Welling [4] simplify the Chebyshev expansion to a $K = 1$ polynomial and apply a renormalization trick.

Derivation. A first-order Chebyshev filter on the normalized Laplacian $L_{\mathrm{sym}} = I - D^{-1/2}AD^{-1/2}$ (whose eigenvalues lie in $[0, 2]$) gives

$$g_\theta(L_{\mathrm{sym}})x = \theta_0 x + \theta_1(L_{\mathrm{sym}} - I)x = \theta_0 x - \theta_1 D^{-1/2}AD^{-1/2}x.$$

Setting $\theta_0 = -\theta_1 = \theta$ yields $g_\theta(L_{\mathrm{sym}})x = \theta(I + D^{-1/2}AD^{-1/2})x$. The matrix $I + D^{-1/2}AD^{-1/2}$ has eigenvalues in $[0, 2]$; stacking many such layers causes exploding or vanishing signals. The *renormalization trick* replaces it with the augmented normalized matrix:

$$I + D^{-1/2}AD^{-1/2} \;\longrightarrow\; \widehat{A} = \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}, \qquad \widetilde{A} = A + I,$$

which has eigenvalues in $(-1, 1]$ (Proposition 5.13). Adding the learnable projection $W^{(l)}$ and a nonlinearity recovers Definition 3.1.

Proposition 5.13 (Spectral Properties of $\widehat{A}$)**.** *The eigenvalues $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_n$ of $\widehat{A}$ lie in $(-1, 1]$, with $\mu_1 = 1$ (eigenvector $\widetilde{D}^{1/2}\mathbf{1}$). The i-th eigenvector of $\widehat{A}$ coincides with that of $\widetilde{L}_{\text{sym}} = I - \widehat{A}$: smooth (low-frequency) eigenvectors have μ_i close to 1; oscillatory (high-frequency) eigenvectors have μ_i close to -1.*

Proof. Since $\widehat{A} = I - \widetilde{L}_{\text{sym}}$, we have $\mu_i = 1 - \tilde{\lambda}_i$ where $\tilde{\lambda}_i$ are the eigenvalues of $\widetilde{L}_{\text{sym}}$. Observe that $\widetilde{L} = \widetilde{D} - \widetilde{A} = (D+I) - (A+I) = L$, so $\widetilde{L}_{\text{sym}} = \widetilde{D}^{-1/2} L \widetilde{D}^{-1/2}$. Setting $y = \widetilde{D}^{-1/2} x$ and using Proposition 5.3:

$$x^\top \widetilde{L}_{\text{sym}}\, x = y^\top L y = \sum_{(u,v)\in E} (y_u - y_v)^2 \geq 0,$$

so $\tilde{\lambda}_i \geq 0$ and $\mu_i \leq 1$. For the upper bound, use $(y_u - y_v)^2 \leq 2(y_u^2 + y_v^2)$:

$$y^\top L y \leq 2 \sum_v d_v y_v^2 < 2 \sum_v \tilde{d}_v y_v^2 = 2\, x^\top x,$$

where the strict inequality uses $d_v < \tilde{d}_v = d_v + 1$. Hence $\tilde{\lambda}_i < 2$ and $\mu_i > -1$. □

Remark 5.14 (Low-pass filtering and oversmoothing)**.** Each GCN layer multiplies frequency i by μ_i. Since $\mu_1 = 1$, the smooth DC component is preserved exactly, while $|\mu_i| < 1$ for all $i \geq 2$, so every other frequency is attenuated. After K layers the gain becomes $\mu_i^K \to 0$ for $i \geq 2$: all non-DC components are driven to zero, leaving only the constant-per-component signal. This is the spectral mechanism behind oversmoothing (Section 6.1).

6 Limitations and Extensions

6.1 Oversmoothing

Each GCN layer multiplies by the low-pass filter $\widehat{A}$. Stacking many layers therefore applies low-pass filtering repeatedly, progressively removing high-frequency (discriminative) components from node embeddings. In the limit, embeddings become indistinguishable.

Proposition 6.1 (Oversmoothing in the Linear GCN)**.** *Consider the linear GCN (no nonlinearity): $H^{(K)} = \widehat{A}^K X W$. As $K \to \infty$, every row of $\widehat{A}^K X$ converges to a vector that is constant within each connected component of G.*

Proof. Write $\widehat{A} = U_{\widehat{A}} \Lambda_{\widehat{A}} U_{\widehat{A}}^\top$ where eigenvalues $\mu_1 = 1 \geq \mu_2 \geq \cdots \geq \mu_n > -1$. Then $\widehat{A}^K = U_{\widehat{A}} \Lambda_{\widehat{A}}^K U_{\widehat{A}}^\top$. Since $|\mu_i| < 1$ for $i \geq 2$, $\Lambda_{\widehat{A}}^K \to \text{diag}(1, 0, \ldots, 0)$, so $\widehat{A}^K \to u_1 u_1^\top$ where u_1 is the dominant eigenvector. For a connected graph, $u_1 \propto \widetilde{D}^{1/2}\mathbf{1}$, so $(\widehat{A}^K X)_{v,:} \to c\, \tilde{d}_v^{1/2} \cdot \mathbf{r}^\top$ for some fixed vector $\mathbf{r}$ and constant c. All rows converge to a scalar multiple of the same vector. □

Remark 6.2 (Spectral interpretation)**.** Oversmoothing is low-pass filtering taken to the extreme: with enough layers, only the DC component (constant signal on each connected component) survives. The rate of convergence is governed by the *spectral gap* $1 - \mu_2$: a graph with a large spectral gap oversmooths faster.

Remark 6.3 (Practical consequence)**.** Most effective GNNs use $K = 2$–4 layers. Remedies include residual connections, jumping knowledge networks (concatenating embeddings from all layers), and DropEdge.

6.2 Mini-Batch GNNs for Scalability

Full-batch training requires loading the entire feature matrix $H^{(l)} \in \mathbb{R}^{n \times d_l}$ and the normalized adjacency $\widehat{A} \in \mathbb{R}^{n \times n}$ at every gradient step, which becomes impractical on large graphs. The natural solution would be to select a mini-batch $\mathcal{B} \subset V$ of target nodes and compute only their embeddings — but this is harder than in standard deep learning, because nodes are connected by edges. Computing $h_v^{(K)}$ for a node $v \in \mathcal{B}$

requires the features of all nodes within K hops of v (Proposition 2.6), whose count can grow exponentially with K (*neighborhood explosion*). Several sampling strategies have been proposed to address this; we review the main families below.

6.2.1 Neighborhood Sampling

Introduced with GraphSAGE [5], neighborhood sampling fixes a *fanout* s and, for each active node v at each layer, draws s neighbors at random instead of aggregating over the full $\mathcal{N}(v)$:

$$h_v^{(l+1)} = \sigma\left(\text{CONCAT}\left(h_v^{(l)},\ \frac{1}{s}\sum_{u\in\mathcal{S}^{(l)}(v)} h_u^{(l)}\right) W^{(l)}\right), \qquad \mathcal{S}^{(l)}(v)\subset\mathcal{N}(v),\ |\mathcal{S}^{(l)}(v)| = s.$$

This bounds the per-node cost but does *not* eliminate the exponential growth: a K-layer GNN on a single target node still accesses up to s^K nodes, one per leaf of the sampled computation tree. In practice this forces shallow networks ($K = 2$ or 3) with moderate fanout. PinSage [8] deployed this approach at web scale with $s = 25$ on a graph of billions of nodes.

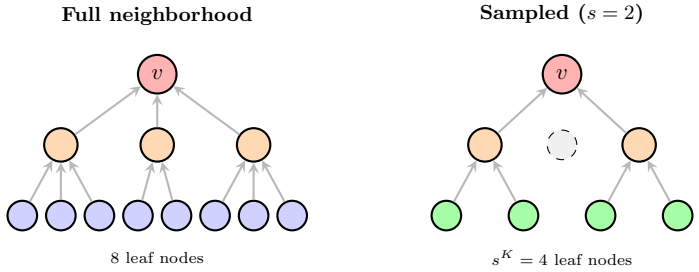


Figure 3: Neighborhood sampling with $K = 2$, $s = 2$. The full tree (left) has $3 \times 3 = 9$ layer-2 nodes; sampling $s = 2$ neighbors per node (right) reduces this to $s^2 = 4$, but growth is still exponential in K.

6.2.2 Subgraph-Based Methods

A conceptually different approach samples an entire connected subgraph $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ and runs *full* message passing within it. Because aggregation never leaves $G_{\mathcal{S}}$, the cost scales linearly with $|V_{\mathcal{S}}|$ and the exponential growth is entirely avoided. Two influential methods differ in how the subgraph is constructed.

ClusterGCN [9]. The graph is first partitioned into p balanced clusters $\{C_1, \ldots, C_p\}$ using METIS [10], a multilevel partitioning algorithm that minimizes the number of edges crossing cluster boundaries (*edge cut*). At each training step, one or more clusters are selected and the GCN is run on the induced subgraph. Inter-cluster edges are discarded, introducing a bias for nodes at cluster boundaries; minimizing the edge cut with METIS keeps this bias small. In practice, sampling several clusters per batch and retaining the inter-cluster edges within the batch reduces the bias further.

GraphSAINT [11]. Instead of a fixed partition, GraphSAINT samples a new subgraph at every training step. Three samplers are proposed: *node sampling* (draw r nodes uniformly, take the induced subgraph), *edge sampling* (draw r edges, take the endpoint subgraph), and *random walk sampling* (run r walks of length w, take all visited nodes). Random walk sampling tends to produce denser, better-connected subgraphs, giving lower variance gradient estimates. To correct for the non-uniform inclusion probabilities, GraphSAINT reweights each node's loss contribution by $1/p(v)$ where $p(v)$ is the probability that v is included in the sampled subgraph, making the loss estimator unbiased.

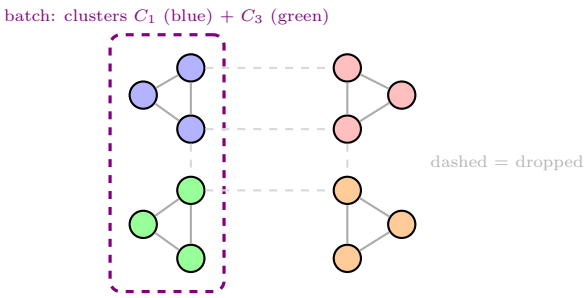


Figure 4: ClusterGCN on a graph partitioned into 4 clusters. The mini-batch selects C_1 and C_3; full GCN runs on the induced subgraph (solid edges). Inter-cluster edges (dashed) are dropped.

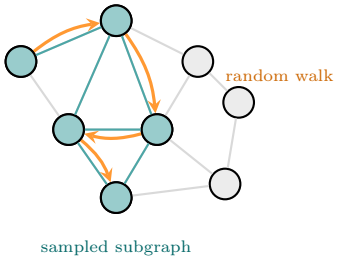


Figure 5: GraphSAINT with random walk sampling. The walk (orange arrows) visits 5 nodes (teal); the induced subgraph on those nodes (teal edges) is the mini-batch. Nodes and edges not visited (gray) are excluded.

6.2.3 Layer-Wise Sampling

A third family avoids both the per-node exponential cost of neighborhood sampling and the need for a global graph partition. Instead of growing an independent computation tree per target node, layer-wise methods draw a *single shared pool* $\mathcal{S}^{(l)}$ of nodes at each layer, used as the message source for all nodes at layer $l+1$. The total number of nodes accessed is thus $O(Kt)$ where $t = |\mathcal{S}^{(l)}|$, linear in depth.

FastGCN [12]. FastGCN draws the pool $\mathcal{S}^{(l)}$ for each layer independently of the other layers, using importance sampling (higher probability for nodes that contribute more to the aggregation) to reduce variance within a layer. However, because the pools are not coordinated across layers, many nodes in $\mathcal{S}^{(l)}$ may not be neighbors of any node in $\mathcal{S}^{(l+1)}$, contributing nothing and inflating variance across layers.

LADIES [13]. LADIES (Layer-Dependent Importance Sampling) fixes this by building the pools top-down: starting from the target batch $\mathcal{B}$, the pool for layer l is sampled from the *neighborhood* of the pool for layer $l+1$. This guarantees every sampled node at layer l contributes at least one message to the layer above, substantially reducing variance compared to FastGCN while keeping the same $O(Kt)$ cost.

6.3 Heterophily

GCN's mean aggregation is implicitly designed for the *homophilic* setting, where connected nodes tend to share the same label. Many real-world graphs are instead *heterophilic*: edges frequently connect nodes of different classes.

Definition 6.4 (Homophily Ratio)**.** For a node classification task with labels $\{y_v\}_{v \in V}$, the *homophily ratio*

is

$$h = \frac{|\{(u,v) \in E : y_u = y_v\}|}{|E|} \in [0,1].$$

$h \approx 1$: strongly homophilic (same-class edges dominate). $h \approx 0$: strongly heterophilic (different-class edges dominate).

Example 6.5. The Cora citation network has $h \approx 0.81$ (papers mostly cite papers in the same field). In contrast, the WebKB dataset (web pages linking between student and faculty pages) has $h \approx 0.22$.

Remark 6.6 (Heterophily and GNNs)**.** Heterophily does not mean that the graph structure is uninformative: a node systematically surrounded by nodes of a different class still carries a signal that a GNN can in principle exploit. Whether low homophily ratio genuinely hurts GNN performance in practice is debated; Platonov et al. [14] show that many conclusions from the heterophily literature are sensitive to benchmark choice and that the picture is less clear than often presented.

6.4 Long-Range Dependencies

A K-layer GNN can only aggregate information from nodes within K hops (Proposition 2.6). When relevant signal originates far away in the graph, increasing K is the natural fix — but this runs directly into over-smoothing (Section 6.1). Capturing long-range dependencies without losing local discriminative information remains an open challenge for message-passing GNNs.

6.5 Attention-Based GNNs

6.5.1 Graph Attention Networks (GAT)

GAT [15] replaces the fixed normalization of GCN with *learnable attention weights* computed from node features.

Definition 6.7 (GAT Layer)**.** Let $W \in \mathbb{R}^{d \times d'}$ be a linear projection and $a \in \mathbb{R}^{2d'}$ an attention vector. The *unnormalized attention score* for edge (v,u) is

$$e_{vu} = \mathrm{LeakyReLU}\big(a^\top [Wh_v \,\|\, Wh_u]\big),$$

where $\|$ denotes concatenation. The *attention coefficient* is

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{u' \in \mathcal{N}(v)} \exp(e_{vu'})}.$$

The updated embedding is

$$h'_v = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}\, W h_u\right).$$

Remark 6.8 (Multi-head attention)**.** In practice, M independent attention heads are computed in parallel and their outputs concatenated or averaged:

$$h'_v = \Big\|_{m=1}^{M} \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(m)}\, W^{(m)} h_u\right).$$

Multi-head attention stabilizes training and captures diverse neighborhood features simultaneously.

Remark 6.9 (Expressiveness)**.** GAT is still a local message-passing GNN: attention is only over $\mathcal{N}(v)$, not over all nodes. A node with a clearly irrelevant neighbor can assign it a near-zero weight, unlike GCN which uses the fixed normalization $1/\sqrt{\tilde{d}_v \tilde{d}_u}$.

6.5.2 Graph Transformers

Graph Transformers apply self-attention over *all pairs* of nodes, not just neighboring ones, enabling full global information exchange in a single layer.

The attention score for pair (v, u) is

$$e_{vu} = \frac{(Qh_v)^\top (Kh_u)}{\sqrt{d'}}, \qquad \alpha_{vu} = \mathrm{softmax}_{u' \in V}(e_{vu'}),$$

and the update is $h'_v = \sum_{u \in V} \alpha_{vu} V h_u$ (where Q, K, V are query/key/value projections, and we slightly abuse notation by using V for the value matrix). This is identical to a standard Transformer block applied to the set of node embeddings.

Remark 6.10 (Cost and structural encoding)**.** All-pairs attention costs $O(n^2)$ per layer, versus $O(m)$ for local message passing. On graphs with $n \gg 10^4$ this is prohibitive without sparsification. Moreover, the attention mechanism is agnostic to graph structure; structural information must be injected via *positional encodings*: Laplacian eigenvectors, random-walk landing probabilities, or shortest-path distances are common choices. SGFormer [16] replaces quadratic all-pairs attention with a linear-complexity global attention mechanism that aggregates information through global feature summaries rather than explicitly computing attention scores between every pair of nodes, reducing the per-layer complexity from $O(n^2)$ to $O(n)$ while retaining the ability to capture long-range interactions.

7 Applications

7.1 Board Games: AlphaZero with GNNs

AlphaZero [17] evaluates a board position with a neural network $f_\theta(\text{state}) \to (\text{policy}, \text{value})$, where the policy head selects moves and the value head estimates the win probability. The original architecture uses CNNs on the regular grid of games like Go and chess.

GNN extension. Ben-Assayag and El-Yaniv [18] replace the CNN backbone with a GNN. The board is encoded as a graph: positions are nodes, and spatial or logical adjacencies define edges. Message passing aggregates information from adjacent positions, analogously to a convolutional filter on the grid.

Advantage. A GNN-based policy generalizes across board sizes without retraining, since the same message-passing function applies regardless of the graph structure. In contrast, a CNN architecture trained on a 9×9 Go board cannot be directly applied to a 19×19 board.

7.2 Routing and ETA Prediction: Google Maps

Graph encoding. Derrow-Pinion et al. [19] model the road network as a graph where road segments are nodes and intersections define edges. Node features include historical speed data, road type, and time of day.

Message passing. A GNN propagates delay information along the road network. If a segment experiences congestion, its embedding is updated and this information flows to neighboring segments via message passing, producing accurate downstream delay estimates. The loss is the mean absolute error between predicted and observed arrival times.

Impact. The GNN-based system was deployed in Google Maps and reported up to 50% improvement in ETA accuracy in cities with complex road networks.

7.3 Recommender Systems: PinSage

Bipartite graph. Ying et al. [8] model the Pinterest platform as a bipartite graph: one set of nodes represents *boards* (collections curated by users) and the other represents *pins* (images). An edge (b, p) exists if pin p is saved to board b.

GNN architecture. PinSage applies GraphSAGE-style message passing on this bipartite graph to learn pin embeddings. A new or sparse pin can obtain an embedding from its neighborhood in the graph (the boards that saved it and the other pins on those boards), even with minimal direct interaction history.

Scalability. The graph has billions of nodes and edges. PinSage uses neighborhood sampling (Section 6.2.1) to keep training tractable, and curriculum learning (gradually increasing the difficulty of negative samples) to improve embedding quality.

References

[1] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *ICLR*, 2015.

[2] McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[3] Wu, Z. et al. MoleculeNet: A benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

[4] Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

[5] Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. *NeurIPS*, 2017.

[6] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *ICLR*, 2014.

[7] Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. *NeurIPS*, 2016.

[8] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. *KDD*, 2018.

[9] Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. *KDD*, 2019.

[10] Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[11] Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. GraphSAINT: Graph sampling based inductive learning method. *ICLR*, 2020.

[12] Chen, J., Ma, T., and Xiao, C. FastGCN: Fast learning with graph convolutional networks via importance sampling. *ICLR*, 2018.

[13] Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., and Gu, Q. Layer-dependent importance sampling for training deep and large graph convolutional networks. *NeurIPS*, 2019.

[14] Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. Revisiting heterophily for graph neural networks. *NeurIPS*, 2023.

[15] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *ICLR*, 2018.

[16] Wu, Q., Zhao, W., Yang, C., Zhang, H., Nie, F., Jiang, H., Bian, J., and Zhou, J. SGFormer: Simplifying and empowering transformers for large-graph representations. *NeurIPS*, 2023.

[17] Silver, D. et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[18] Ben-Assayag, G. and El-Yaniv, R. Train on small, play the large: Scaling up board games with AlphaZero and GNN. *arXiv:2110.12359*, 2021.

[19] Derrow-Pinion, A. et al. ETA prediction with graph neural networks in Google Maps. *CIKM*, 2021.