Graph Neural Networks for Node Classification

Why Node Classification on Graphs?

Real-world graphs: social networks, web pages, proteins.

Task: predict a node’s category using both features and connections.

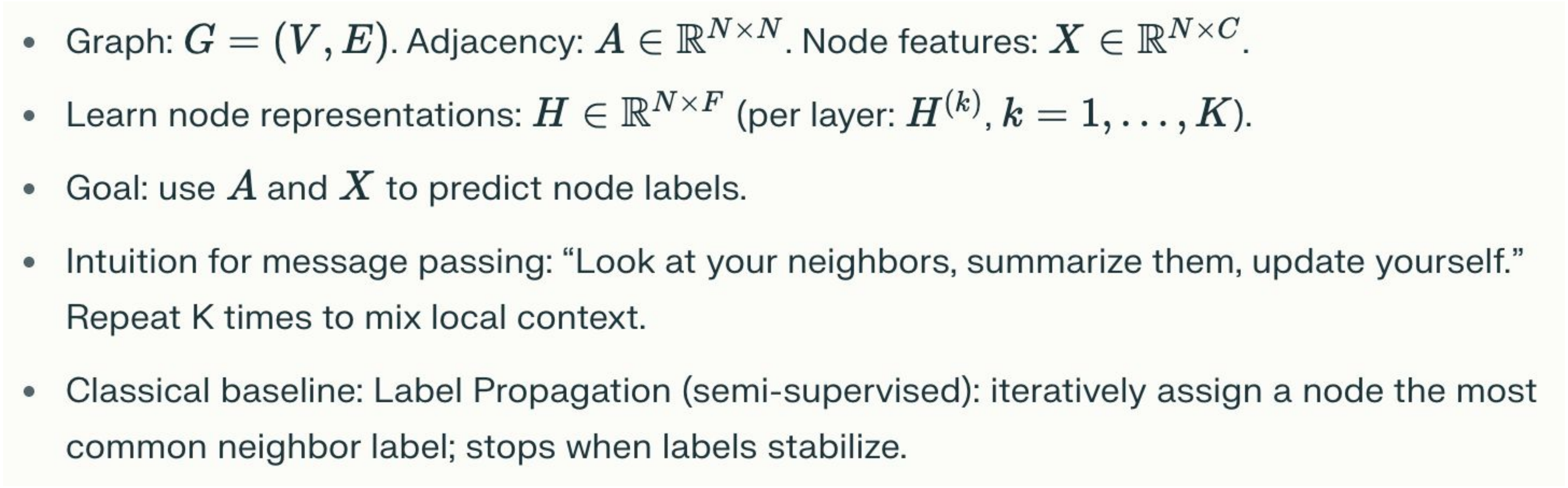
Intuition: “You are who you connect with.” Neighbors’ information helps classify you.

Examples:

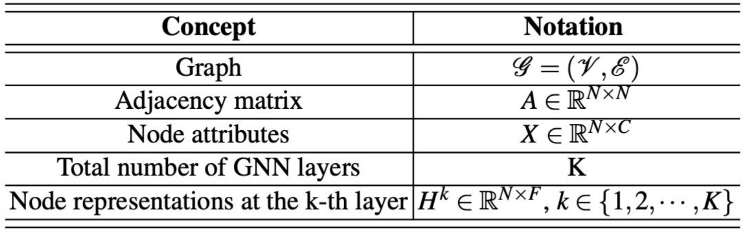
- Social: predict a user’s political leaning from friends’ leanings and profile features.
- Web: classify a page’s topic using its content and hyperlinks.
- Biology: infer a protein’s function from its amino-acid features and interaction partners.

Key idea: better node representations → better classification.

Problem Setup and Notation (Intuition → Math)



- Graph: $G = (V, E)$. Adjacency: $A \in \mathbb{R}^{N \times N}$. Node features: $X \in \mathbb{R}^{N \times C}$.
- Learn node representations: $H \in \mathbb{R}^{N \times F}$ (per layer: $H^{(k)}, k = 1, \ldots, K$).
- Goal: use A and X to predict node labels.
- Intuition for message passing: "Look at your neighbors, summarize them, update yourself." Repeat K times to mix local context.
- Classical baseline: Label Propagation (semi-supervised): iteratively assign a node the most common neighbor label; stops when labels stabilize.



Concept	Notation
Graph	$\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Adjacency matrix	$A \in \mathbb{R}^{N \times N}$
Node attributes	$X \in \mathbb{R}^{N \times C}$
Total number of GNN layers	K
Node representations at the k-th layer	$H^k \in \mathbb{R}^{N \times F}, k \in \{1, 2, \cdots, K\}$

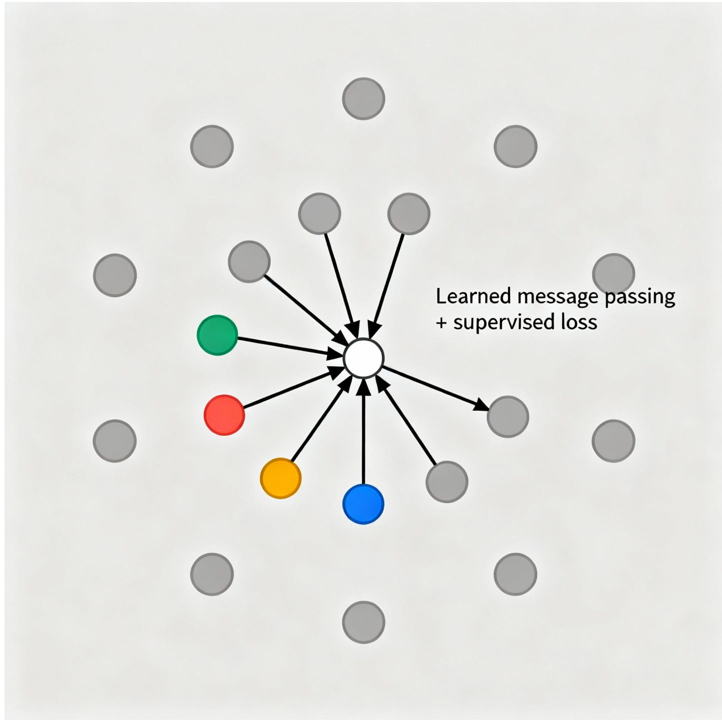
Intuition: What are supervised GNNs?

Real-world setup: we have a graph (users linked by friendships, web pages linked by hyperlinks, proteins linked by interactions) and a few labeled nodes.

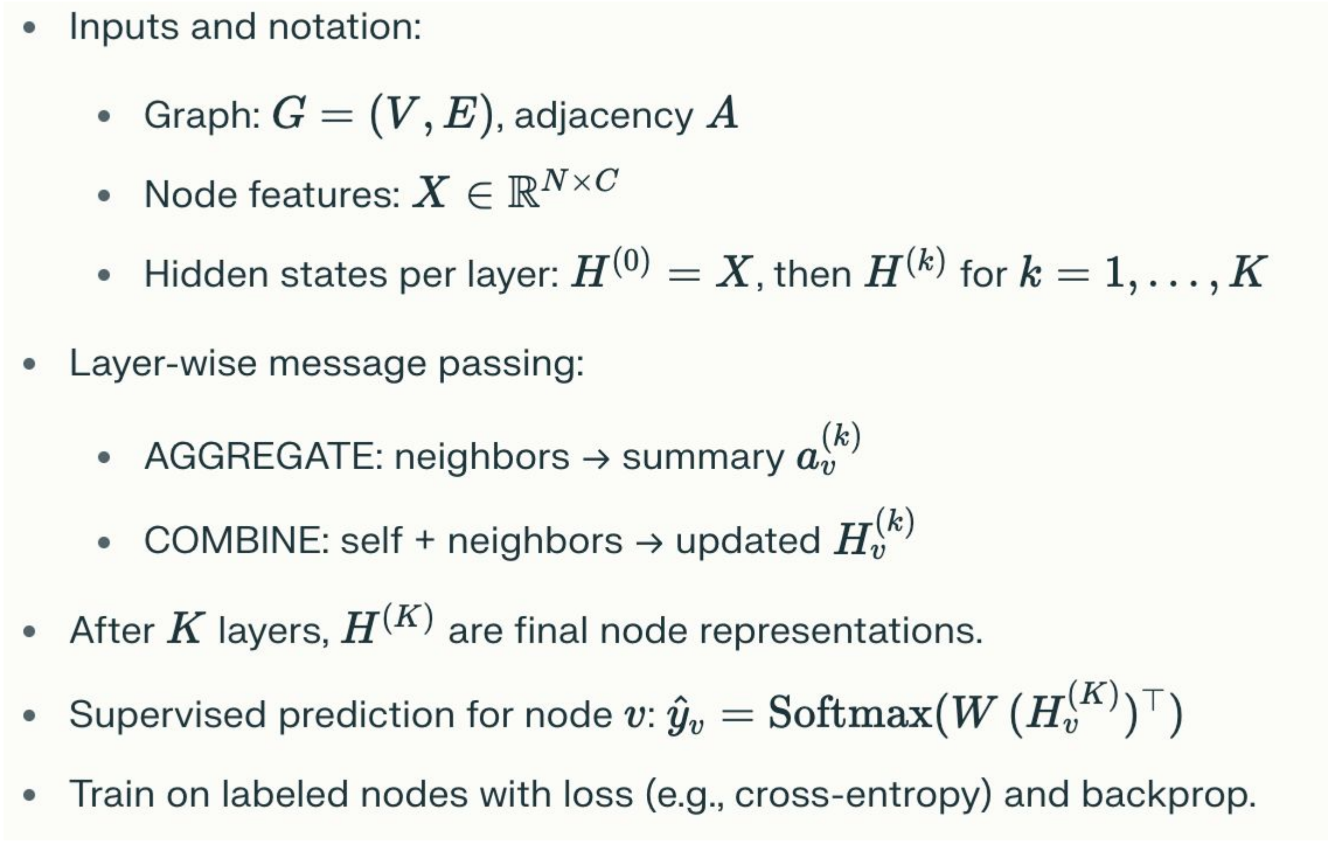
Goal: predict labels for unlabeled nodes by using both node features and connections.

Core intuition: "You are influenced by who you connect with." Each node looks at neighbors, summarizes them, and updates itself—repeated K times.

Supervised twist: we compare predictions on labeled nodes with their true labels to learn the best way to aggregate and combine.

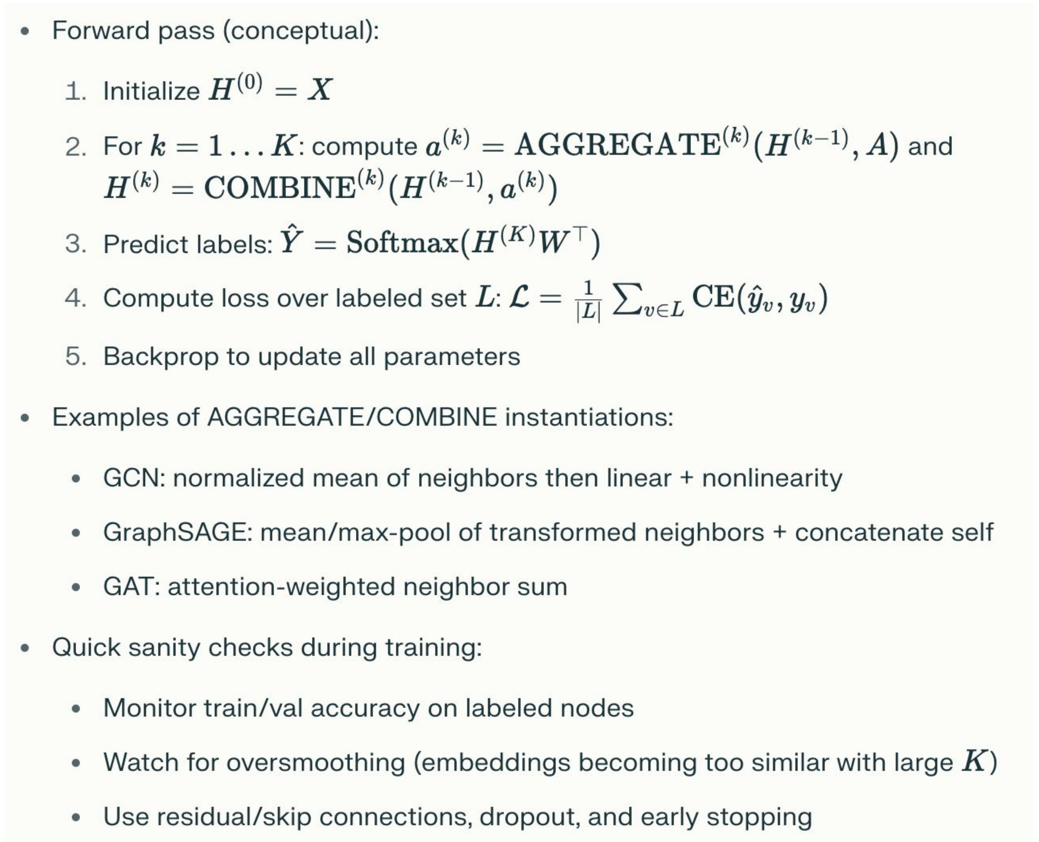


General Framework (from intuition to equations)



- Inputs and notation:
  - Graph: $G = (V, E)$, adjacency A
  - Node features: $X \in \mathbb{R}^{N \times C}$
  - Hidden states per layer: $H^{(0)} = X$, then $H^{(k)}$ for $k = 1, \ldots, K$
- Layer-wise message passing:
  - AGGREGATE: neighbors → summary $a_v^{(k)}$
  - COMBINE: self + neighbors → updated $H_v^{(k)}$
- After K layers, $H^{(K)}$ are final node representations.
- Supervised prediction for node v: $\hat{y}_v = \mathrm{Softmax}(W\,(H_v^{(K)})^\top)$
- Train on labeled nodes with loss (e.g., cross-entropy) and backprop.

Concrete flow: forward pass to training loop



- Forward pass (conceptual):
  1. Initialize $H^{(0)} = X$
  2. For $k = 1 \ldots K$: compute $a^{(k)} = \mathrm{AGGREGATE}^{(k)}(H^{(k-1)}, A)$ and $H^{(k)} = \mathrm{COMBINE}^{(k)}(H^{(k-1)}, a^{(k)})$
  3. Predict labels: $\hat{Y} = \mathrm{Softmax}(H^{(K)} W^{\top})$
  4. Compute loss over labeled set L: $\mathcal{L} = \frac{1}{|L|} \sum_{v \in L} \mathrm{CE}(\hat{y}_v, y_v)$
  5. Backprop to update all parameters
- Examples of AGGREGATE/COMBINE instantiations:
  - GCN: normalized mean of neighbors then linear + nonlinearity
  - GraphSAGE: mean/max-pool of transformed neighbors + concatenate self
  - GAT: attention-weighted neighbor sum
- Quick sanity checks during training:
  - Monitor train/val accuracy on labeled nodes
  - Watch for oversmoothing (embeddings becoming too similar with large K)
  - Use residual/skip connections, dropout, and early stopping