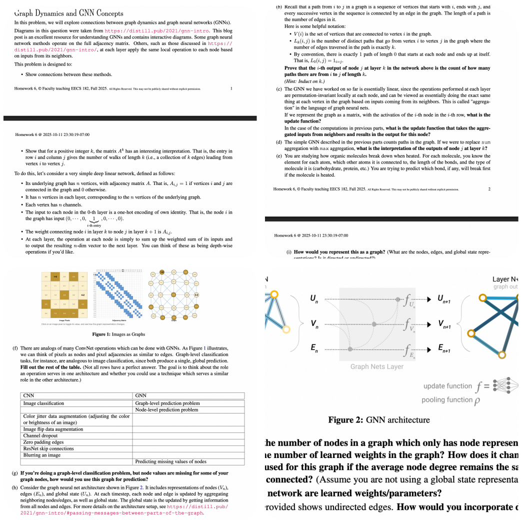


Graph Dynamics and GNN Concepts

In this problem, we will explore connections between graph dynamics and graph neural networks (GNNs).

Diagrams in this question were taken from https://distill.pub/2021/gnn-intro. This blog post is an excellent resource for understanding GNNs and contains interactive diagrams. Some graph neural network methods operate on the full adjacency matrix. Others, such as those discussed in https://distill.pub/2021/gnn-intro/, at each layer apply the same local operation to each node based on inputs from its neighbors.

This problem is designed to:

- Show connections between these methods.

Homework 6, © Faculty teaching EECS 182, Fall 2025. All Rights Reserved. This may not be publicly shared without explicit permission. 1

Homework 6 @ 2025-10-11 23:30:19-07:00

- Show that for a positive integer k, the matrix A^k has an interesting interpretation. That is, the entry in row i and column j gives the number of walks of length k (i.e., a collection of k edges) leading from vertex i to vertex j.

To do this, let's consider a very simple deep linear network, defined as follows:

- Its underlying graph has n vertices, with adjacency matrix A. That is, $A_{i,j} = 1$ if vertices i and j are connected in the graph and 0 otherwise.
- It has n vertices in each layer, corresponding to the n vertices of the underlying graph.
- Each vertex has n channels.
- The input to each node in the 0-th layer is a one-hot encoding of own identity. That is, the node i in the graph has input $(0, \cdots, 0, \underbrace{1}_{i\text{-th entry}}, 0, \cdots, 0)$.
- The weight connecting node i in layer k to node j in layer $k+1$ is $A_{i,j}$.
- At each layer, the operation at each node is simply to sum up the weighted sum of its inputs and to output the resulting n-dim vector to the next layer. You can think of these as being depth-wise operations if you'd like.

Image Pixels
Adjacency Matrix
Graph
Click on an image pixel to toggle its value, and see how the graph representation changes.

Figure 1: Images as Graphs

(f) There are analogs of many ConvNet operations which can be done with GNNs. As Figure 1 illustrates, we can think of pixels as nodes and pixel adjacencies as similar to edges. Graph-level classification tasks, for instance, are analogous to image classification, since both produce a single, global prediction. **Fill out the rest of the table.** (Not all rows have a perfect answer. The goal is to think about the role an operation serves in one architecture and whether you could use a technique which serves a similar role in the other architecture.)

CNN	GNN
Image classification	Graph-level prediction problem
	Node-level prediction problem
Color jitter data augmentation (adjusting the color or brightness of an image)	
Image flip data augmentation	
Channel dropout	
Zero padding edges	
ResNet skip connections	
Blurring an image	
	Predicting missing values of nodes

(g) **If you're doing a graph-level classification problem, but node values are missing for some of your graph nodes, how would you use this graph for prediction?**

(h) Consider the graph neural net architecture shown in Figure 2. It includes representations of nodes (V_n), edges (E_n), and global state (U_n). At each timestep, each node and edge is updated by aggregating neighboring nodes/edges, as well as global state. The global state is the updated by getting information from all nodes and edges. For more details on the architecture setup, see https://distill.pub/2021/gnn-intro/#passing-messages-between-parts-of-the-graph.

(b) Recall that a path from i to j in a graph is a sequence of vertices that starts with i, ends with j, and every successive vertex in the sequence is connected by an edge in the graph. The length of a path is the number of edges in it.

Here is some helpful notation:

- $V(i)$ is the set of vertices that are connected to vertex i in the graph.
- $L_k(i,j)$ is the number of distinct paths that go from vertex i to vertex j in the graph where the number of edges traversed in the path is exactly k.
- By convention, there is exactly 1 path of length 0 that starts at each node and ends up at itself. That is, $L_0(i,j) = 1_{i=j}$.

Prove that the i-th output of node j at layer k in the network above is the count of how many paths there are from i to j of length k.

(Hint: Induct on k.)

(c) The GNN we have worked on so far is essentially linear, since the operations performed at each layer are permutation-invariant locally at each node, and can be viewed as essentially doing the exact same thing at each vertex in the graph based on inputs coming from its neighbors. This is called "aggregation" in the language of graph neural nets.

If we represent the graph as a matrix, with the activatios of the i-th node in the i-th row, **what is the update function?**

In the case of the computations in previous parts, **what is the update function that takes the aggregated inputs from neighbors and results in the output for this node?**

(d) The simple GNN described in the previous parts counts paths in the graph. If we were to replace `sum` aggregation with `max` aggregation, **what is the interpretation of the outputs of node j at layer k?**

(e) You are studying how organic molecules break down when heated. For each molecule, you know the element for each atom, which other atoms it is connected to, the length of the bonds, and the type of molecule it is (carbohydrate, protein, etc.) You are trying to predict which bond, if any, will break first if the molecule is heated.

Homework 6, © Faculty teaching EECS 182, Fall 2025. All Rights Reserved. This may not be publicly shared without explicit permission. 2

Homework 6 @ 2025-10-11 23:30:19-07:00

(i) **How would you represent this as a graph?** (What are the nodes, edges, and global state representations? Is it directed or undirected?)

Figure 2: GNN architecture

he number of nodes in a graph which only has node represen
e number of learned weights in the graph? How does it chan
used for this graph if the average node degree remains the sa
connected? (Assume you are not using a global state representa
network are learned weights/parameters?
rovided shows undirected edges. **How would you incorporate**

Even with minimal prompting, the extended thinking time allows the model to sort through all of the problems anyway

Please read these screenshots in detail. Your job is to write solutions to each part of this question. Make sure to delineate which subpart you are working on, and make sure to show your work adequately

Thought for 7m ...

I'll label the output of node j at layer k as $h_j^{(k)} \in \mathbb{R}^n$.

Let $A \in \mathbb{R}^{n\times n}$ be the adjacency matrix and let $H^{(k)} \in \mathbb{R}^{n\times n}$ be the matrix whose j-th row is $h_j^{(k)\top}$.

(a) Output of the j-th node at layer k in terms of A

Inputs (layer 0):

- Node i has input $h_i^{(0)} = e_i$, the i-th standard basis vector in $\mathbb{R}^n$.
- Thus $H^{(0)} = I_n$.

Update rule (sum aggregation with weights A_{ij}):

$$h_j^{(k+1)} = \sum_{i=1}^{n} A_{ij} h_i^{(k)}.$$

In matrix form, with "nodes = rows":

$$H^{(k+1)} = A^\top H^{(k)}.$$

Starting from $H^{(0)} = I$,

$$H^{(k)} = (A^\top)^k H^{(0)} = (A^\top)^k.$$

So the output of node j at layer k is the j-th row of $(A^\top)^k$, i.e.

$$h_j^{(k)} = \mathrm{row}_j((A^\top)^k).$$

Equivalently, as a column vector,

$$h_j^{(k)\top} = A^k e_j,$$

so $h_j^{(k)}$ is the j-th column of A^k.

Correct.

(b) Show that the i-th output of node j at layer k equals the number of length-k paths from $itoj$

Let $L_k(i, j)$ be the number of distinct paths of length exactly k from i to j.
From part (a),

$$(h_j^{(k)})_i = (A^k)_{ij}.$$

We'll show $(A^k)_{ij} = L_k(i, j)$ by induction on k.

Base case $k = 0$.
By convention, there is exactly one path of length 0 from a node to itself and none to any other node:

$$L_0(i, j) = \mathbf{1}_{i=j}.$$

Also $A^0 = I$, so

$$(A^0)_{ij} = \delta_{ij} = \mathbf{1}_{i=j} = L_0(i, j).$$

Inductive step. Assume for some $k \geq 0$ that

$$(A^k)_{i\ell} = L_k(i, \ell) \quad \text{for all } i, \ell.$$

Consider $k + 1$:

$$A^{k+1} = A^k A \quad \Rightarrow \quad (A^{k+1})_{ij} = \sum_{\ell=1}^{n} (A^k)_{i\ell} A_{\ell j}.$$

Interpretation of the sum:

- $A_{\ell j} = 1$ iff there is an edge (path of length 1) from ℓ to j.
- $(A^k)_{i\ell} = L_k(i, \ell)$ is the number of paths of length k from i to ℓ.

Thus each term $(A^k)_{i\ell}A_{\ell j}$ counts the number of paths of length $k + 1$ that go from i to ℓ (length k), then from ℓ to j (one more edge). Summing over all intermediate nodes ℓ counts **all** length-$k + 1$ paths from i to j:

$$(A^{k+1})_{ij} = \sum_{\ell} L_k(i, \ell)A_{\ell j} = L_{k+1}(i, j).$$

Therefore, for all k,

$$(h_j^{(k)})_i = (A^k)_{ij} = L_k(i, j),$$

so the i-th output channel of node j at layer k is exactly the number of length-k paths from i to j.

Correct.

(c) Update function in matrix form

We already wrote the layer-to-layer update as

$$H^{(k+1)} = A^\top H^{(k)},$$

where the i-th row of $H^{(k)}$ is the activation of node i at layer k.

At the **node level**, the aggregation and update for node j are

$$\underbrace{m_j^{(k)}}_{\text{aggregated message}} = \sum_{i \in V(j)} A_{ij} h_i^{(k)}, \qquad \underbrace{h_j^{(k+1)}}_{\text{updated state}} = m_j^{(k)}.$$

So the aggregation is a weighted sum over neighbors, and the "update function" here is just the identity (no nonlinearity or learned matrix).

Correct, if not a bit strange in defining excessive variables

(d) Interpretation with max aggregation instead of sum

Now let the update be

$$h_j^{(k+1)}[c] = \max_{i:A_{ij}=1} h_i^{(k)}[c],$$

where $[\cdot]_c$ is the c-th channel.

Layer 0: $h_i^{(0)} = e_i$.
So at layer 1,

$$h_j^{(1)}[i] = \begin{cases} 1, & \exists \text{ edge } i \to j \\ 0, & \text{otherwise.} \end{cases}$$

Inductively, max aggregation propagates "existence" instead of "count":

- $h_j^{(k)}[i] = 1$ if **there exists at least one** path of length k from i to j;
- $h_j^{(k)}[i] = 0$ otherwise.

So with max aggregation, node j's output encodes a **reachability indicator**: for each i, does there exist a length-k path from i to j?

Correct.

(e) Molecule-breaking problem

You want to predict **which bond breaks first**.

(e.i) Graph representation

- **Nodes:** atoms.
  Node features include:
  - element type (e.g., atomic number or one-hot over elements),
  - possibly degree/valence, charge, etc.
- **Edges:** chemical bonds between atoms.
  Edge features include:
  - bond length,
  - bond type (single, double, etc.).
- **Global state:** properties of the whole molecule, e.g.
  - molecule type (carbohydrate / protein / etc., one-hot),
  - possibly temperature/pressure if known.

Chemical bonds are inherently bidirectional, so the natural graph is **undirected**. In implementations that expect directed edges, you'd typically create two directed edges (one each way) with the same features.

Correcct, but includes extra features not present in the problem (bond type, valence, temp/press)

(e.ii) Using the last GNN layer to make the prediction

This is an **edge-level** prediction task (which bond breaks). A common approach:

1. Run the GNN for several layers to obtain final edge representations $e_{uv}^{(L)}$ for each bond (u, v).
2. Apply a small MLP or linear classifier to each edge embedding to produce a scalar score or a probability that that bond will break.
3. Choose the bond with the highest score (or apply a threshold plus a "no bond breaks" option).

If you want a single "which bond (or none)" prediction, you can also pool all edge scores into a global softmax.

Does state the solution of softmax, but offers additional options as well

(e.iii) Encoding node representations for the input

For each atom (node), construct a feature vector such as:

$$x_v = [\text{one-hot(element)},\ \text{(optional) degree/valence},\ \text{maybe other atomic properties}].$$

Examples:

- One-hot vector over possible elements (C, H, O, N, ...).
- Integer degree or one-hot over {1,2,3,4+}.
- Optional: formal charge, aromatic flag, etc.

Edge features would include bond length as a scalar (possibly normalized) and bond type as one-hot.

Once again, the solution is a bit verbose, and has extra features not present in the problem.
That being said, it is still correct.

(f) CNN ⇔ GNN analogies table

Here is one reasonable way to fill in the table.

CNN operation / task	**GNN analogue / interpretation**
Image classification	Graph-level prediction problem (already given).
Semantic segmentation / per-pixel classification	Node-level prediction problem (already in GNN column).
Color jitter data augmentation	Randomly perturb node features (add noise / jitter to node attributes).
Image flip data augmentation	Apply graph isomorphisms / permutations (e.g., reflect node coordinates or randomly permute node indices) as

CNN operation / task	GNN analogue / interpretation
	data augmentation.
Channel dropout	Dropout on node feature channels (randomly zero some dimensions of node embeddings).
Zero padding edges	Treat missing neighbors as implicit zero-features; equivalently, aggregation over existing neighbors is like zero padding non-edges.
ResNet skip connections	Residual / skip connections between GNN layers (add previous node embedding to the updated one).
Blurring an image	Graph smoothing / diffusion: repeatedly averaging node features with neighbors (Laplacian smoothing/message passing).
Image inpainting / predicting missing pixels	Predicting missing values of nodes (already in GNN column).

(There isn't always a one-to-one perfect match; the key is the *role* each operation plays.)

Almost all entries match the staff solution. However there are some that differ slightly. For example ChatGPT's zero-padding explanation does state to treat non-neighbors as "implicit zero-features", but it is not nearly as concise or coherent as the staff solution.

(g) Graph-level classification with missing node values

You can still use the full graph:

1. **Initialize** nodes with missing features using a special "unknown" embedding or zeros (possibly plus a binary mask feature indicating "missing").
2. Run the GNN normally:
   - Message passing will propagate information from neighbors with known features into these missing nodes.
3. Perform **graph-level pooling** (sum/mean/max over all node embeddings, or a learned global state) and feed that to a classifier to obtain the graph label.

Optionally, you can jointly train the GNN to **impute** missing node features (a node-level auxiliary task) while also training for graph-level classification.

Stated the heart of the solution (initializing with a special token), but did not specify about randomly removing nodes from training graphs. So I would score this as half credit.

(h) Graph Nets architecture (Figure 2)

We have node embeddings V_n, edge embeddings E_n, and (optionally) global state U_n, and update functions f_V, f_E, f_U.

(h.i) Doubling the number of nodes

Assume we only have node representations and no global state.

- **Number of learned weights:**
  The weights live in the shared update networks (e.g., the MLP implementing f_V). These are reused for *every* node. Doubling the number of nodes does **not** change the number of parameters; it stays the same.
- **Computation with same average degree:**
  If the average degree is constant d, then the number of edges $|E|$ scales like $d|V|$. Doubling $|V|$ roughly doubles $|E|$, and message passing is $O(|V| + |E|)$.
  ⇒ Total computation per layer roughly **doubles**.
- **Computation if the graph is fully connected:**
  For a fully connected graph on n nodes, $|E| \approx n^2$.
  Doubling nodes from n to $2n$ makes

$$|E| : n^2 \;\longrightarrow\; (2n)^2 = 4n^2,$$

  so the computation per layer becomes roughly **4× larger**. Correct.

(h.ii) Where are the learned weights?

- In the **update functions** f_V, f_E, and (if used) f_U, which are typically MLPs.
- Also in any learned readout / pooling MLP applied to node, edge, or global representations at the end.

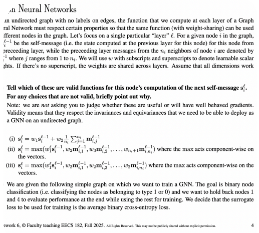
All nodes (and edges) share the same f_V (and f_E) parameters. Correct.

(h.iii) Incorporating directed edges

Several equivalent ways; one standard approach:

- Represent each edge as an ordered pair $(\text{sender}, \text{receiver})$ with its own edge embedding.
- In message passing, aggregate **incoming** messages at each receiver node from all edges whose receiver is that node.

For graphs that are originally undirected, you can model them as *two* directed edges (one in each direction) sharing the same features. For genuinely directed graphs, you just include the directed edges that actually exist, with asymmetric adjacency and possibly direction encoded in the edge features. Correct.



n Neural Networks

an undirected graph with no labels on edges, the function that we compute at each layer of a Graph ral Network must respect certain properties so that the same function (with weight-sharing) can be used fferent nodes in the graph. Let's focus on a single particular "layer" ℓ. For a given node i in the graph, $^{\ell-1}$ be the self-message (i.e. the state computed at the previous layer for this node) for this node from receeding layer, while the preceeding layer messages from the n_i neighbors of node i are denoted by ^{1} where j ranges from 1 to n_i. We will use w with subscripts and superscripts to denote learnable scalar hts. If there's no superscript, the weights are shared across layers. Assume that all dimensions work

Tell which of these are valid functions for this node's computation of the next self-message $\mathbf{s}_i^\ell$. For any choices that are not valid, briefly point out why.

Note: we are *not* asking you to judge whether these are useful or will have well behaved gradients. Validity means that they respect the invariances and equivariances that we need to be able to deploy as a GNN on an undirected graph.

(i) $\mathbf{s}_i^\ell = w_1\mathbf{s}_i^{\ell-1} + w_2\frac{1}{n_i}\sum_{j=1}^{n_i}\mathbf{m}_{i,j}^{\ell-1}$

(ii) $\mathbf{s}_i^\ell = \max(w_1^\ell\mathbf{s}_i^{\ell-1}, w_2\mathbf{m}_{i,1}^{\ell-1}, w_3\mathbf{m}_{i,2}^{\ell-1}, \dots, w_{n_i+1}\mathbf{m}_{i,n_i}^{\ell-1})$ where the max acts component-wise on the vectors.

(iii) $\mathbf{s}_i^\ell = \max(w_1^\ell\mathbf{s}_i^{\ell-1}, w_2\mathbf{m}_{i,1}^{\ell-1}, w_2\mathbf{m}_{i,2}^{\ell-1}, \dots, w_2\mathbf{m}_{i,n_i}^{\ell-1})$ where the max acts component-wise on the vectors.

We are given the following simple graph on which we want to train a GNN. The goal is binary node classification (i.e. classifying the nodes as belonging to type 1 or 0) and we want to hold back nodes 1 and 4 to evaluate performance at the end while using the rest for training. We decide that the surrogate loss to be used for training is the average binary cross-entropy loss.

ework 6, © Faculty teaching EECS 182, Fall 2025. All Rights Reserved. This may not be publicly shared without explicit permission. 4

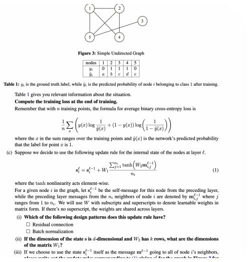


Figure 3: Simple Undirected Graph

nodes	1	2	3	4	5
y_i	0	1	1	1	0
$\hat{y}_i$	a	b	c	d	e

Table 1: y_i is the ground truth label, while $\hat{y}_i$ is the predicted probability of node i belonging to class 1 after training.

Table 1 gives you relevant information about the situation.

Compute the training loss at the end of training.

Remember that with n training points, the formula for average binary cross-entropy loss is

$$\frac{1}{n}\sum_x\left(y(x)\log\frac{1}{\hat{y}(x)} + (1-y(x))\log\left(\frac{1}{1-\hat{y}(x)}\right)\right)$$

where the x in the sum ranges over the training points and $\hat{y}(x)$ is the network's predicted probability that the label for point x is 1.

(c) Suppose we decide to use the following update rule for the internal state of the nodes at layer ℓ.

$$\mathbf{s}_i^\ell = \mathbf{s}_i^{\ell-1} + W_1\frac{\sum_{j=1}^{n_i}\tanh\left(W_2\mathbf{m}_{i,j}^{\ell-1}\right)}{n_i} \quad (1)$$

where the tanh nonlinearity acts element-wise.

For a given node i in the graph, let $\mathbf{s}_i^{\ell-1}$ be the self-message for this node from the preceding layer, while the preceding layer messages from the n_i neighbors of node i are denoted by $\mathbf{m}_{i,j}^{\ell-1}$ where j ranges from 1 to n_i. We will use W with subscripts and superscripts to denote learnable weights in matrix form. If there's no superscript, the weights are shared across layers.

(i) **Which of the following design patterns does this update rule have?**
- ☐ Residual connection
- ☐ Batch normalization

(ii) **If the dimension of the state s is d-dimensional and W_2 has k rows, what are the dimensions of the matrix W_1?**

(iii) If we choose to use the state $\mathbf{s}_i^{\ell-1}$ itself as the message $\mathbf{m}^{\ell-1}$ going to all of node i's neighbors,

Since the model was so accurate on the first question, I did not reprompt it and simply fed it the next question to see how well it did

Great work! Now do the same for the following problem:

Thought for 8m …

Interestingly this question is shorter and was submitted using less pictures but it took longer for ChatGPT to compute answers to the questions

I'll label the internal / "self" state of node i at layer ℓ as s_i^ℓ, and the message from a neighbor to node i at layer $\ell - 1$ as $m_{i,j}^{\ell-1}$.

(a) Validity of proposed update rules

Recall the requirements for a **GNN on an undirected graph with unlabeled edges**:

- The **same function with the same parameters** must be applied at every node (weight sharing).
- The function must be **permutation-invariant** w.r.t. the ordering of neighbors (because neighbors of an undirected node are an unordered set).

We check each option.

(a.i)

$$s_i^\ell = w_1 s_i^{\ell-1} + w_2 \frac{1}{n_i}\sum_{j=1}^{n_i} m_{i,j}^{\ell-1}$$

- Uses two scalars w_1, w_2 that are **shared** across all nodes.
- Aggregation over neighbors is a **mean** (sum divided by n_i), which is permutation-invariant in the neighbors.

✅ **Valid**: respects weight sharing and permutation invariance. Correct.

(a.ii)

$$s_i^{\ell} = \max\left(w_1^{\ell} s_i^{\ell-1},\ w_2 m_{i,1}^{\ell-1},\ w_3 m_{i,2}^{\ell-1},\ \ldots,\ w_{n_i+1} m_{i,n_i}^{\ell-1}\right)$$

(max taken componentwise)

Issues:

- Each neighbor j gets its **own** scalar w_{j+1}.
  The number of weights depends on n_i, the node's degree.
- Two nodes with the same neighborhood structure but a different *ordering* of neighbors will use messages multiplied by different w_js. So the result **depends on neighbor order**, not just the set of neighbors.

❌ **Not valid**: breaks both permutation invariance and global weight sharing.

Correct.

(a.iii)

$$s_i^{\ell} = \max\left(w_1^{\ell} s_i^{\ell-1},\ w_2 m_{i,1}^{\ell-1},\ w_2 m_{i,2}^{\ell-1},\ \ldots,\ w_2 m_{i,n_i}^{\ell-1}\right)$$

(max taken componentwise)

- The neighbors all use the **same** scalar w_2. So the number of parameters does not depend on degree.
- The max over the multiset $\{w_2 m_{i,j}^{\ell-1}\}_j$ is **permutation-invariant** w.r.t. neighbor order.

✅ **Valid**: respects permutation invariance and weight sharing.

Correct.

Summary:

- (i) valid
- (ii) **not** valid (degree- and order-dependent weights)
- (iii) valid

(b) Training loss at the end of training

We hold out **nodes 1 and 4** for evaluation, so the **training nodes** are:

- node 2, node 3, node 5.

From the table:

node i	1	2	3	4	5
y_i	0	1	1	1	0
$\hat{y}_i$	a	b	c	d	e

For average binary cross-entropy over n training points:

$$\mathrm{L} = \frac{1}{n}\sum_{x}\left(y(x)\log\frac{1}{\hat{y}(x)} + (1 - y(x))\log\frac{1}{1-\hat{y}(x)}\right)$$

Here $n = 3$ (nodes 2,3,5):

- Node 2: $y_2 = 1, \hat{y}_2 = b$
  Contribution: $\log\frac{1}{b}$.
- Node 3: $y_3 = 1, \hat{y}_3 = c$
  Contribution: $\log\frac{1}{c}$.
- Node 5: $y_5 = 0, \hat{y}_5 = e$
  Contribution: $\log\frac{1}{1-e}$.

So

$$\mathrm{L}_{\text{train}} = \frac{1}{3}\left(\log\frac{1}{b} + \log\frac{1}{c} + \log\frac{1}{1-e}\right)$$

Equivalently,

$$\mathrm{L}_{\text{train}} = -\frac{1}{3}\left(\log b + \log c + \log(1-e)\right).$$

Correct.

(c) Update rule analysis

We're given the node-state update (for layer ℓ):

$$s_i^{\ell} = s_i^{\ell-1} + W_1\frac{\sum_{j=1}^{n_i}\tanh\left(W_2 m_{i,j}^{\ell-1}\right)}{n_i} \tag{1}$$

with $\tanh$ applied elementwise.

(c.i) Design patterns

Options:

- **Residual connection?**
  Yes: $s_i^\ell = s_i^{\ell-1} + (\text{something})$ is a classic residual / skip connection.
- **Batch normalization?**
  No: there's no normalization over a batch or even over nodes; we just average over neighbors.

✅ It **has a residual connection.**

❌ It **does not use batch normalization.**

Correct.

(c.ii) Dimensions of W_1

Assume:

- Node state $s_i^\ell \in \mathbb{R}^d$.
- Message $m_{i,j}^{\ell-1}$ has same dimension (typical), so $m_{i,j}^{\ell-1} \in \mathbb{R}^d$.
- Matrix W_2 has k rows.

Then

- W_2 must be $k \times d$ so that
  $W_2 m_{i,j}^{\ell-1} \in \mathbb{R}^k$.
- The neighbor-averaged term inside $W_1(\cdot)$ is also in $\mathbb{R}^k$.
- To add the result to $s_i^{\ell-1} \in \mathbb{R}^d$, W_1 must map $\mathbb{R}^k \to \mathbb{R}^d$.

So

$$W_1 \in \mathbb{R}^{d\times k} \quad .$$

Correct.

(c.iii) Explicit updates for nodes $i = 2$ and $i = 3$

Assumption: the **message from node r to any neighbor** is just its previous-layer state:

$$m_r^{\ell-1} = s_r^{\ell-1}.$$

For node i, incoming messages from neighbors are then

$$m_{i,j}^{\ell-1} = s_{\text{neighbor}_j(i)}^{\ell-1}.$$

From Figure 3 (simple undirected graph), the neighbor sets are:

- $N(1) = \{2, 4, 5\}$
- $N(2) = \{1, 4, 5\}$
- $N(3) = \{5\}$
- $N(4) = \{1, 2\}$
- $N(5) = \{1, 2, 3\}$

So:

- $n_2 = |N(2)| = 3$
- $n_3 = |N(3)| = 1$

Node $i = 2$

Neighbors: $1, 4, 5$.

Using (1):

$$s_2^\ell = s_2^{\ell-1} + W_1 \frac{1}{n_2} \sum_{j \in N(2)} \tanh(W_2 s_j^{\ell-1})$$
$$= s_2^{\ell-1} + W_1 \frac{1}{3} (\tanh(W_2 s_1^{\ell-1}) + \tanh(W_2 s_4^{\ell-1}) + \tanh(W_2 s_5^{\ell-1})).$$

So

$$s_2^\ell = s_2^{\ell-1} + \frac{1}{3} W_1 [\tanh(W_2 s_1^{\ell-1}) + \tanh(W_2 s_4^{\ell-1}) + \tanh(W_2 s_5^{\ell-1})].$$

Correct.

Node $i = 3$

Neighbors: only 5, so $n_3 = 1$.

Using (1):

$$s_3^\ell = s_3^{\ell-1} + W_1 \frac{1}{1} \sum_{j \in N(3)} \tanh(W_2 s_j^{\ell-1})$$
$$= s_3^{\ell-1} + W_1 \tanh(W_2 s_5^{\ell-1}).$$

So

$$s_3^\ell = s_3^{\ell-1} + W_1 \tanh(W_2 s_5^{\ell-1}).$$ Correct.

(These expressions are fully expanded over neighbors and expressed in terms of earlier-layer states.)

Overall ChatGPT 5.1 Extended Thinking was extremely accurate on the HW problems and solved them quite efficiently, if sometimes overly verbose