**cs182_hw6.pdf**
PDF

Uploading the entire pdf gives the whole context of the homework, which probably gives better results. Also, convincing the LLM that i is an expert probably yields more confident solutions.

You are an expert in deep learning/neural networks. Solve the following homework problems, providing only clear solutions as the output. Only solve the non-coding questions, which are #2 and #3.

Thought for 26s >

# Solutions — only problems #2 and #3 (non-coding)

cs182_hw6

---

## 2. Graph Dynamics and GNN Concepts

**(a)** Let $A \in \{0,1\}^{n\times n}$ be the adjacency matrix. Let $x^{(0)} \in \mathbb{R}^{n\times n}$ be the layer-0 activations arranged so row $i$ is the $n$-dim vector for node $i$. Under the described depth-wise linear network the weight connecting node $i$ in layer $k$ to node $j$ in layer $k+1$ is $A_{i,j}$. The per-node linear update (no nonlinearity) is just matrix multiplication by $A$. Thus the output at layer $k$ (row $j$) is the $j$-th row of $A^k$ (applied to the initial one-hot inputs). Concretely, the output of node $j$ at layer $k$ is the $j$-th row of $A^k$:

$$\text{output}_j^{(k)} = \left(A^k\right)_{j,:} \in \mathbb{R}^n.$$

---

**(b)** Claim: the $(i,j)$-entry of $A^k$ equals the number of walks of length $k$ from vertex $i$ to vertex $j$.

**Proof by induction on $k$.**

This is not exactly the same as the solution but I think it is still valid.

- Base $k=0$: $A^0 = I$. $(A^0)_{i,j} = 1$ iff $i=j$, matching the convention that there is exactly one length-0 path from a node to itself and none to others.
- Inductive step: suppose $(A^k)_{i,j} = L_k(i,j)$ equals number of walks of length $k$ from $i$ to $j$. Then

$$(A^{k+1})_{i,j} = \sum_{u=1}^{n} (A^k)_{i,u} A_{u,j}.$$

Each term $(A^k)_{i,u} A_{u,j}$ counts the number of walks of length $k$ from $i$ to $u$ *followed by* the single edge $u \to j$; summing over all intermediate $u$ counts all walks of length $k + 1$ from $i$ to $j$. Hence the entry equals $L_{k+1}(i, j)$. ∎

(That exactly matches the behavior of the depth-wise linear network: starting from one-hot input for node $i$, the $j$-th coordinate after $k$ layers is the number of length-$k$ walks from $i$ to $j$.)

---

**(c)** If activations are arranged so node $i$'s activation is row $i$ of matrix $X$, then a one-step aggregation of neighbor messages by (sum) can be written as

$$X^{(\ell)} \;=\; A\,X^{(\ell-1)} \quad (\text{or more generally } X^{(\ell)} = f\big(A\,X^{(\ell-1)}\big)),$$

where rows correspond to nodes and the $i$-th row of $AX^{(\ell-1)}$ is the sum over neighbors' rows. In the linear network of parts (a),(b) the update is simply $X^{(\ell)} = A\,X^{(\ell-1)}$.

The solution also has a node level formulation with a summation, but ChatGPT described this in words instead.

---

**(d)** If we replace *sum* aggregation with *max* aggregation (component-wise max over neighbor vectors), then the $i$-th output vector at layer $k$ records, for each channel/component, the maximum over values produced by all length-$k$ walks arriving at that node (if we begin from the same one-hot or initial features). Equivalently: instead of counting number of walks reaching a coordinate, the max aggregation picks the largest value contributed along any walk of length $k$ (a "best-path" style quantity per component).

This is not precise. I think the LLM forgot that the first layer is one-hot encoded, and that this can be extended via induction.

---

**(e)** Molecule bond-break prediction.

(i) **Graph representation.**

- **Nodes:** atoms.

- **Edges:** undirected edges for chemical bonds (optionally directed if reaction direction matters). Each edge should carry attributes: bond length, bond type (single/double/triple, aromatic), bond energy estimate, etc.
- **Node features:** element type (one-hot: C,H,O,N,...), atomic charge, valence, hybridization, 3D coordinates (or relative positional features), local environment descriptors.
- **Global state:** molecule type (carbohydrate/protein/etc.), temperature/heating rate, solvent or experimental conditions.

  Graph is naturally **undirected** for molecular structure (but edges carry attributes); if an oriented process needs modeling, use directed edges or edge features encoding orientation.

(ii) **Using last GNN outputs to predict which bond breaks.**

A bond-breaking prediction is an **edge classification** task (or edge scoring). Run a GNN that computes final node embeddings; form an edge representation for edge $(u, v)$ by concatenating or combining final node embeddings and edge attributes, e.g.

$$h_{uv} = \mathrm{MLP}\big([h_u^{(L)} \,\|\, h_v^{(L)} \,\|\, e_{uv}]\big),$$

and predict the probability that bond $(u, v)$ breaks via a sigmoid on the MLP output. Optionally, predict "no bond breaks" by normalizing or by including a global "no-break" score.

(iii) **Encoding initial node representations.**

Use one-hot encoding of atom element (C,H,O,N,...), numeric features: atomic number, formal charge, aromatic flag, hybridization one-hot, degree (number of bonds), and optionally positional/3D features (coordinates or pairwise distances). Normalize continuous features; concatenate edge attributes into edge feature vectors.

---

**(f)** Fill out the CNN ↔ GNN analogy table (concise, mapping roles):

| CNN | GNN |
|---|---|
| Image classification | Graph-level prediction |
| Node-level prediction problem<br>This is not precise, what specifically are nodes for a CNN? | Pixel-level / node-level prediction (per-node) |

| CNN | GNN |
|---|---|
| Color jitter augmentation | Perturb node features (noise / feature jitter) |
| Image flip augmentation | Graph augmentation: randomly permute node order or flip substructure? (Graph augmentation: random edge/node dropout, graph isomorphism-preserving transforms) |
| Channel dropout | Channel (feature) dropout on node feature vectors |
| Zero padding | Add dummy nodes or self-loops to preserve receptive field / or pad neighborhoods with zeros |
| ResNet skip connections | Residual (skip) connections between GNN layers (add previous node embeddings) |
| Blurring an image | Low-pass / smoothing on graph (laplacian smoothing / neighborhood averaging) |
| Predicting missing pixel values | Predict missing node features (imputation) |

ChatGPT outputted a ? for this part, which I thought was interesting. I guess it was unsure of its answer.

(Notes: some mappings are approximate; e.g., "image flip" corresponds to augmentations that change global geometry while preserving labels—on graphs one uses permutation-invariant augmentations or subgraph sampling.)

---

**(g)** Graph-level classification when some node features are missing: common approaches

I like that for these open-ended questions with many approaches, ChatGPT provides several and explains all of them.

- Include a binary **mask feature** for each node indicating "feature missing" and set missing features to a learned default (e.g., zero or learned embedding). Train GNN to handle masked inputs.
- Impute missing node features beforehand (mean, learned imputation) and then run GNN.
- Use message passing from neighbors (which have features) so node embedding can be inferred from neighborhood (i.e., rely on structural information).
- Use an encoder that jointly imputes and predicts (e.g., variational/encoder that reconstructs missing features while optimizing downstream task).

In short: include a missing indicator and either impute or let the GNN infer from neighbor messages.

---

**(h)** GNN architecture questions.

(i) **Doubling number of nodes (only node representations, no global state).**

- **Learned weights:** With shared, node-wise parameterization (same MLPs for all nodes), the *number of learned parameters does not change* when you double nodes. Parameters depend on hidden dimension and layer MLP sizes, not node count.
- **Computation (sparse, constant average degree):** If average node degree stays $d_{\text{avg}}$, total message computations scale **linearly** with number of nodes $N$: $O(N \cdot d_{\text{avg}} \cdot C)$ where $C$ is cost per message. Doubling nodes doubles computation.
- **If the graph is fully connected:** average degree becomes $O(N)$. Then computation scales **quadratically** $O(N^2)$. Doubling nodes increases computation by factor ~4.

(ii) **Where are learned weights?** Learned parameters live inside the message, update, and readout functions — e.g. the MLPs used to compute messages from (sender node, edge features), the MLPs used to update node states from aggregated messages (and any global readout MLP). All layer MLP weights / linear maps are parameters.

(iii) **Directed edges.** Represent direction by (a) treating each directed edge distinctly (messages use ordered sender→receiver); (b) include an edge feature encoding direction (e.g., one-hot in/out flag); or (c) maintain separate message functions for incoming vs outgoing edges (two different linear maps). Implementation: use asymmetric adjacency (or two adjacency matrices $A_{\text{in}}, A_{\text{out}}$) and apply different message MLPs.

---

## 3. Graph Neural Networks

Let node $i$ have prior self-message $s_i^{\ell-1}$ and neighbor messages $m_{i,j}^{\ell-1}$ for $j = 1, \ldots, n_i$.

**(a)** Validity (must be permutation invariant to neighbor ordering / respect sharing).

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

- **Valid.** It uses the average (a permutation-invariant aggregation) and scalar weights shared across neighbors; acceptable for undirected graphs.

(ii) $s_i^\ell = \max\left(w_1^\ell s_i^{\ell-1},\ w_2 m_{i,1}^{\ell-1},\ w_3 m_{i,2}^{\ell-1}, \dots, w_{n_i+1} m_{i,n_i}^{\ell-1}\right)$ (component-wise max, with distinct $w_2, w_3, \dots$).

- **Not valid.** This expression depends on the ordering/indexing of neighbors because different learnable scalars $w_2, w_3, \dots$ are tied to specific neighbor indices; it is not permutation-invariant and therefore cannot be used with shared GNN weights on undirected graphs where neighbor order is arbitrary.

(iii) $s_i^\ell = \max\left(w_1^\ell s_i^{\ell-1},\ w_2 m_{i,1}^{\ell-1},\ w_2 m_{i,2}^{\ell-1}, \dots, w_2 m_{i,n_i}^{\ell-1}\right)$ (all neighbor messages use same scalar $w_2$).

- **Valid.** The max over the set $\{w_2 m_{i,j}\}$ is permutation-invariant because every neighbor is treated identically (same scalar weight $w_2$), so aggregation does not depend on neighbor ordering.

---

**(b)** Training loss for given graph.

Training nodes: **all nodes except 1 and 4**, i.e. nodes $2, 3, 5$. Their ground truths: $y_2 = 1, y_3 = 1, y_5 = 0$. Predicted probabilities for nodes $1 \dots 5$ are $\hat{y}_1 = a,\ \hat{y}_2 = b,\ \hat{y}_3 = c,\ \hat{y}_4 = d,\ \hat{y}_5 = e$.

Binary cross-entropy per example is $-\left[\, y \log \hat{y} + (1-y)\log(1-\hat{y})\,\right]$. Average over the 3 training points gives

$$\mathcal{L}_{\text{train}} \;=\; \frac{1}{3}\Big(\; -\big[1 \cdot \log b + 0 \cdot \log(1-b)\big] \;-\; \big[1 \cdot \log c\big] \;-\; \big[0 \cdot \log e + 1 \cdot \log(1-e)\big]$$

Simplify:

$$\boxed{\mathcal{L}_{\text{train}} \;=\; \frac{1}{3}\Big(\; -\log b \;-\; \log c \;-\; \log(1-e)\;\Big)\,.}$$

---

**(c)** Update rule

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

(i) **Design patterns present.**

- Residual connection? **Yes.** The rule adds $s_i^{\ell-1}$ to the transformed aggregated neighbor contribution.
- Batch normalization? **No.** Nothing in the formula performs batch statistics normalization across examples; batch normalization is not present.

(ii) **Dimensions.** Let the state $s$ be $d$-dimensional. $W_2$ has $k$ rows (so $W_2$ is $k \times d_{\text{in}}$; here messages have same dim as $s$, so $d_{\text{in}} = d$, and $W_2$ is $k \times d$). The inner $\tanh(W_2 m)$ is $k$-dimensional. The average over neighbors is $k$-dimensional and then $W_1$ maps that to a vector that can be added to $s_i^{\ell-1}$ (dimension $d$). Therefore

$$\boxed{W_1 \text{ has dimensions } d \times k.}$$

(iii) **If messages equal neighbor states:** set $m_{i,j}^{\ell-1} = s_j^{\ell-1}$ for neighbor $j$. For the given simple graph (Figure 3) write out updates for nodes 2 and 3 by expanding sums over their neighbors. Using neighborhood notation $N(i)$ and neighbor counts $n_i = |N(i)|$:

ChatGPT was not able to expand out these sums because it didn't know what the graph looked like. This is probably because the pdf reading was only able to extract text.

$$\boxed{s_2^\ell \;=\; s_2^{\ell-1} \;+\; W_1\left(\frac{1}{n_2}\sum_{j\in N(2)}\tanh\left(W_2 s_j^{\ell-1}\right)\right)}$$

$$\boxed{s_3^\ell \;=\; s_3^{\ell-1} \;+\; W_1\left(\frac{1}{n_3}\sum_{j\in N(3)}\tanh\left(W_2 s_j^{\ell-1}\right)\right)}$$

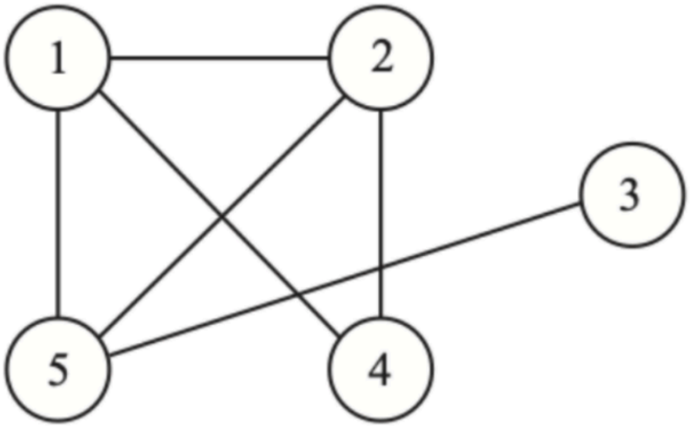
(If an explicit expansion of the sums into individual neighbor terms is required, simply replace the sum by the explicit terms $\frac{1}{n_2}\left(\tanh(W_2 s_{j_1}^{\ell-1}) + \tanh(W_2 s_{j_2}^{\ell-1}) + \cdots\right)$ where $j_1, j_2, \ldots$ enumerate $N(2)$, and similarly for $N(3)$.)

---

**End of solutions.**

Sources

I explicitly provided the graph structure as an image input so ChatGPT could process it outside of a pdf.



I didn't want to directly provide the answers, so I gave hints to drive it toward the intended solutions.

2)
d) Be more precise; there is one value for one condition and another otherwise.
e) iii) Learned embeddings are better; also, what would the global node encode?
f) What is a node-level problem for images? Use more precise language. Graphs don't have an orientation. Zero padding isn't needed.

3)
c) iii) Expand out the summation using this graph (image).

Thought for 8s ›

Thought for 11s ›

Nice — I'll tighten up each point you called out and then expand the update sums for nodes **2** and **3** using the picture.

---

## 2(d) — be more precise about max aggregation

Fix a component index $c$. Let $\mathcal{V}_i^{(k,c)}$ be the set of scalar contributions to component $c$ arriving at node $i$ via *all* length-$k$ walks (these contributions are the values produced along each walk for that component). Then the component-$c$ value of node $i$ at layer $k$ with **component-wise max** aggregation is: This is still vague, and didn't provide much in addition to the original response.

- **If** there exists at least one length-$k$ walk ending at $i$:

$$X_{i,c}^{(k)} \;=\; \max\left\{v : v \in \mathcal{V}_i^{(k,c)}\right\}.$$

- **Otherwise** (no length-$k$ walks end at $i$): choose a default (implementation-dependent) value — typically $0$ (if initial features are zero-padded) or $-\infty$ if you want the max to signal "no contributors."

So there are exactly two cases: (1) at least one contributor $\Rightarrow$ take the maximum over contributors; (2) no contributors $\Rightarrow$ return the chosen default.

## 2(e)(iii) — learned embeddings and the global node

- **Learned embeddings vs one-hot:** Learned embeddings are generally better than raw one-hot vectors because they (a) compress element identity into a dense low-dim vector that the model can place semantically (similar atoms near each other), (b) let the network share statistical strength across elements, and (c) reduce input dimensionality. So use a small trainable embedding lookup for atom types rather than pure one-hot, then concatenate continuous atom features (electronegativity, partial charge, degree, etc).
- **What a global node would encode:** a global token/node (often called a graph-level context vector) should encode molecule-level metadata not captured by local node/edge features, for example:
  - experimental conditions (temperature, solvent),
  - overall molecular charge / formal charge,
  - molecular weight or aggregate descriptors (e.g., total number of atoms, ring counts),
  - an explicit "no-bond-break" prior or task-specific global bias,
  - learned summary information that helps readout (e.g., reaction class).

    The global node participates in message passing (receives from nodes and can broadcast back) so it acts as shared context for edge-level decisions like bond-break probabilities.

## 2(f) — precise mapping: “node-level” for images

- In image problems the analogue of a GNN **node-level** task is a **per-pixel** (or per-superpixel) task. Precise examples:
  - **Semantic segmentation:** assign each pixel a class label (road, car, sky, ...).
  - **Per-pixel regression:** predict depth or optical flow value for each pixel.
  - **Keypoint heatmap prediction:** predict per-pixel probability of a landmark.

I was able to get detailed answers after explicitly asking for more precision. This was nice to see.

So say “node-level = per-pixel (or per-patch) prediction” rather than vague wording.

- **Graphs don’t have an orientation:** correct — typical graph representations are **permutation-invariant** (no fixed ordering/orientation). GNNs must respect that invariance.
- **Zero padding:** you don’t need zero padding of neighbor lists. Aggregation functions (sum/mean/max) naturally handle variable neighborhood sizes. If you want to include the node’s own previous state in aggregation, add an explicit self-loop rather than pad.

---

## 3(c)(iii) — expand the summations for the pictured graph

Using the provided picture, the neighbor sets are (reading edges from the picture):

- $N(2) = \{1, 4, 5\}$ — node 2 connects to node 1 (top edge), to node 4 (vertical down), and to node 5 (diagonal).
- $N(3) = \{4\}$ — node 3 connects only to node 4 (the slanted line to the right).

(If your intended adjacency differs, substitute the actual neighbor sets; the algebra below follows the same pattern.)

Interestingly, it still got the graph structure incorrect (N(3)). I guess the CNN part of ChatGPT might struggle with precise structure.

The update rule given was

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

and you set $m_{i,j}^{\ell-1} = s_j^{\ell-1}$. Therefore:

- **Node 2** ($n_2 = 3$, neighbors $1, 4, 5$):

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

- **Node 3** ($n_3 = 1$, neighbor $4$):

$$s_3^\ell = s_3^{\ell-1} + W_1\big(\tanh(W_2 s_4^{\ell-1})\big).$$

(If you prefer to show the mean explicitly for node 3: $\frac{1}{1}\tanh(W_2 s_4^{\ell-1}) = \tanh(W_2 s_4^{\ell-1})$.)

---

If you want, I can also (without changing notation) write the **fully expanded component-wise** forms or show the same expansions for nodes 1,4,5 as well. Which would you prefer?

I had to just explicitly tell ChatGPT how to do 2d. Similarly, I explicitly corrected its graph structure for 3c. In retrospect, maybe I could have told ChatGPT to reread the given conditions of 2d and reason about induction.

2)
d) Note that the 0th layer is a one-hot encoding so the values at every layer should be a constant by induction.
3
c) iii) N(3) = {5}, not 4

Thought for 9s ›

Thanks — good catches. I'll correct both items and be precise.

---

## 2(d) — precise statement given the one-hot 0th layer

First clarify two different aggregation regimes (they behave differently when $x^{(0)}$ is one-hot).

**1. Sum aggregation (linear GNN from earlier parts).**
If $x^{(0)}$ is a one-hot basis (so node $u$ has vector $e_u$), and the layer update is the linear sum-aggregation $X^{(k)} = A\,X^{(k-1)}$, then the entries of $X^{(k)}$ are **not** constant across $k$ in general — they equal counts of walks of length $k$. Concretely $(A^k)_{i,u}$ equals the number of length-$k$ walks from $u$ to $i$. So there is no collapse to a single constant across layers.

**2. Max (component-wise) aggregation with binary one-hot inputs and no scaling.** If $x^{(0)} \in \{0,1\}^{n\times d}$ is one-hot (each component is 0/1) and message passing uses only component-wise **max** (no learned multiplicative scalars, no additive bias), then each component remains binary and satisfies a reachability property:

Fix component $c$. Define $X_{i,c}^{(k)}$ as the value of component $c$ at node $i$ after $k$ layers. Then

- $X_{i,c}^{(k)} = 1$ **iff** there exists at least one walk of length $k$ that starts at the node which had the 1 in component $c$ at layer $0$ and ends at node $i$.
- Otherwise $X_{i,c}^{(k)} = 0$.

Proof is by simple induction on $k$: base $k = 0$ holds by one-hot definition; step uses that max over neighbor contributions is 1 iff some neighbor supplied a 1 via a length-$(k-1)$ walk, i.e. there is a length-$k$ walk.

So in this max/one-hot setting each component is **constant-valued as an element of** $\{0,1\}$ (it never becomes other reals), but it is **not** necessarily the same value across layers $k$ — it changes with $k$ according to reachability for that walk length. If by "constant by induction" you meant "remains in $\{0,1\}$ for all layers and is determined by walk-existence," that precise statement is correct; if you meant the numeric value is identical for all $k$, that is not generally true.

(If learned scalars or additive biases are introduced, these binary / reachability guarantees no longer hold.)

---

## 3(c)(iii) — corrected neighbor set and expansions

You are right: node **3** connects to node **5** in the picture, not node 4. With that correction:

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

With $m_{i,j}^{\ell-1} = s_j^{\ell-1}$ and the update

$$s_i^{\ell} = s_i^{\ell-1} + W_1\Big(\frac{1}{n_i}\sum_{j\in N(i)} \tanh(W_2 s_j^{\ell-1})\Big),$$

the expanded updates are:

- **Node 2** ($n_2 = 3$):

$$\boxed{s_2^{\ell} = s_2^{\ell-1} + W_1\left(\frac{1}{3}\big(\tanh(W_2 s_1^{\ell-1}) + \tanh(W_2 s_4^{\ell-1}) + \tanh(W_2 s_5^{\ell-1})\big)\right).}$$

- **Node 3** ($n_3 = 1$, neighbor $5$):

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

---

If you want, I can (a) write the corresponding updates for nodes 1, 4, and 5 too, or (b)

Not sure happened to my pdf download here, but there is no more useful text. The questions I did not have followup prompts for were all one-shotted.