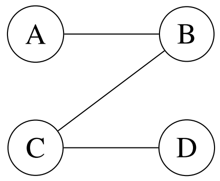
EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Discussion 6

1. Graph Neural Network Forward Pass

Consider the following undirected graph G:



In this problem, we are going to work with an undirected graph without edge weights. We are imposing these limitations to make the problem simpler and to let us focus on the core ideas behind the forward pass. In practice, edges can be directed and have weights.

When a GNN layer is applied to a graph, it produces a new graph with the same topology as the original graph but with (potentially) different values in the nodes and the edges. After passing a graph through a number of these GNN layers, we can use the graph embedding for a variety of downstream tasks. In this problem, we will walk through a simplified forward pass of graph neural networks to help build concrete intuition for how GNNs operate (and so we will not be thinking about the downstream tasks). We will gradually add layers of complexity to the forward pass to allow GNNs to become more expressive.

To begin with, let us assign vectors v_A, v_B, v_C, v_D to nodes A, B, C, D respectively. Let:

$$v_A = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, v_B = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, v_C = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, v_D = \begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

We will define our update function for our nodes as follows:

$$f_v(v_i) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} v_i$$

In practice, we could have different update functions at different layers of our network (and more generally, these update functions are learnable). For the sake of this problem, we will reuse the same update function at every layer of the network.

For example, to produce the node value at timestep $t+1$, we must apply the update rule to the node from timestep t, and so we have:

$$v_i^{(t+1)} = f_v(v_i^{(t)})$$

Thus, to compute $v_A^{(1)}$ and $v_A^{(2)}$, we have:

$$v_A^{(1)} = f_v(v_A^{(0)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 13 \\ 6 \end{bmatrix}$$

$$v_A^{(2)} = f_v(v_A^{(1)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 13 \\ 6 \end{bmatrix} = \begin{bmatrix} 69 \\ 58 \end{bmatrix}$$

In general, to produce the graph at timestep $t+1$, we will apply the update rules to each node in our graph from timestep t. Suppose that at timestep 0, the graph G is as above. Let us denote the state of G at timestep t by $G^{(t)}$.

(a) Using the updates rule above, **compute $\mathbf{G^{(1)}}$ and $\mathbf{G^{(2)}}$**.

Solution:

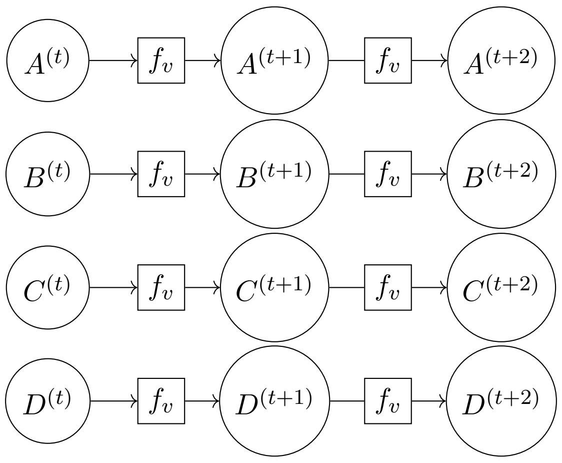
$$f_v(v_A^{(0)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 13 \\ 6 \end{bmatrix}, \qquad f_v(v_A^{(1)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 13 \\ 6 \end{bmatrix} = \begin{bmatrix} 69 \\ 58 \end{bmatrix}$$

$$f_v(v_B^{(0)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 7 \\ -2 \end{bmatrix}, \qquad f_v(v_B^{(1)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ -2 \end{bmatrix} = \begin{bmatrix} 11 \\ 26 \end{bmatrix}$$

$$f_v(v_C^{(0)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 16 \\ 10 \end{bmatrix}, \qquad f_v(v_C^{(1)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 16 \\ 10 \end{bmatrix} = \begin{bmatrix} 98 \\ 74 \end{bmatrix}$$

$$f_v(v_D^{(0)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} -3 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{-10} \end{bmatrix}, \qquad f_v(v_D^{(1)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -10 \end{bmatrix} = \begin{bmatrix} \text{-47} \\ -6 \end{bmatrix}$$

(b) We can visualize two iterations of our current update rule with a diagram such as the following:



$$\begin{tikzcd}
A^{(t)} \arrow[r] & \boxed{f_v} \arrow[r] & A^{(t+1)} \arrow[r] & \boxed{f_v} \arrow[r] & A^{(t+2)} \\
B^{(t)} \arrow[r] & \boxed{f_v} \arrow[r] & B^{(t+1)} \arrow[r] & \boxed{f_v} \arrow[r] & B^{(t+2)} \\
C^{(t)} \arrow[r] & \boxed{f_v} \arrow[r] & C^{(t+1)} \arrow[r] & \boxed{f_v} \arrow[r] & C^{(t+2)} \\
D^{(t)} \arrow[r] & \boxed{f_v} \arrow[r] & D^{(t+1)} \arrow[r] & \boxed{f_v} \arrow[r] & D^{(t+2)}
\end{tikzcd}$$

From this diagram, it is clear to see that our GNN is not leveraging the topology of our graph in its forward pass. The way we overcome this is through message passing. In practice, we can apply message passing to both nodes and edges. In this problem, we will only consider applying message passing to nodes to simplify things. Let us consider the new update rule for nodes:

$$v_i^{(t+1)} = f_v(v_i^{(t)}) + \sum_{v_j \in N(v_i)} f_v(v_j^{(t)})$$

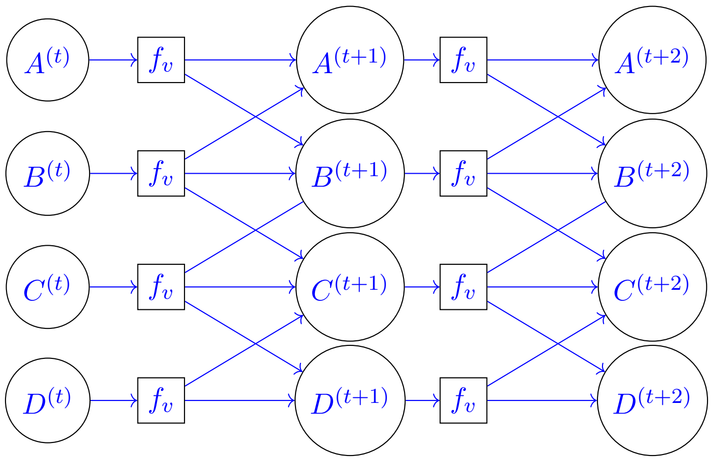
Where $N(v_i)$ is the set of neighbors of node v_i.
Find $\mathbf{G^{(1)}}$ under this new update rule.

Solution: (Only v_A is shown, but similar formulas can be applied to the other nodes. Depending on the node's number of neighbors, the sum will have 2 or 3 terms (# neighbors + 1)).

$$v_A^{(1)} = f_v(v_A^{(0)}) + f_v(v_B^{(0)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 20 \\ 4 \end{bmatrix},$$

(c) **Draw a diagram like the one in part b reflecting two iterations of our new update rule.**

Solution:

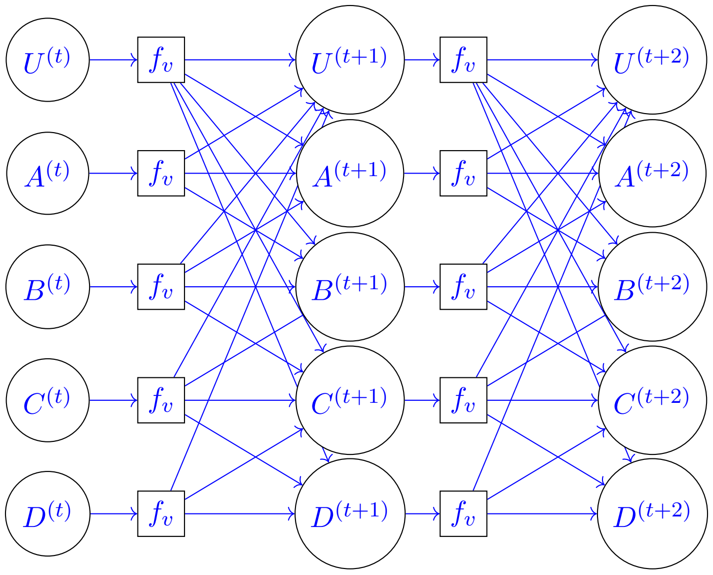


(d) Suppose the shortest path between nodes u and v in a graph traverses K edges. **How many iterations of updates must we do for information from node u to reach node v**? *(Hint: Consider the network diagram you made in part c)*

Solution: From looking at the network diagram, we can see that it will take K iterations for these nodes to communicate with eachother.

(e) If two nodes are connected, they will eventually be able to communicate with each other given enough layers. However, if we want to be able to apply the same network architecture to an arbitrary graph, then we have no guarantee that two connected nodes will be able to communicate with each other with our architecture (such as if we have L layers but we are now processing a graph which has two nodes separated by $L + 1$ edges). One approach to fixing this issue is to add a dummy node to our graph that is maximally connected to all of the original nodes in the graph. We can think of this node as representing the global state of the graph. **Draw a diagram like the one from part c reflecting the addition of this new dummy node. How does this solve our problem?**

Solution:



This solves our problem by allowing an arbitrary pair of nodes to communicate with each other in at most 2 layers.

(f) Although we are thinking about the global state as a "node", in practice we often think of it as a separate entity from the graph, and we allow it to have dimension different from that of our nodes. Correspondingly, we allow our GNN to have a separate update function for the global state, often written at f_U. Since we now have two separate update functions, our diagram from the last part will look different. To make things simpler, let us assume that the nodes have the same dimension as the global state ($U \in \mathbb{R}^2$). **Write an update rule for the global state.**

Solution: The actual update rule is a design choice. Some possibilities are:

$$U^{(t+1)} = f_U(U^{(t)}) + \sum_{v_i \in G} f_v(v_i^{(t)})$$

$$U^{(t+1)} = f_U(U^{(t)}) + f_v\left(\sum_{v_i \in G} v_i^{(t)}\right)$$

$$U^{(t+1)} = f_U(U^{(t)}) + \sum_{v_i \in G} v_i^{(t)}$$

$$U^{(t+1)} = f_U\left(U^{(t)} + \sum_{v_i \in G} v_i^{(t)}\right)$$

2. Understand GNN Through Graph Theory

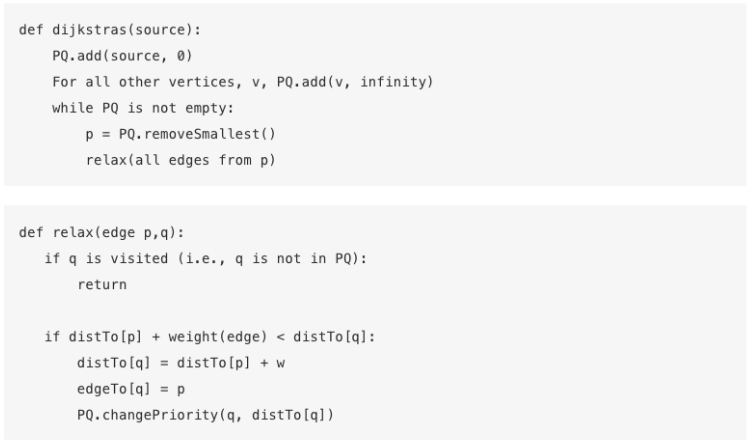
You've seen how Graph Neural Nets (GNN) naturally generalizes the mechanism of ConvNets. While it is indeed helpful to understand from ConvNet's perspective, the advent of GNN is actually to solve problems in graph theory. This question is a thought exercise for you to learn how the concepts in GNN could emerge from just solving traditional graph problems.

Consider finding a **shortest path algorithm** without knowing the underlying function. This kind of problem, to predict a function of a graph, is common in multiple applications such as in social network studies and molecular biology. For each data sample, we are given

- a hashtable $\mathbf{G}$ where each entry is (*source index*, *target index*, *edge length*)
- a label of $\mathbb{R}^N$ vector $\mathbf{V}$ (assume that all *edge lengths* are positive)

We decided to frame this problem as a graph neural net given some background knowledge. Our job is essentially to learn a **mapping function** $f_\theta(\mathbf{G}) = \mathbf{V}$, where the underlying function f is **Dijkstra** algorithm starting from one given source vertex, which we do not know when starting this problem.

Here is the pseudocode for Dijkstra:



```
def dijkstras(source):
    PQ.add(source, 0)
    For all other vertices, v, PQ.add(v, infinity)
    while PQ is not empty:
        p = PQ.removeSmallest()
        relax(all edges from p)
```

```
def relax(edge p,q):
   if q is visited (i.e., q is not in PQ):
       return

   if distTo[p] + weight(edge) < distTo[q]:
       distTo[q] = distTo[p] + w
       edgeTo[q] = p
       PQ.changePriority(q, distTo[q])
```

Figure 1: Pesudocode from `https://joshhug.gitbooks.io/hug61b`

(a) **Is this a node-level, edge-level, or graph-level prediction?** (*Hint: There's no one correct answer. Think about why each option may or may not make sense.*)

Solution:

Since we are predicting an overall property of a graph, the straightforward idea would be framing this as a graph-level prediction problem. Node-level prediction, however, can also be helpful since well-learnt node embeddings could help the **relax** method to be more accurate, given that **relax** can be understood as updating each node's embedding according to neighbor distances.

(b) In GNN, a key property is message passing. **Which procedure in the pseudocode above resembles the idea of message passing? How many iterations (forward pass) does it at most take to propagate the information through the graph?**

Solution:

The **relax** method acts like message passing in GNN since it updates the distances according to the rule of `distTo[q] = distTo[p] + w`, edges by edges, E times where E denotes the number of edges. Therefore it will take at most E iterations to propagate the information through the graph.

(c) In **relax** function we need a $\min$ function, which is not differentiable. Can you think of a way to approximate this with non-linearities in neural net?

Solution:

There could be multiple answers. One way to create $\min$ function using differentiable non-linearities is with softmax and log, namely:

$$f(x) = -\frac{1}{\lambda}\log\sum_i e^{-\lambda x_i} \quad \text{where } \lambda > 0$$

where λ controls the smoothness of softmax. The larger λ is, the closer $f(x)$ approximates the minimum. To create this differentiable min function, we need 2 non-linear layers.

3. Graph Neural Network Conceptual Questions

Solution:

This question, except for part B., is also covered in dis 07 since most sections did not have time for the remaining parts. The solution for part A, C, D, E will be posted after discussion 7.

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:

A. You are studying how organic molecules break down when heated. For each molecule, you know the element and weight of each atom, which other atoms its connected to, the length of the bond the atoms, 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.

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

   (b) How would you use the outputs of the last GNN layer to make the prediction?

   (c) How would you encode the node representation?

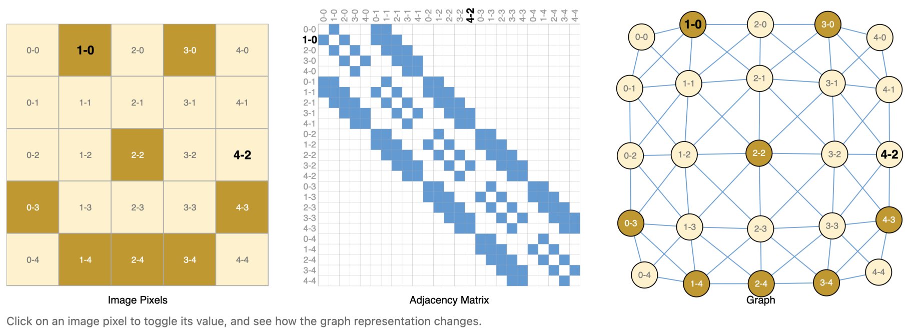


Image Pixels
Adjacency Matrix
Graph
Click on an image pixel to toggle its value, and see how the graph representation changes.

Figure 2: Images as Graphs

B. There are analogs of many ConvNet operations which can be done with GNNs. As Figure 2 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
Solution: Semantic segmentation (classifying what object is present at each pixel)	Node-level prediction problem
Color jitter data augmentation (adjusting the color or brightness of an image)	**Solution:** Jittering node values
Image flip data augmentation	**Solution:** A flip preserves graph values and connectivity, but changes its spatial orientation. Graphs don't have an orientation, so they don't need an analog. More generally, image flips augment the data by exploiting invariances in the task (i.e. an image's class shouldn't change if you flip it), and there might be similar invariances in graph problems.
Dropout	**Solution:** Zero some nodes (or edges) at one layer of the network
Zero padding edges	**Solution:** Zero padding addresses the fact that conv nets require that every pixel has the same number of neighbors. Graphs don't need an equivalent since they already handle variable numbers of neighbors
ResNet skip connections	**Solution:** Add a skip connection to the update - i.e. $v_i^{t+1} = v_i^t + UPDATE(v_i^t)$
Blurring an image	**Solution:** Averaging node values with neighbors
Solution: Image inpainting (filling in a missing section of an image)	Predicting missing values of nodes
Solution: Only using kernels with the same value for each index except the center (explanation: in GNNs, you typically run the same update function on each neighbor (or often first sum or average the neighbors, then run the update fn on the result). This means all neighbors are treated the same. CNN kernels, in contrast, have different values for at different spatial positions, so a left-side neighbor and a right-side neighbor don't get the same update. The CNN equivalent to edge-order invariance is to use the same value for each index in the kernel. The center pixel can still be different, since this corresponds to the current node value, which updated separately from the neighboring nodes.	Edge-order invariance - i.e. neighboring nodes/edges are a set with no ordering

C. 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?

D. Consider the graph neural net architecture shown in Figure 3. 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 pooling 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.

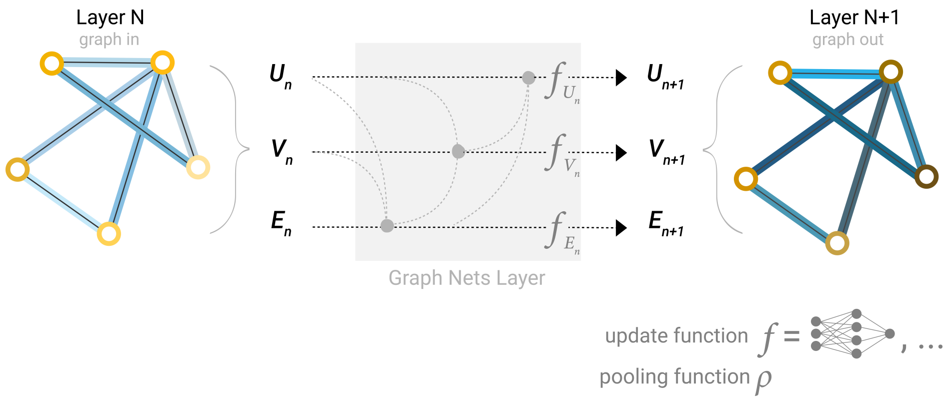


Figure 3: GNN architecture

(a) If we double the number of nodes in a graph which only has node representations, how does this change the number of learned weights in the graph? How does it change the amount of computation used for this graph if the average node degree remains the same? What if the graph if fully connected? (Assume you are not using a global state representation).

(b) Where in this network are learned weights incorporated?

(c) The diagram provided shows undirected edges. How would you incorporate directed edges?

(d) The differences between an MLP and a RNN include (a) weights are shared between timesteps and (b) data is fed in one timestep at a time. How would you make an MLP-like GNN? What about an RNN-like GNN?

E. Let's say you run a large social network and are trying to predict whether individuals in the network like a particular ad. Each individual is represented as a node in the graph, and we are doing a node-level prediction problem. You have labels for around half of the people in the network and are trying to predict the other half. Unlike a traditional ML problem, where there there are many independent training points, here we have a single social network. How would you do prediction on this graph?

F. (Optional) Play around with different GNN design choices in the GNN playground at https://distill.pub/2021/gnn-intro/#gnn-playground. Which design choices lead to the best AUC?