EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Midterm Review: CNNs

1. CNN Design decisions

Consider a CNN for classification consisting of the layers shown below. "..." indicates appropriate arguments that have been passed in.

```
nn.Conv2d(...),
nn.ReLU(),
nn. MaxPool2d(...),
nn. Conv2d(...),
nn. ReLU(),
nn. MaxPool2d(...),
nn. Flatten(),
nn. Linear(...),
nn. Softmax(...)
```

In the questions below, explain how you would modify this design to achieve particular objectives.

(a) You would like to use fewer parameters in the fully connected layer at the end of the network. The parameters in the final fully connected layer are equal to (num classes) * (flattened output from the last conv block). We can reduce this flattened output by either reducing the number of channels in the last conv layer or by reducing the spatial dimension of the feature map. You can reduce the spatial dimension by increasing the stride in the conv/pooling layers, or by adding additional conv/pooling layers with stride > 1. Removing padding from the conv/pooling layers also slightly decreases the spatial dimension.

(b) You would like the model to be able to handle images of different sizes. Replace the Flatten and fully connected layers with a global pooling layer.

(c) You would like to increase the receptive field of each pixel in the feature map output by the second conv layer. Increase the stride of the first conv or max pool layer. Increasing the kernel size of either conv layer or the first max pool layer has a small effect too.

(d) You would like every conv layer to leave the height and width dimensions of its input unchanged. Use a stride of 1 and add padding to each conv layer. In Pytorch, this is 'SAME' padding. Manually, you would pad by $(K-1)/2$ on each side, where K is the kernel height/width.

(e) You would like to add 50 more layers to this network without suffering from vanishing gradients. Add Resblocks (blocks with residual connections) to the network rather than simple conv layers.

2. 3D Convolutions

We've seen CNNs in 1D and 2D settings. Now, let's consider a 3D setting: you are building a classifier to detect the activity shown in short video clips.

(a) One approach is to build a CNN which uses 3D convolutions. What is the size of the parameters in each convolutional layer? Write your answer in terms of F (number of filters), S (stride), C (number

of input channels), H (input height), W (input width), T (input time length), K (kernel size). The weight will be (F, C, K, K, K) (Note: in some implementations, the C dimension comes last). The bias is F.

(b) Another approach is to process frames individually, then use an RNN to aggregate information over time. How would you combine a convolutional encoder with an RNN? At each timestep, encode the current frame with the same convolutional encoder. This encoder should include a flatten or global pooling layer which collapses the input into a single vector. This vector is then passed as the input at that timestep to the RNN. Make the final classification using a linear layer on top of the finall hidden state of the RNN.

EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Discussion 5

1. **Where Could We Use Graph Neural Network** In this problem, we aim to find out where using a graph is an appropriate way to solve the problems highlighted below. Answer with True/ False on where you think a GNN should be applied.

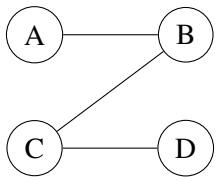
   A. We want to do image segmentation (i.e., predict what class of object is shown in each pixel.)

   B. We want to predict what is the likely play of a football team. Your data consists of the coordinate position of each player at each timestep.

   C. We want to find influential papers by identifying citation patterns between different papers. Your data consists of a list of which papers cite which other papers.

2. 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:

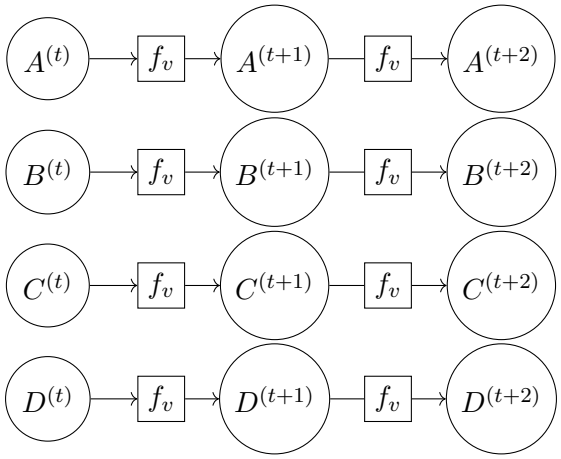
$$v_A^{(1)} = f_v(v_A^{(0)}) = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 13 \\ 6 \end{bmatrix}$$

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

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 $\mathbf{G^{(2)}}$.**

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



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.

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

(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 $\mathbf{u}$ to reach node $\mathbf{v}$?** *(Hint: Consider the network diagram you made in part c)*

(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?**

(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.**

(g) **Draw a diagram like the one in part c reflecting two iterations of our network using f_v and the update rule you wrote in the previous part.**

3. **Graph Neural Network Conceptual Questions** 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?

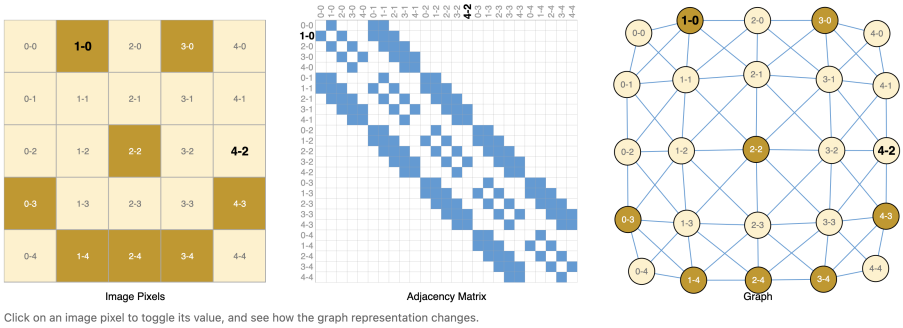


Figure 1: Images as Graphs

   B. 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	
Dropout	
Zero padding edges	
ResNet skip connections	
Blurring an image	
	Predicting missing values of 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 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 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.

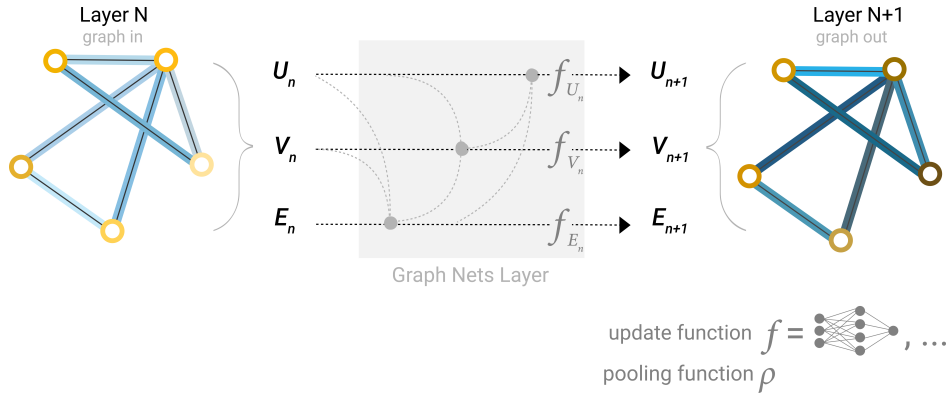


Figure 2: GNN architecture

(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?

Contributors:

- Matthew Lacayo.
- Olivia Watkins.
- Jerome Quenum.
- Anant Sahai.
- Anrui Gu.

