CS 282 Deep Neural Networks **Fall 2022**

Lecture 11: GNN

Lecturer: Anant Sahai *Scribe: Jiashu Liang, Anirudh Rengarajan*

11.1 Key ideas of the Convolutional Neural Networks (CNNs)

Images have a structure with patterns that we want to learn and we need to create inductive biases to help us learn the structure. The key ideas including:

1. Convolutions with weight-sharing AND having an "image" at each layer: build the local convolutions and then use hierarchical depth to see the entire image
2. Residual Connections to fight dying gradients: every layer has an effect when parameters change on what happens at the end
3. Normalization to "adaptive speed bump" exploding gradients: brings down growing activation values
4. Pooling to downsample: allowing distant information to more quickly get used
   - Max Pooling $\left[\text{MaxPool}(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}) = \max(a,\, b,\, c,\, d,\, e,\, f,\, g,\, h,\, i) \right]$: "routes" gradients to specific parts of images
5. Data Augmentation, Dropout (including stochastic depth regularization), Label Smoothing

Food for thought: What is the conceptual similarity and difference between Residual Connections and Pooling?

Residual Connections and Pooling can both allow the gradients to depend on more things. You can also realize the effect of Residual Connections by Pooling (i.e., sum pooling all the layers before each layer). However, this will result in quadratic intrinsic growth of the gradients.

11.2 Graph Neural Networks (GNNs) as the "generalization" of CNNs

11.2.1 Basic GNN model

Instead of a 2D grid for an image, we have a graph with information in nodes. First, we can have a Simple Assumption as follows.

Simplifying Assumption: We can define a Single Graph topology. A graph can be defined by a tuple (V, E) where V is the set of all nodes and E are the set of all edges that connects the nodes in V. For GNN, we attempt to do a Graph-level classification task based on information in nodes and connections between said nodes.

An example of a GNN in a Single Graph topology can be shown in Figure 11.1. The nodes are connected with the black lines in the same layer. Then each node in the next layer is connected to its neighbors in the previous layer (dotted red lines) and itself in the previous layer (dotted black lines).

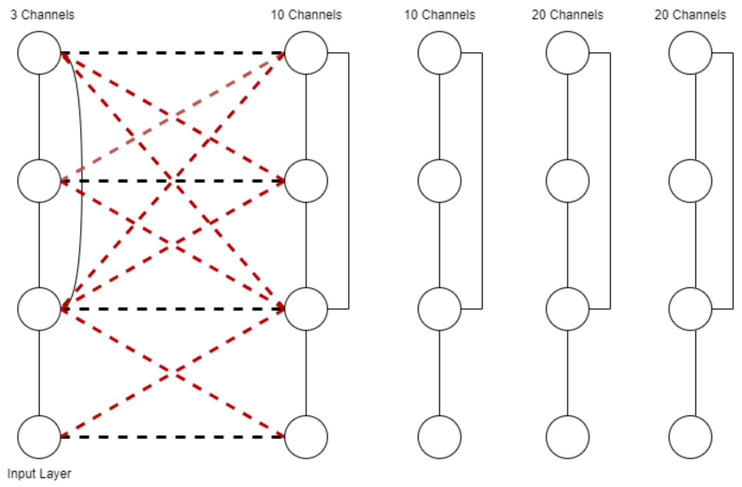
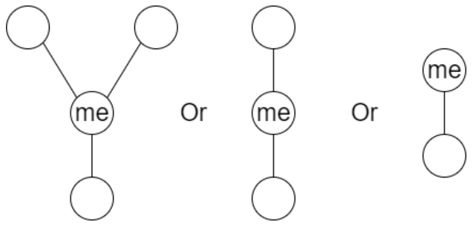


Figure 11.1: Example GNN showing connections inside a layer and between layers.

11.2.2 Differences between CNNs and GNNs from a Neighbor Perspective

What is different about "me" in these instances:

CNN: $\begin{bmatrix} a & b & c \\ d & me & f \\ g & h & i \end{bmatrix}$ vs. GNN:



- We might have different numbers of neighbors to "me". In contrast, one pixel of an image usually has eight neighbor pixels in CNNs.
- We don't have separate names for neighbors of "me". This means that the neighbors do not have a particular order in GNNs. In contrast, the eight neighbor pixels of one pixel have their particular position in CNNs, like the pixel "b" is on the top of "me" and the pixel "f" is on the right.

11.3 Extension of key ideas of CNNs to GNNs

11.3.1 Weight sharing

The most important idea behind CNN is the convolution with weight-sharing. What kind of function can we have in place of convolution, respecting the properties of a graph?

1. First, this function should have some learnable parameters associated with it (like w_1 in Eqn 11.1a).
2. Second, it should take two arguments, the node itself and its neighbor nodes, because these are all the information we know related to this node.
3. Then, a method is needed to combine the information of these neighbor nodes regardless of the ordering or cardinality of neighbors. This method is not learnable and we need to choose from possible choices, including Sum (used in Eqn 11.1a), Maximum, Minimum, Product, Softmax, Variance, and so on.
4. Finally, neighbor nodes also have their learnable function, g_{w_2}(neighbor node).

These requirements lead us to Equation 11.1a, where "me" represents the node itself and "them" represents one particular neighbor node.

$$f_{w_1}[\text{me}, \sum_{\text{neighbors}} g_{w_2}(\text{them})] \tag{11.1a}$$

$$f_{w_1}[\text{me}, \sum_{\text{neighbors}} g_{w_2}(\text{me}, \text{them})] \tag{11.1b}$$

$$f_{w_1}[\text{me}, \sum_{\text{neighbors}} s_{w_2}(\text{me}, \text{them}) g_{w_3}(\text{them})] \tag{11.1c}$$

More generally, g_{w_2} can depend on "me" and "them" together to explain their connections, as shown in Equation 11.1b.

We can further factorize $g_{w_2}(\text{me}, \text{them})$ into $s_{w_2}(\text{me}, \text{them})$ and $g_{w_3}(\text{them})$ and get Equation 11.1c, where $g_{w_3}(\text{them})$ is regarded as the learnable function of "them" and $s_{w_2}(\text{me}, \text{them})$ is regarded as the connection (or similarity) of "me" and "them". In Transformer Architecture, $s_{w_2}(\text{me}, \text{them})$ is also known as "attention" because it is a learned amount of how much we pay attention to this neighbor.

We can also understand $\sum_{\text{neighbors}} s_{w_2}(\text{me}, \text{them}) g_{w_3}(\text{them})$ as a weighted average of $g_{w_3}(\text{them})$ with $s_{w_2}(\text{me}, \text{them})$ as the weights. We can even use these weights to do softmax.

Food for thought: If we view the GNN in an adjacency matrix formulation, is there a way to use our usual CNN weight operations more directly?

Answer: There is a way to leverage adjacency matrix formulation with use-cases in graph signal processing for signal reflection.

Researchers have generalized the idea of signal processing from one- or two-dimensional functions to general graph relationships. A convolution can be seen as an operation that respects the natural shift invariance on the infinite topology. The infinite topology means that the signal can keep going to the right or left. For a graph, there is no information about what the natural shift would be but there is indeed something we can do with the adjacency matrix. We can take a walk on the graph, or we can take products of the adjacency edge and itself. We can think of shifts on a graph as a kind of repeated product on the unit line. As an example, we can generalize the infinite line to the finite line by shifting on the circle because we can rotate on the circle.

There is also a beautiful connection between convolutions and the invocation of a different domain called frequency domain convolutions. In signal processing, convolution in the original domain corresponds to the multiplication in the frequency domain. But it turns out that the frequency domain is just the eigenbasis corresponding to a particular matrix associated with the structure because convolutions commute with each other and can share the eigenbasis. This allows people to consider what actions can commute with such a matrix formulation of a graph network, with one such formulation being the adjacency matrix. So you could imagine what operations (as matrices) can commute with the adjacency matrix. Those will be the counterpart of convolutions and will respect the entire graph in this abstract way. Graph signal processing considers those objects relevant in graph neural nets. This entire approach is sometimes encapsulated by the spectral methods.

However, there are also drawbacks to this representation. (*https://distill.pub/2021/gnn-intro/*) The number of nodes in a graph can sometimes be on the order of millions, and the number of edges per node can be highly variable. Often, this leads to very sparse adjacency matrices, which are space-inefficient. Another problem is that many adjacency matrices can encode the same connectivity, and there is no guarantee that these different matrices would produce the same result in a deep CNN (that is to say, they are not permutation invariant).

11.3.2 Pooling

Pooling (downsampling) groups similar nodes and coarsens the image in CNNs. Here, the similarity usually means how close the pixels are. In GNN, we just need a similar clustering method to "coarsen" the graph. Under the assumption that the graph topology is fixed, we can pre-compute a specific clustering of this graph based on its topological attribute. In this way, the four nodes in Figure 11.1 are shrunk to two nodes in Figure 11.2.

The clustering can also be learnable. We can use some similarity measures to cluster, like similar neighborhoods or similar values inside them. We can even use the learned similarity to simulate the effect of the clustering without doing a full clustering.

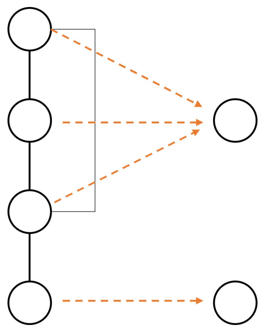


Figure 11.2: Example of pre-computed pooling in GNN

However, the use of pooling in GNN is not as useful as in CNN. Researchers have done experiments where they replaced well-defined pooling with random downsampling and found the network also performed very well. The reason is that the network can learn the appropriate information in other places, like $s_{w_2}(\text{me}, \text{them})$ in Equation 11.1c. The actual activation function, f_{w_1}, can determine certain patterns of "me" and "them" for the next layer, meaning it will learn to cluster information together in certain places if necessary. GNN has flexibility in the architecture itself.

11.3.3 What doesn't change?

In fact, almost everything else remains the same. For example, the residual connection only requires the structure of the object to be the same across layers, which GNN satisfies as well. GNN can also employ all normalization used in CNN, like layer norm, batch norm, instance norm, group norm, and so on. All the normalization needs are that the outputs have some sense of "likeness" to average over.

11.4 What if the graph topology is not fixed?

Let's generalize our assumption: the graphs can still give a single output but no longer have the same topology. What's the most naive thing we could do?

We could try to force the graphs into a similar shape. One way of doing so is to get rid of all topology, make all the nodes fully connected, and use edge labels to tell whether they were connected in the original or not. However, this does not work so well, especially from a computational point of view, because lots of graphs of interest are sparsely connected. Making the graph fully connected will result in a lot of unnecessary computation.

So are there any other solutions? In fact, we could ignore the topology mismatch and see whether it still works. To see whether something still works, two different parts need to be checked, whether it can still run and whether it can give good answers if it can still run.

Will the network still run if we have lots of different graphs as our data during training?

This question can also be broken down into two questions: does it still run as pseudo-code, and does it still run as code?

In terms of running pseudo-code, the answer is yes. Because everything is local, we are just iterating over local neighborhoods. We just have fewer local neighborhoods or more local neighborhoods for different graphs. The residual connections would definitely still work as long as the topology of each layer stays the same. The normalization will still be meaningful, for example, if we just divide the nodes by the different numbers for different sizes of layers in the layer norm. It's only when the nodes are clustered ahead of time that the network may not be able to run. However, we seldom employ clustering because it does not make much of a difference (as mentioned before). Therefore, we can still run it as the pseudo-code.

In terms of running the actual code, we might have to worry about the size of the arrays allocated to make sure things fit. But the important thing is that the weights are not changed. The number of weights we have to learn does not change even if we have more different graphs.

Food for thought: Do we need to pad the graph with null nodes?

In CNN, we need to care about the pixels on the boundaries of images because they have less neighbor pixels. Usually we have zero padding for them or we just ignore these pixels. However, the nodes do not have the same number of neighbors generically. We don't have to pad the graph with null nodes because we never have to compute anything for the null nodes.

11.5 A little intro to RNN

After learning GNN, one question that pops up is whether we could have more weight sharing. In a ConvNet traditionally, the weights are shared within a layer but not across layers. We talked about the idea of the neural ODE, which was the perspective of a ConvNet as a ResNet solved as a differential equation. If the convergence of the neural ODE is wanted, we had to invoke some kind of weight sharing, at least hard or soft weight sharing across layers to ensure the limit existence since there were similar behaviors with respect to time. This requirement raises the question: should we share weights across layers?

We should do weight-sharing if the hierarchical structure has a self-similarity at different scales. This is one of the key design choices in the RNN family.

- Note that the word "layer" can be used in the same way we thought about the layer from a convolution point of view, which is what we backprop through. Once we implement weight sharing across layers, we can still backprop through it.

Let's give an example of an RNN now. Consider a task where you are required to identify a person's attributes based on their name. First, you represent the name by a sequence of characters, then represent these characters as a graph with internal labels. As such, we associate different names with different sizes of graphs, as shown in Figure 11.3.

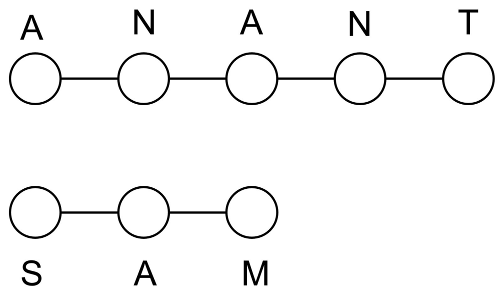


Figure 11.3: Graph interpretation of name stings

One possible solution is to just use a GNN. However, when people started working on these problems, they were actually motivated by the connection with signal processing. In signal processing, you might have seen finite impulse response (FIR) filters and infinite impulse response (IIR) filters. The difference between them is that IIR filters have internal states, like momentum form in the momentum acceleration method. So for the things that are sequential in nature, we can employ the analogy of an IIR filter and think of a network that has internal states. We can treat these self-connections as a kind of internal state.

Another thing worth mentioning is that FIR filters act on the input but the IIR filter consumes the input, one at a time, like the momentum consuming the gradient in each cycle. The whole point of an RNN is to have this eating input behavior used in the network.