CS 189/289A Introduction to Machine Learning
Fall 2023 Jennifer Listgarten, Jitendra Malik DIS12

1 Message Passing Graph Neural Network

Graphs are a representation which supports arbitrary (pairwise) relational structure. To have a neural network model that can operate on arbitrary relational structures, we need to introduce specialized formalism, which can broadly be classified under the umbrella term of *graph neural networks (GNNs)*. One desiderata of a GNN is to be able to learn representations of nodes that depends on the structure of the graph, and can hence be useful for downstream tasks of interest. *Message passing* is a general formalism used by GNNs to learn such representations.

In this problem, we will first give a concrete illustration of the basic building blocks of message passing. Then, we will explore three "flavors" of message passing that underpin the vast majority of GNN layers in the literature: convolutional, attentional, and general message passing. We will see that some familiar neural network architectures, e.g. convolution and attention layers, can be formulated as message passing on a graph with particular parameterizations of the messages. Finally, we will view GNNs from the lens of imposing inductive bias on the functions a model can learn, based on our prior on the underlying symmetry structure of the data.

To start, let's define some notations. Suppose we are given an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$, along with a set of node features $\mathbf{X} \in \mathbf{R}^{d\times|\mathcal{V}|}$. We wish to use this information to generate *learned node embeddings* $\mathbf{z}_u, \forall u \in \mathcal{V}$. During each message passing iteration in a GNN, a *hidden embedding* $\mathbf{h}_u^{(k)}$ is generated, representing the updated embedding of node $u \in \mathcal{V}$ in the k iteration, based on the information aggregated from u's graph neighborhood $\mathcal{N}(u)$ (which could include u itself). In its most general form, a message passing update can be expressed as:

$$\begin{aligned}
\mathbf{h}_u^{(k)} &= \phi^{(k)}\left(\mathbf{h}_u^{(k-1)}, \bigoplus\left(\left\{\psi^{(k)}(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}), \forall v \in \mathcal{N}(u)\right\}\right)\right) \\
&= \phi^{(k)}\left(\mathbf{h}_u^{(k-1)}, \bigoplus\left(\left\{\mathbf{m}_{vu}^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)\right) \\
&= \phi^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathbf{m}_u^{(k)}\right)
\end{aligned}$$

where $\phi^{(k)}$ (update function), $\psi^{(k)}$ (message function) are arbitrary differentiable functions (e.g. parameterized by neural networks). $\bigoplus$ is an aggregation operator, typically a nonparametric operation e.g. summation (often with appropriate normalization) or maximum. For notational simplicity, we use $\mathbf{m}_{vu}^{(k)} = \psi^{(k)}(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)})$ to denote the 'message' from a 'sender' node v to 'receiver' node u, and $\mathbf{m}_u^{(k)}$ to denote the aggregated messages from all of u's neighbors. The superscript (k) on functions and embeddings is used to distinguish between different rounds of message passing (often omitted for notational brevity). A single GNN "layer" can include one or multiple rounds of message passing.

Message construction, aggregation and update can be considered the three main building blocks of a message passing layer. The initial embeddings at $k = 0$ are set to the input features of each node: $\mathbf{h}_u^{(0)} = \mathbf{x}_u, \forall u \in \mathcal{V}$. After k iterations of message passing, the embedding $\mathbf{h}_u^{(k)}$ of node u might encode information about all the nodes in u's k-hop neighborhood that is relevant for the training objective. Suppose we run K iterations of message passing in total, we can use the output of the final layer to define the embedding for each node: $\mathbf{z}_u = \mathbf{h}_u^{(K)}, \forall u \in \mathcal{V}$.

(a) The general formulation above is quite abstract. Let's instantiate it concretely by considering a basic GNN message passing block below:

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

where $\mathbf{W}_{\text{self}}^{(k)}, \mathbf{W}_{\text{neigh}}^{(k)} \in \mathbf{R}^{d^{(k)} \times d^{(k-1)}}$ are trainable parameter matrices, σ is an elementwise non-linearity (e.g. ReLU) and $\mathbf{b}^{(k)} \in \mathbf{R}^{d^{(k)}}$ is the trainable bias parameters. Let's first identify the functions that instantiate the message function ψ, the aggregation function $\bigoplus$ and the update function ϕ.

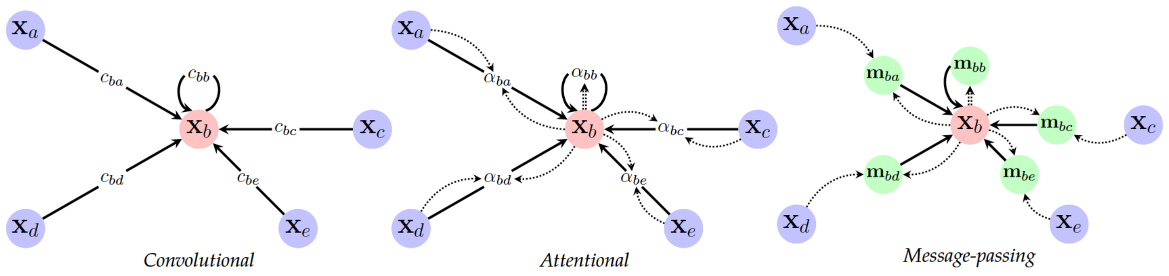

Figure 1: A visualisation of the dataflow for the three flavours of a GNN layer (from Bronstein et al.)

(b) As we have seen, message passing is a fairly general formalism. Designing a message passing GNN layer, which involves choosing appropriate message construction, aggregation and update functions for a task of interest, remains an active area of research. Despite the vast space of possible choices, the majority of GNN architecture in the literature can be derived from three 'flavors' of GNN layers: convolutional, attentional and general message-passing, which differs in their message construction (Figure 1).

Let's first consider the convolutional flavor:

$$\mathbf{h}_u^{(k)} = \phi\left(\mathbf{h}_u^{(k-1)}, \bigoplus\left(\left\{\mathbf{W}^{(k)}\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u)\right\}\right)\right)$$

We can express our familiar 2D convolution layer as convolutional message passing on a graph. Consider an input $I \in \mathbf{R}^{L\times H\times d^{(k-1)}}$ (e.g. consider a $L \times H$ image with $d^{(k-1)}$ input channels). Let our learnable parameter be $\mathbf{W}^{(k)} \in \mathbf{R}^{l\times h\times d^{(k)}\times d^{(k-1)}}$ (e.g. consider a $l \times h$ filter with $d^{(k-1)}$ input channels and $d^{(k)}$ output channels). For any given point (i, j), let the input embedding be $\mathbf{h}_{ij}^{(k-1)} = [\mathbf{h}_{ij1}, \mathbf{h}_{ij2}, ...\mathbf{h}_{ijd^{(k-1)}}] \in \mathbf{R}^{d^{(k-1)}}$, where $\mathbf{h}_{ijc}$ is the value of input channel c.

Suppose $l = h = 3$. For a given input channel c_{in} and output channel c_{out}, denote the filter parameter as $\mathbf{W} \in \mathbf{R}^{l\times h}$. Consider a point $u = (x, y)$. Which points are its neighbors? Express its output embedding $\mathbf{h}_u$ of output channel c_{out} in terms of $\mathbf{W}$ and the input embeddings of its neighbors.

(c) Next, let's consider the attentional flavor:

$$\mathbf{h}_u^{(k)} = \phi\left(\mathbf{h}_u^{(k-1)}, \bigoplus\left(\left\{a(\mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)})\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u)\right\}\right)\right)$$

where $a(\mathbf{h}_u, \mathbf{h}_v) \in \mathbf{R}$ are the attention weights.

Suppose the attention weights are computed by a (single-head) scaled dot-product self-attention. What are the neighbors $\mathcal{N}(u)$ of node u? Express the self-attention weight α_{uv} in terms of the input embedding $\mathbf{h}_u$ of node u, the input embeddings of its neighbors $\mathbf{h}_v$ where $v \in \mathcal{N}(u)$, the key and query weight matrices $\mathbf{W}_k, \mathbf{W}_q \in \mathbf{R}^{d \times d}$.

(d) One important thing to note is a representational containment between these approaches: convolutional $\subseteq$ attention $\subseteq$ message passing, since general message passing amounts to allowing arbitrary learnable message functions, whereas convolutional and attentional message passing poses constraints on the form of the message functions. However, this does not imply that general message-passing GNN is always the most useful variants. What are some factors that might affect the effectiveness of these approaches?

(e) We can also view different neural network architectures as injecting inductive bias into a model. Modern machine learning often operates in very high-dimensional spaces, where the training data might not constrain the space of solutions enough and many solutions are equally good fit to the training data, but many of them might not generalize well. Inductive biases allow a learning algorithm to prioritize one solution over another, independent of the observed data. We can use them to express assumptions about either the data-generating process or the space of solutions, with the hope of reducing sample complexity and achieving better generalization.

Two main ways to impose inductive bias are explicit regularization, and posing architectural constraint that encodes the symmetry structure of the data. In previous lectures, we've learned that neural network architectures e.g. convolution and self-attention encode inductive biases, which can be seen by the type of invariance / equivariance they preserve. What are the invariance / equivariance properties of a CNN layer and a self-attention layer? What about a GNN layer? How does a GNN layer achieve its invariance / equivariance, considering our discussion of message passing above?