CS 189/289A Introduction to Machine Learning
Fall 2024 Jennifer Listgarten, Saeed Saremi

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.

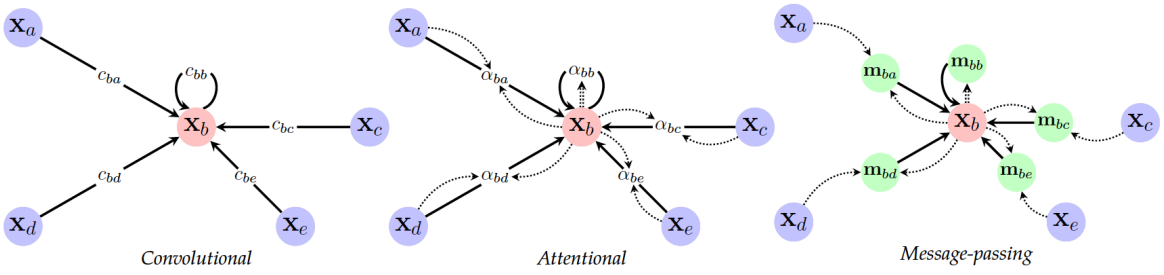

Figure 1: A visualisation of the dataflow for the three flavours of a GNN layer (from Bronstein et al.)

(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 differ 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 $I \in \mathbf{R}^{H\times W\times d^{(k-1)}}$ (e.g. consider a $H \times W$ image with $d^{(k-1)}$ input channels). This image can be represented as a graph in the form of an $H \times W$ grid.

Let our learnable parameters for this layer be $\mathbf{W}^{(k)} \in \mathbf{R}^{h\times w\times d^{(k)}\times d^{(k-1)}}$ (a convolutional layer with filters of size $h \times w$, $d^{(k-1)}$ input channels and $d^{(k)}$ output channels). For any given point (i, j) in the grid graph, 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 at pixel (i, j).

Suppose $h = w = 3$. Consider a point $u = (x, y)$. Which points are its neighbors? Express its output embedding $\mathbf{h}_u^{(k)}$ (or $\mathbf{h}_{xy}^{(k)}$) in terms of $\mathbf{W}^{(k)}$ 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?

2 Kernel PCA

You have seen how to use PCA to do dimensionality reduction by projecting the data to a subspace that captures most of the variability visible in the observed features. The underlying hope is that these directions of variation are also relevant for prediction the quantities of interest.

Standard PCA works well for data that is roughly Gaussian shaped, but many real-world high dimensional datasets have underlying low-dimensional structure that is not well captured by linear subspaces. However, when we lift the raw data into a higher-dimensional feature space by means of a nonlinear transformation, the underlying low-dimensional structure once again can manifest as an approximate subspace. Linear dimensionality reduction can then proceed. As we have seen in class so far, kernels are an alternate way to deal with these kinds of nonlinear patterns without having to explicitly deal with the augmented feature space. This problem asks you to discover how to apply the "kernel trick" to PCA.

Let $\mathbf{X} \in \mathbb{R}^{n \times \ell}$ be the data matrix, where n is the number of samples and ℓ is the dimension of the raw data. Namely, the data matrix contains the data points $\mathbf{x}_j \in \mathbb{R}^{\ell}$ as rows

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{pmatrix} \in \mathbb{R}^{n \times \ell}. \tag{1}$$

(a) Compute $\mathbf{X}\mathbf{X}^\top$ in terms of the singular value decomposition $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ where $\mathbf{U} \in \mathbb{R}^{n \times n}, \mathbf{\Sigma} \in \mathbb{R}^{n \times \ell}$ and $\mathbf{V} \in \mathbb{R}^{\ell \times \ell}$. Notice that $\mathbf{X}\mathbf{X}^\top$ is the matrix of pairwise Euclidean inner products for the data points. How would you get $\mathbf{U}$ if you only had access to $\mathbf{X}\mathbf{X}^\top$?

(b) Given a new test point $\mathbf{x}_{test} \in \mathbb{R}^\ell$, one central use of PCA is to compute the projection of $\mathbf{x}_{test}$ onto the subspace spanned by the k top singular vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$.

Express the scalar projection $z_j = \mathbf{v}_j^\top \mathbf{x}_{test}$ onto the j-th principal component as a function of the inner products

$$\mathbf{X}\mathbf{x}_{test} = \begin{pmatrix} \langle \mathbf{x}_1, \mathbf{x}_{test} \rangle \\ \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_{test} \rangle \end{pmatrix}. \tag{2}$$

Assume that all diagonal entries of $\mathbf{\Sigma}$ are nonzero and non-increasing: $\sigma_1 \geq \sigma_2 \geq \cdots > 0$.

Hint: Express $\mathbf{V}^\top$ *in terms of the singular values* $\mathbf{\Sigma}$*, the left singular vectors* $\mathbf{U}$ *and the data matrix* $\mathbf{X}$*.*

(c) How would you define kernelized PCA for a general kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ (to replace the Euclidean inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$)? For example, the RBF kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\delta^2}\right)$.

Describe this in terms of a procedure which takes as inputs the training data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^\ell$ and the new test point $\mathbf{x}_{test} \in \mathbb{R}^\ell$, and outputs the analog of the previous part's z_j coordinate in the kernelized PCA setting. You should include how to compute $\mathbf{U}$ from the data, as well as how to compute the analog of $\mathbf{X}\mathbf{x}_{test}$ from the previous part.