Graph Neural Networks - Notes

Nihal V. Nayak

March 2020

1 Introduction

Graph Neural Networks (GNN) is a type of neural network which learns the structure of a graph. Learning graph structure allows us to represent the nodes of the graph in the euclidean space which can be useful for several downstream machine learning tasks. Recent work on GNN has shown impressive performance on link prediction, graph classification and semi-supervised tasks (Hamilton et al., 2017b). Since there is a growing interest in the machine learning community to learn more about these techniques, this document provides an introduction to GNN[1].

The document is organized as follows: First, we introduce the basic concepts of graphs and networks. Second, we describe the major steps used in GNNs to compute node embeddings. Next, we describe three GNN techniques that are commonly mentioned in the existing literature. Finally, we include a limited survey of other notable works in the field.

2 Basics

In this section, we will review the basics of networks that will be useful to understand the forthcoming sections.

Graphs consist of a set of nodes V and edges E. They are normally represented with an unweighted adjacency matrix A, which consists of 1s and 0s.

For example, take a look at the graph in figure 1. The graph has 3 nodes and 2 edges. The adjacency matrix is as follows:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{1}$$

The degree matrix is a diagonal matrix that contains information about the degree of each node. The Degree of a node indicates the number of terminating edges for a node (self-loop counts as 2). For example, the degree matrix for fig. 1 is as follows:

[1] This document assumes that you are familiar with the basics of deep learning

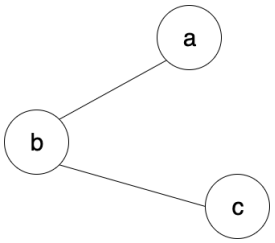

Figure 1: Graph with 3 nodes and 2 undirected edges

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

In GNN, each node v is associated with a feature vector $x_v \in \mathcal{R}^d$. Typically, the feature vector is randomly initialized and trained on a downstream task.

Although we have not introduced graph neural network, it would be useful to know how to compute the neighbors for a node. For example, from Figure 1 the neighbours of the node b are:

$$\mathcal{N}(b) = \{a, c\} \tag{3}$$

where $\mathcal{N}(.)$ is the neighbour function. Likewise, the neighbor of the node a is:

$$\mathcal{N}(a) = \{b\} \tag{4}$$

As you can see, each node can have a varying number of neighbors. Therefore, we need a neural network that can deal with the varying number of neighbors.

3 Learning on Graphs

Graph Neural Network learns the structure of the graph such that the node embedding reflects the structure in the euclidean space (Hamilton et al., 2017b).

A basic GNN layer consists of two steps:

- AGGREGATE: For a given node, the AGGREGATE step applies a permutation invariant function to its neighbors to generate the aggregated node feature
- COMBINE: For a given node, COMBINE step passes the aggregated node feature to a learnable layer to generate node embedding for the GNN layer

The GNN layers are stacked together to increase the receptive field of clustering. For example, in Fig. 2 we have 2 GNN layers. This means that for every node, we aggregate over the 2-hop neighborhood.

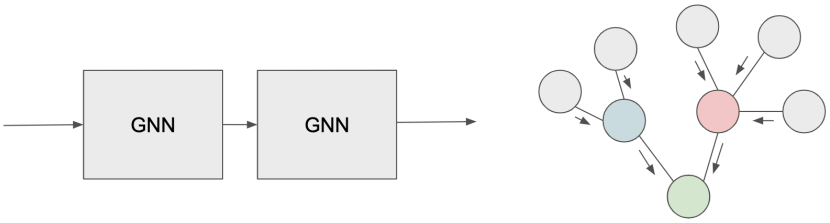

Figure 2: *Left:* 2 GNN layers stacked together and *Right:* shows the how the node feature is generated for a node (green) in a 2 layer GNN. The node (green) learns the node embedding by aggregating over 2-hop neighbourhood.

AGGREGATE and COMBINE steps are sometimes called message passing phase and readout phase in the message passing neural network framework (Gilmer et al., 2017).

3.1 Formal Definition

Consider the Graph $G = (V, E)$, where V is the set of node with features X_v. There are two main steps - AGGREGATE and COMBINE at each l-th layer of the GNN:

$$a_v^{(l)} = \text{AGGREGATE}^{(l)} \left(\left\{ h_u^{(l-1)} \forall u \in \mathcal{N}(v) \right\} \right) \tag{5}$$

where $a_v^{(l)}$ is the aggregated node feature of the neighbourhood, $h_u^{(l-1)}$ is the node feature in neighbourhood $\mathcal{N}(.)$ of node v.

$$h_v^{(l)} = \text{COMBINE}^{(l)} \left(h_v^{(l-1)}, a_v^{(l)} \right) \tag{6}$$

where $h_v^{(l)}$ is the node representation at the l-th iteration. $h_v^{(0)} = x_v$ where x_v is the initial feature vector for the node.

4 Graph Convolutional Networks (GCN)

GCN (Kipf and Welling, 2016) is a graph neural network technique that makes use of the symmetrically normalized graph laplacian[2] to compute the node embeddings. Each GCN block receives node features from the $(l-1)$th GCN block, i.e. the node features from the previous layer is passed to the next GCN layer, which allows for greater laplacian smoothing (Li et al., 2018).

[2]Refer to A for more details on Graph Laplacian.

4.1 Aggregate

Since we don't always have access to the entire graph structure to compute the symmetrically normalized graph laplacian, GCN Aggregator has been interpreted as a Mean Aggregator (Xu et al., 2018a). Therefore,

$$a_v^{(l)} = \text{Mean}\left(\left\{h_u^{(l-1)}, u \in \mathcal{N}(v) \cup \{v\}\right\}\right) \tag{7}$$

For each node v, the algorithm takes the mean of the neighborhood node features along with its feature (self-loop) to compute the aggregated node feature $a_v^{(l)}$ for the lth layer. This formulation makes GCN an inductive graph neural network.

4.2 Combine

The aggregated node feature $a_v^{(l)}$ is multiplied with a learnable weight matrix followed by a non-linear activation (ReLU) to get the node feature $h_v^{(l)}$for the lth layer,

$$h_v^{(l)} = \text{ReLU}\left(W^{(l)} a_v^{(l)}\right) \tag{8}$$

where $W^{(l)}$ is the learnable weight matrix.

5 GraphSage

Graphsage (Hamilton et al., 2017a) is a spatial graph neural network algorithm that learns node embeddings by sampling and aggregating neighbors from multiple hops. Graphsage, unlike GCN, is an inductive learning algorithm which means that it does not require the entire graph structure during learning.

In the original Graphsage paper, the authors describe multiple aggregators - Mean, Pooling and LSTM Aggregator. In this section, we'll be describing the Pool Aggregator and the combine from the original paper. For simplicity, we do not sample the neighborhood nodes as well.

5.1 Aggregate

Graphsage makes use of a max-pooling aggregator to compute an aggregated node feature. All the neighborhood node features are passed through a linear layer. We then pass these features through a non-linear activation before applying *max* filter. Formally,

$$a_v^{(l)} = \max\left(\left\{\sigma\left(W_{\text{pool}}^{(l-1)} h_u^{(l-1)} + b\right), u \in \mathcal{N}(v)\right\}\right) \tag{9}$$

where $a_v^{(l)}$ is the aggregated node feature for the lth layer. $W_{\text{pool}}^{(l-1)}$ and b are the learnable parameters.

5.2 Combine

The combine function concatenates the $(l-1)$th layer node feature $h_v^{(l-1)}$ with the aggregated feature vector $a_v^{(l)}$ and pass the vector through a fully connected neural network:

$$h_v^{(l)} = \text{ReLU}\left(W^{(l)}[h_v^{(l-1)}, a_v^{(l)}]\right) \tag{10}$$

where $W^{(l)}$ is a learnable weight matrix for lth layer.

6 Graph Attention Network (GAT)

Graph Attention Network (Veličković et al., 2018) is a spatial graph neural network technique that uses Self Attention to aggregate the neighborhood node features. Self Attention is equivalent to computing a weighted mean of the neighbourhood node features.

The original paper uses K *masked self attention* modules to aggregate node features. For simplicity, we consider only one self attention block.

6.1 Aggregate

GAT uses an Attention module to compute the weighted mean of the neighborhood node features as follows:

$$a_v^{(l)} = \text{Attention}\left(\left\{h_u^{(l-1)}, u \in \mathcal{N}(v) \cup \{v\}\right\}\right) \tag{11}$$

The authors use self-attention (Bahdanau et al., 2014) but the aggregation technique is agnostic to the choice attention mechanism.

6.2 Combine

The aggregated node feature $a_v^{(l)}$ is multiplied with a learnable weight matrix followed by a non-linear activation to get the node feature $h_v^{(l)}$for the lth layer,

$$h_v^{(l)} = \text{LeakyReLU}\left(W^{(l)} a_v^{(l)}\right) \tag{12}$$

where $W^{(l)}$ is the learnable weight matrix.

7 Related Works

Apart from GCN, Graphsage, and GAT, there are few more works that frequently appear in the literature. SGN (Wu et al., 2019) simplifies GCN by removing non-linearity between the GCN layers. Furthermore, to achieve the same "receptive field" of having K GCN layers, they take the graph laplacian to the power of K and show equivalent performance. Xu et al. (2018b) introduced jumping knowledge networks that aggregate node features generated from

all the previous layers to compute the final vector using mean or max-pooling. This allows the model to learn better structure-aware representations from different neighborhood aggregations. The works described so far have not taken advantage of the relations in the graph. R-GCN (Schlichtkrull et al., 2018) and Directed-GCN (Marcheggiani and Titov, 2017) extend GCN to relational graphs.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org.

Hamilton, W., Ying, Z., and Leskovec, J. (2017a). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40:52–74.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks.

Li, Q., Han, Z., and Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Marcheggiani, D. and Titov, I. (2017). Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations*.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018a). How powerful are graph neural networks?

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. (2018b). Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*.

A Graph Laplacian

The adjacency matrix and degree matrix can be combined to get the Graph Laplacian (commonly occurs in the literature).

$$L = D - A \tag{13}$$

where L is the graph laplacian, D is the degree matrix and A is the adjacency matrix.

The symmetric normalized graph laplacian can be formally defined as:

$$L_{\text{sym}} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{14}$$

where I is the Identity matrix.

The elements of L_{sym} are given by:

$$L_{\text{ij}} = \begin{cases} 1 & if\ i == j \text{ and } deg(v_i) \neq 0 \\ -\dfrac{1}{\sqrt{deg(v_i) deg(v_j)}} & if\ i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

B Original GCN

In section 4, we interpreted GCN (Kipf and Welling, 2016) as a mean aggregator. However, there is a subtle difference between our interpretation and the original GCN.

Consider the adjacency matrix A and its degree matrix D. Let $H^{(l)}$ be the node features for the lth layer and $H^{(0)} = X$. The propagation rule for each GCN layer is as follows:

$$f(H^{(l)}, \tilde{A}) = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right) \tag{16}$$

with $\tilde{A} = I + A$ where I is the identity matrix and degree matrix $\tilde{D}$ of $\tilde{A}$.