Introduction to Graph Neural Networks

Vignesh Kumar Thangarajan
CMPE-258 San Jose State University

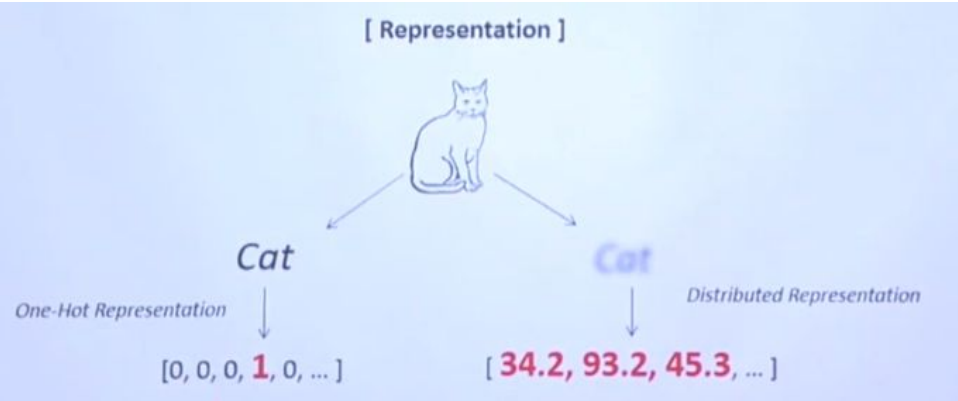
Data Representation

Data in most machine learning algorithms are typically represented in the Euclidean space.

we often have an embedding layer to represent text and flattening layer for image data.

These representations have ruled out any inherent relationship between the data for the processing layers to consider.

Distributed Vector Representation



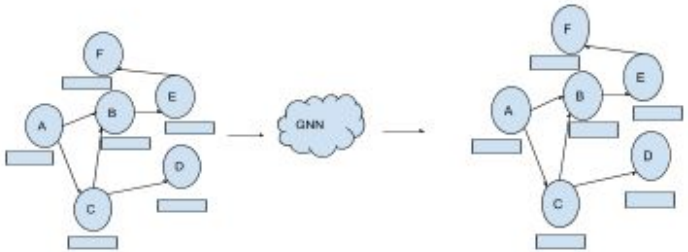
Graph Representation

The graphs can be either directed or undirected graph and it is purely based on the application design consideration.

Once we have a graph representation of data it is fed to the processing layers of GNN.

The output of which will then be used per application need and it can be fed to a loss computation layer and back propagate it to the Graph for each node to adjust the weights to improve the accuracy.

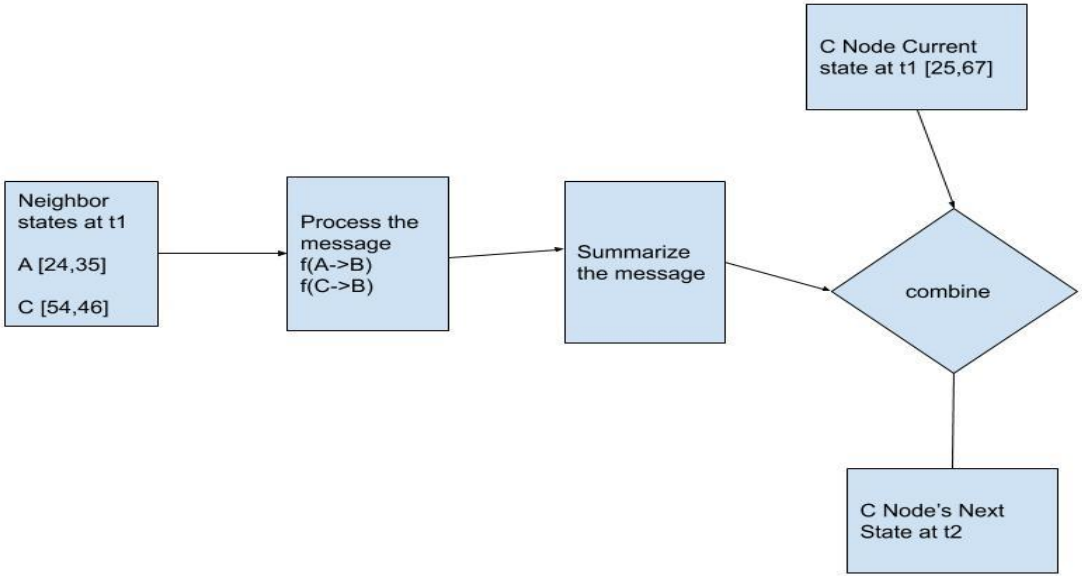
Graph



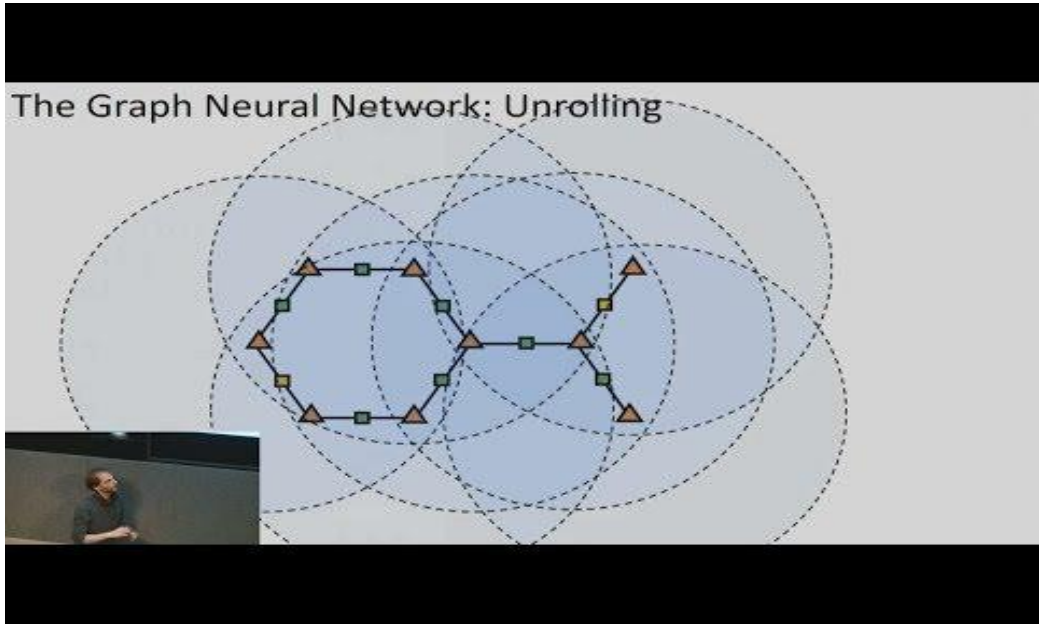
Message Passing

1. In our example, the node B has two neighbors — A and C. So, it will receive updates from both A and C in every clock time. The updates are called **messages**.
2. The messages are then **prepared** using Mathematical functions applied on both the messages independently. A bias term can also be added to keep track of the number of nodes the updates coming from.
3. Prepared messages are then **summarized**. A new function applied over the Union of all messages from all the neighbor nodes.
4. Output of step 3 and current state of node C (at time t-1) is then **combined** and fed to either a recurrent network layer or convolution layer based on the application need.
5. Output of step 4 is now the **new state** of node C at time t.

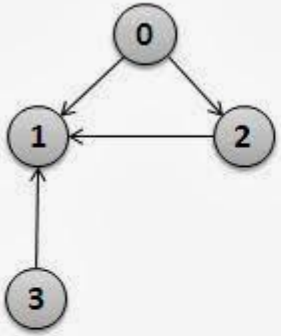
Flow diagram



Message passing to all nodes



Adjacency Matrix



	0	1	2	3
0	0	1	1	0
1	0	0	0	0
2	0	1	0	0
3	0	1	0	0

Adjacency Matrix Representation of Directed Graph

Variants of GNN

We can use several variants of GNN and it is completely based on the problem we want to solve and the design considerations we take.

If we use a RNN for back propagation, it is called Gated GNN (GRU) and Graph Convolution Network (GCN) if we instead use convolution layer.

Comparison of functions in GNN

TABLE 2
Different variants of graph neural networks.

Name	Variant	Aggregator	Updater
Spectral Methods	ChebNet	$\mathbf{N}_k = \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{X}$	$\mathbf{H} = \sum_{k=0}^{K} \mathbf{N}_k \mathbf{\Theta}_k$
	1^{st}-order model	$\mathbf{N}_0 = \mathbf{X}$ $\mathbf{N}_1 = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}$	$\mathbf{H} = \mathbf{N}_0\mathbf{\Theta}_0 + \mathbf{N}_1\mathbf{\Theta}_1$
	Single parameter	$\mathbf{N} = (\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{X}$	$\mathbf{H} = \mathbf{N}\mathbf{\Theta}$
	GCN	$\mathbf{N} = \bar{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\bar{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}$	$\mathbf{H} = \mathbf{N}\mathbf{\Theta}$
Non-spectral Methods	Neural FPs	$\mathbf{h}^t_{\mathcal{N}_v} = \mathbf{h}^{t-1}_v + \sum_{k=1}^{\mathcal{N}_v} \mathbf{h}^{t-1}_k$	$\mathbf{h}^t_v = \sigma(\mathbf{h}^t_{\mathcal{N}_v}\mathbf{W}^{\mathcal{N}_v}_L)$
	DCNN	Node classification: $\mathbf{N} = \mathbf{P}^*\mathbf{X}$ Graph classification: $\mathbf{N} = 1^T_N\mathbf{P}^*\mathbf{X}/N$	$\mathbf{H} = f(\mathbf{W}^c \odot \mathbf{N})$
	GraphSAGE	$\mathbf{h}^t_{\mathcal{N}_v} = \text{AGGREGATE}_t(\{\mathbf{h}^{t-1}_u, \forall u \in \mathcal{N}_v\})$	$\mathbf{h}^t_v = \sigma(\mathbf{W}^t \cdot [\mathbf{h}^{t-1}_v \| \mathbf{h}^t_{\mathcal{N}_v}])$
Graph Attention Networks	GAT	$\alpha_{vk} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_k]))}{\sum_{j\in\mathcal{N}_v}\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_j]))}$ $\mathbf{h}^t_{\mathcal{N}_v} = \sigma\left(\sum_{k\in\mathcal{N}_v}\alpha_{vk}\mathbf{W}\mathbf{h}_k\right)$ Multi-head concatenation: $\mathbf{h}^t_{\mathcal{N}_v} = \Big\|_{m=1}^{M}\sigma\left(\sum_{k\in\mathcal{N}_v}\alpha^m_{vk}\mathbf{W}^m\mathbf{h}_k\right)$ Multi-head average: $\mathbf{h}^t_{\mathcal{N}_v} = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{k\in\mathcal{N}_v}\alpha^m_{vk}\mathbf{W}^m\mathbf{h}_k\right)$	$\mathbf{h}^t_v = \mathbf{h}^t_{\mathcal{N}_v}$
Gated Graph Neural Networks	GGNN	$\mathbf{h}^t_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{h}^{t-1}_k + \mathbf{b}$	$\mathbf{z}^t_v = \sigma(\mathbf{W}^z\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^z\mathbf{h}^{t-1}_v)$ $\mathbf{r}^t_v = \sigma(\mathbf{W}^r\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^r\mathbf{h}^{t-1}_v)$ $\widetilde{\mathbf{h}^t_v} = \tanh(\mathbf{W}\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}(\mathbf{r}^t_v \odot \mathbf{h}^{t-1}_v))$ $\mathbf{h}^t_v = (1 - \mathbf{z}^t_v) \odot \mathbf{h}^{t-1}_v + \mathbf{z}^t_v \odot \widetilde{\mathbf{h}^t_v}$

Applications

1. Calculating Molecular Fingerprints. For example, to study a complex protein structure, we can represent that in graph like each atom as a separate node and their connections as edges.
2. Self healing computer programs.
3. Bayesian Neural Network and Bayesian based GNN. The plain vanilla Bayesian Networks are also represented as graphs. These networks are pivotal in measuring the confidence of classification or prediction accuracy that the network is doing — which is, very relevant in almost all real world applications.

References

1. Microsoft Research
2. https://arxiv.org/pdf/1901.00596.pdf
3. https://www.youtube.com/watch?v=lDRb3CjESmM
4. https://arxiv.org/pdf/1812.08434.pdf
5. https://theaisummer.com/Graph_Neural_Networks/

Special Thanks to Professor - Vijay Eranti