An Introduction to Graph Neural Network(GNN) For Analysing Structured Data

Understand What GNN Is and What GNN Can Do

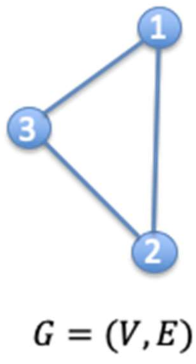
source: Manuchi, via pixabay (CC0)

Graph Neural Network(GNN) recently has received a lot of attention due to its ability to analyze graph structural data. This article gives a gentle introduction to Graph Neural Network. It covers some graph theories for the ease to understand graphs and the problems in analyzing graphs. It then introduces Graph Neural Network in different forms and their principles. It also covers what GNN can do and some applications of GNN.

Graph Theory

First, we need to know what is a graph.

A graph is a data structure consisting of two components: ***vertices,*** and ***edges***. It is used as a mathematical structure to analyze the pair-wise relationship between objects and entities. Typically, a graph is defined as *G=(V, E),* where *V* is a set of nodes and *E* is the edges between them.



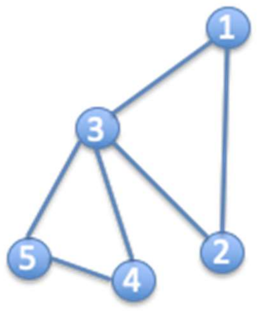
A simple graph. Figure by author

A graph is often represented by an Adjacency matrix, *A*. If a graph has *N* nodes, then A has a dimension of (*N*x*N*). People sometimes provide another feature matrix to describe the nodes in the graph. If each node has *F* numbers of features, then the feature matrix *X* has a dimension of (*N*x*F*).

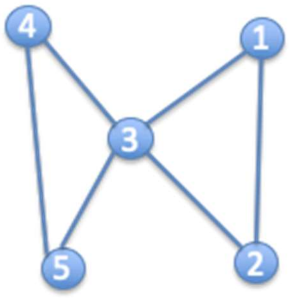
Why Is a Graph Difficult To Analyze?

Firstly, a graph does not exist in a Euclidean space, which means it cannot be represented by any coordinate systems that we are familiar with. This makes the interpretation of graph data much harder as compared to other types of data such as waves, images, or time-series signals(“text” can also be treated as time-series), which can be easily mapped to a 2-D or 3-D Euclidean space.

Secondly, a graph does not have a fixed form. Why? Look at the example below. Graph (A) and Graph (B) have a completely different structure and visually different. But when we convert it to adjacency matrix representation, the two graphs have the same adjacency matrix (if we don't consider the weight of edges). So should we consider these two graphs are the same or different?

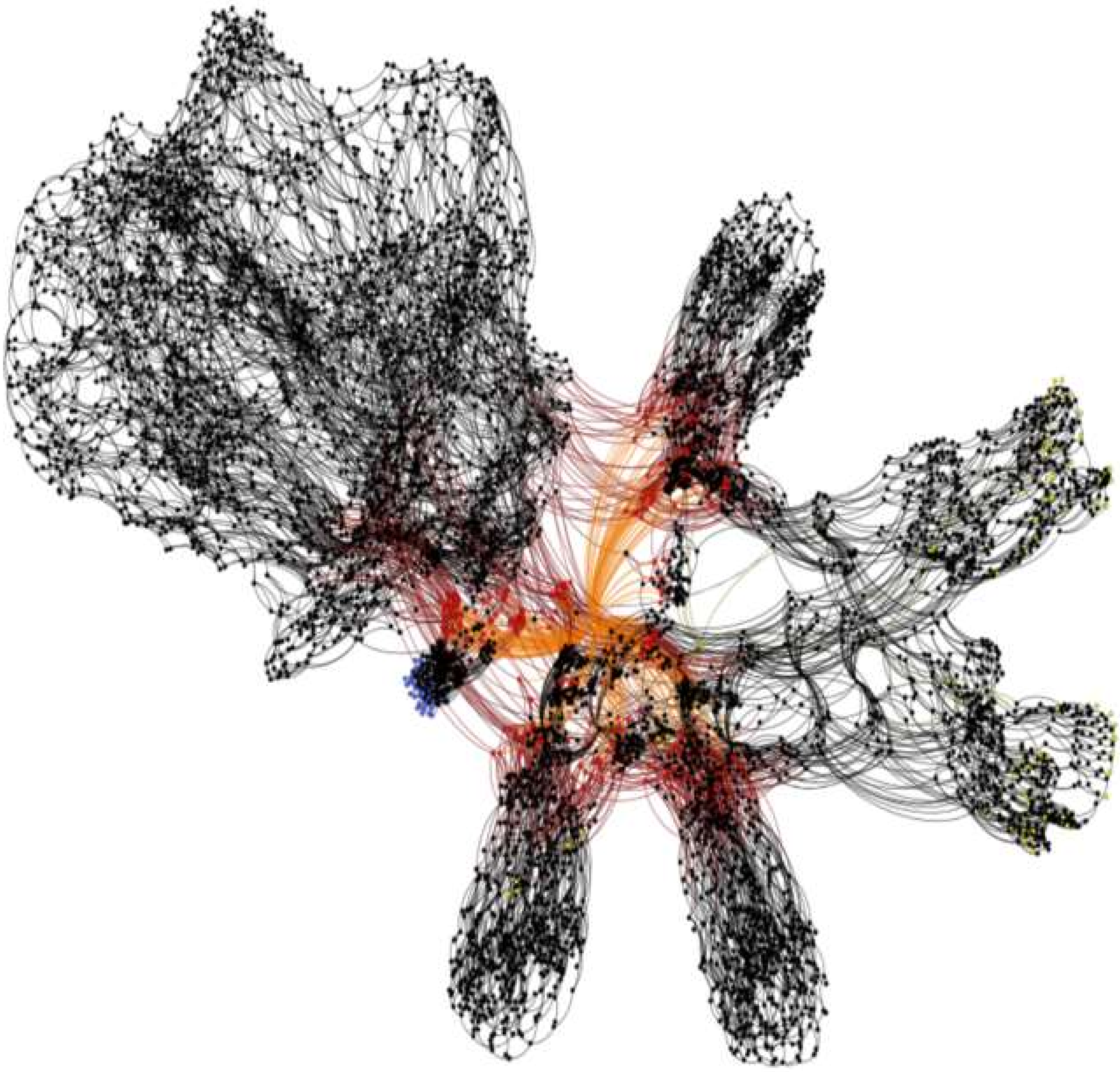


Graph (A). Figure by author



Graph (B). Figure by author

And lastly, a graph is in general hard to visualize for human interpretation. I'm not talking about small graphs like the examples above. I'm talking about giant graphs that involve hundreds or thousands of nodes. The dimension is very high and nodes are densely grouped, making it even difficult for a human to understand the graph. Therefore, it is challenging to train a machine for this task. The example below shows a graph modeling the logic gates in an integrated circuit.



Example of a giant graph: circuit netlist. Figure from J. Baehr et. al. “Machine Learning and Structural Characteristics of Reverse Engineering”

Why Use Graphs?

The reasons that people choose to work on graphs can be summarized in a few points as listed below:

1. Graphs provide a better way of dealing with abstract concepts like relationships and interactions. They also offer an intuitively visual way of thinking about these concepts. Graphs also form a natural basis for analyzing relationships in a social context.

2. Graphs can solve more complex problems by simplifying the problems into simpler representations or transform the problems into representations from different perspectives.
3. Graph Theories and concepts are used to study and model Social Networks, Fraud patterns, Power consumption patterns, Virality and Influence in Social Media. Social Network Analysis (SNA) is probably the best-known application of Graph Theory for Data Science.

Traditional Graph Analysis Methods

Traditional methods are mostly algorithm-based, such as :

1. searching algorithms, e.g. BFS, DFS
2. shortest path algorithms, e.g. Dijkstra's algorithm, Nearest Neighbour
3. spanning-tree algorithms, e.g. Prim's algorithm
4. clustering methods, e.g. Highly Connected Components, k-mean

The limitation of such algorithms is that we need to gain prior knowledge of the graph at certain confidence before we can apply the algorithm. In other words, it provides no mean for us to study the graph itself. And most importantly, there is no way to perform graph level classification.

Graph Neural Network

Graph Neural Network, as how it is called, is a neural network that can directly be applied to graphs. It provides a convenient way for node level, edge level, and graph level prediction task.

There are mainly three types of graph neural networks in the literature:

1. Recurrent Graph Neural Network

2. Spatial Convolutional Network
3. Spectral Convolutional Network

The intuition of GNN is that nodes are naturally defined by their neighbors and connections. To understand this we can simply imagine that if we remove the neighbors and connections around a node, then the node will lose all its information. Therefore, the neighbors of a node and connections to neighbors define the concept of the node.

Having this in mind, we then give every node a state *(x)* to represent its concept. We can use the node state *(x)* to produce an output *(o)*, i.e. decision about the concept. The final state *(x_n)* of the node is normally called "node embedding". The task of all GNN is to determine the "node embedding" of each node, by looking at the information on its neighboring nodes.

We will start with the most pioneer version of Graph Neural Network, Recurrent Graph Neural Network, or *RecGNN*.

Recurrent Graph Neural Network

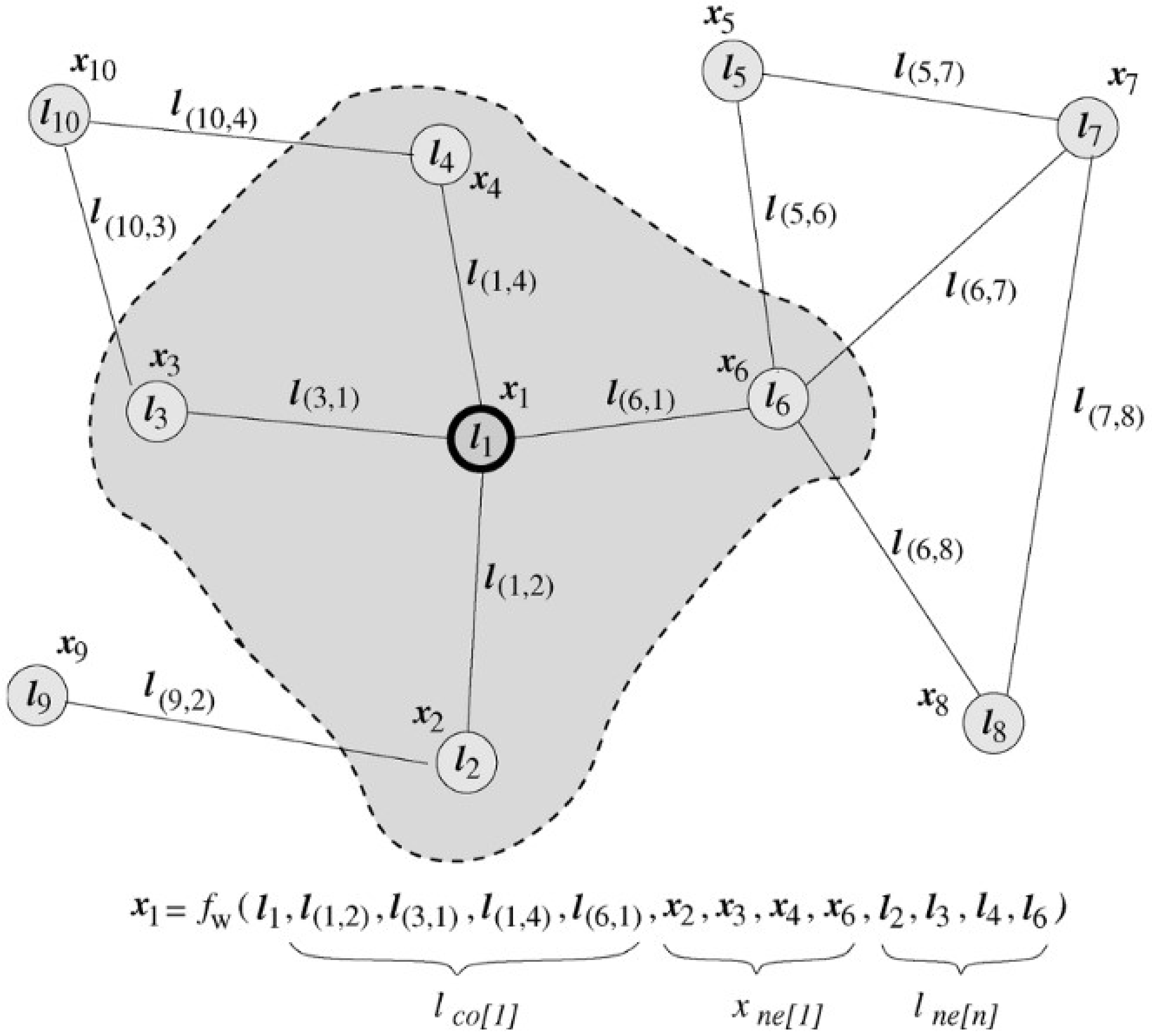
As introduced in the original GNN paper, RecGNN is built with an assumption of Banach Fixed-Point Theorem. Banach Fixed-Point Theorem states that: *Let (X,d) be a complete metric space and let (T:X→X) be a contraction mapping. Then T has a unique fixed point (x∗) and for any x∈X the sequence T_n(x) for n→∞ converges to (x∗).* This means if I apply the mapping *T* on *x* for *k* times, x^k should be almost equal to x^(k-1), i.e.:

$$x^k = T\left(x^{k-1}\right), \mathrm{k} \in (1, n)$$

RecGNN defines a parameterized function *f_w:*

$$\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\text{co}[n]}, \boldsymbol{x}_{\text{ne}[n]}, \boldsymbol{l}_{\text{ne}[n]})$$

where ***l_n, l_co, x_ne, l_ne*** represents the features of the current node **[*n*]**, the edges of the node **[*n*]**, the state of the neighboring nodes, and the features of the neighboring nodes. (In the original paper, the author referred node features as node labels. This might make some confusion.)



An illustration of node state update based on the information in its neighbors. Figure from "The Graph Neural Network Model"

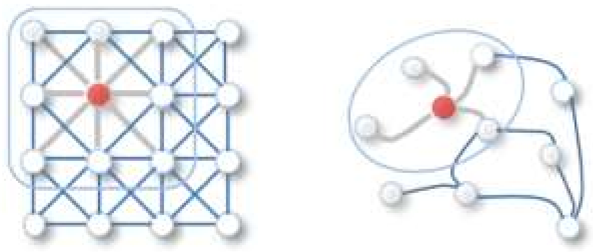
Finally, after k iterations, the final node state is used to produce an output to make a decision about each node. The output function is defined as:

$$\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$$

Spatial Convolutional Network

The intuition of Spatial Convolution Network is similar to that of the famous CNN which dominates the literature of image classification and segmentation tasks. To understand CNN on images, you can check out this post which explains CNN in detail.

In short, the idea of convolution on an image is to sum the neighboring pixels around a center pixel, specified by a filter with parameterized size and learnable weight. Spatial Convolutional Network adopts the same idea by aggregate the features of neighboring nodes into the center node.


Left: Convolution on a regular graph such as an image. Right: Convolution on the arbitrary graph structure. Figure from "A Comprehensive Survey on Graph Neural Networks"

Spectral Convolutional Network

As compared to other types of GNN, this type of graph convolution network has a very strong mathematics foundation. Spectral Convolutional Network is built on graph signal processing theory. And by simplification And approximation of graph convolution.

By *Chebyshev polynomial approximation* (Hammond et al. 2011), graph convolution can be simplified to below form:

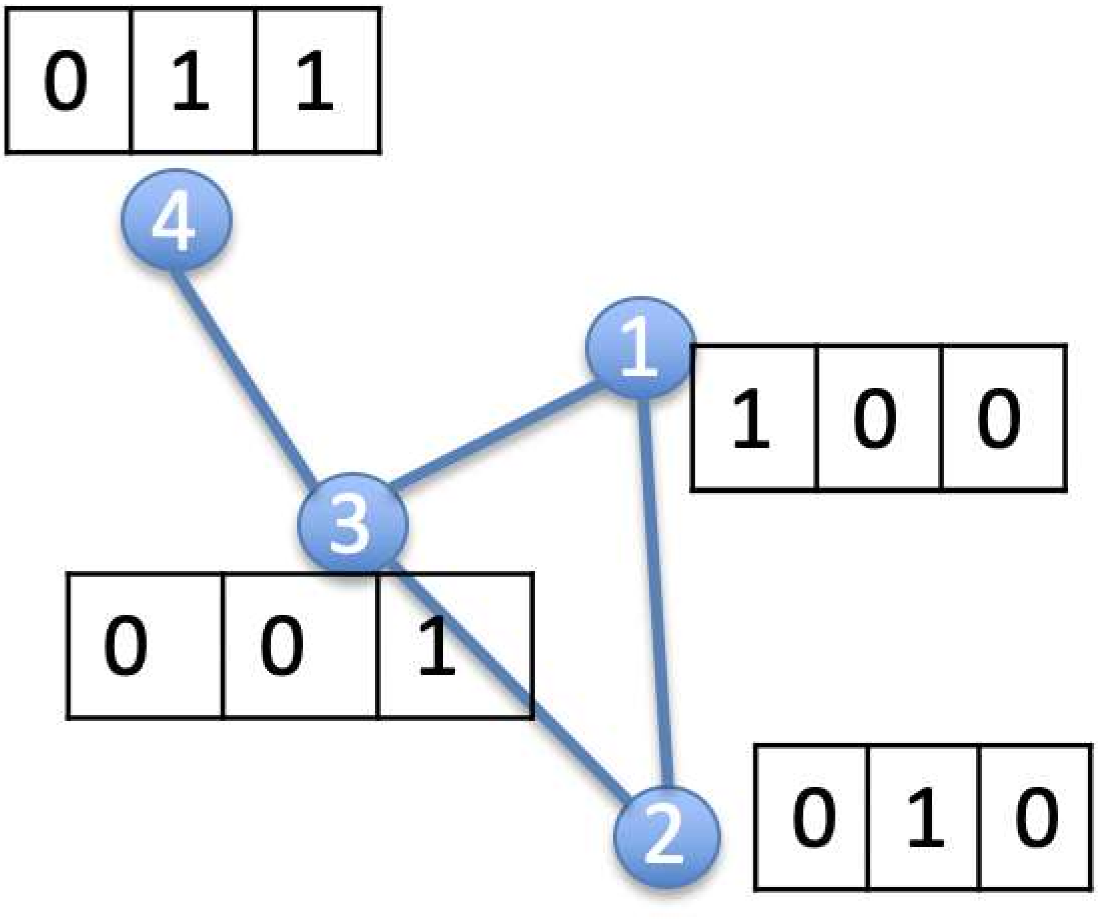
$$g_{\theta'} * x \approx \sum_{k=0}^{K} \theta_k T_k(\Lambda),$$

After further simplification, the GCN paper suggests a 2-layered neural network structure, which can be described in one equation as below:

$$Z = f(X, A) = softmax\left(\hat{A} Relu(\hat{A} X W^{(0)}) W^{(1)}\right)$$

where A_head is the pre-processed Laplacian of original graph adjacency matrix A. (Details of the mathematics can be found in GCN paper. It will take much effort to fully explain.)

This formula looks very familiar if you have some experience in machine learning. This is nothing but two fully-connected layer structure that is commonly used. But it indeed serves as graph convolution in this case. I will show why it can perform graph convolution below.



Example of a graph with a feature assigned to each node. Figured by author

Let’s consider we have a simple graph with 4 nodes. Each of these nodes is assigned a feature matrix as shown in the figure above. It is easy to come out with a graph adjacency matrix and feature matrix as shown below:

$$\begin{array}{|c|c|c|c|}\hline 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 \\ \hline \end{array} \qquad \begin{array}{|c|c|c|}\hline 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 1 \\ \hline 0 & 1 & 1 \\ \hline \end{array}$$

Adjacency matrix (A) Feature matrix (X)

Example of the adjacency matrix and feature matrix. Figure by author

Note that the diagonal of the adjacency matrix is purposely changed to ‘1’ to add a self-loop for every node. This is to include the feature of every node itself when we perform feature aggregation.

We then perform *AxX (*let’s forget about the Laplacian of A and the weight matrix *W* first for simplicity of explanation.*)*

$$A \times X = H$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 1 & 1 \\ 1 & 2 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

$$h_{1,1} = A_{1,1}X_{1,1} + A_{1,2}X_{2,1} + A_{1,3}X_{3,1} + A_{1,4}X_{4,1}$$

$$h_{1,2} = A_{1,1}X_{1,2} + A_{1,2}X_{2,2} + A_{1,3}X_{3,2} + A_{1,4}X_{4,2}$$

$$h_{1,3} = A_{1,1}X_{1,3} + A_{1,2}X_{2,3} + A_{1,3}X_{3,3} + A_{1,4}X_{4,3}$$

Example of graph convolution by matrix multiplication. Figure by author

The result of matrix multiplication is shown in the rightmost matrix. Let’s look at the resulted feature of the first node as an example. It is not hard to notice that the result is a sum of all features of [node 1] including the feature of [node 1] itself, and features in [node 4] are not included since its not the neighbor of [node 1]. Mathematically, the adjacency matrix of the graph has value ‘1’ only when there is an edge, and ‘0’ otherwise. This makes the matrix multiplication become the summation of features of nodes that are connected to the reference node.

Therefore, Spectral Convolutional Network and Spatial Convolutional Network, although started on a different basis, share the same propagation rule.

All convolutional graph neural networks currently available share the same format. They all try to learn a function to pass the node information around and update node state by this message passing process.

Any Graph Neural Network can be expressed as a Message Passing Neural Network (J. Gilmer et al. , 2017) with a ***message-passing*** function, a ***node update*** function and a ***readout*** function.

- A Message Passing function: M_t: $m^{l+1} = M_t(H^l, A)$
- A Node Update function: U_t: $H^{l+1} = U_t(H^l, m^{l+1})$
- A Readout function: R_t (for graph classification): $y = R(H^l)$

What Can GNN do?

The problems that GNN solve can be broadly classified into three categories:

1. Node Classification
2. Link Prediction
3. Graph Classification

In *node classification*, the task is to predict the node embedding for every node in a graph. This type of problem is usually trained in a semi-supervised way, where only part of the graph is labeled. Typical applications for node classification include citation networks, Reddit posts, Youtube videos, and Facebook friends relationships.

In *link prediction*, the task is to understand the relationship between entities in graphs and predict if two entities have a connection in between. For example, a recommender system can be treated as link prediction problem where the model is given a set of users' reviews of different products, the task is to predict the users' preferences and tune the recommender system to push more relevant products according to users' interest.

In *graph classification*, the task is to classify the whole graph into different categories. It is similar to image classification but the target changes into graph domain. There is a wide range of industrial problems where graph classification can be applied, for example, in chemistry, biomedical, physics, where the model is given a molecular structure and asked to classify the target into meaningful categories. It accelerates the analysis of atom, molecule or any other structured data types.

Some Real Applications

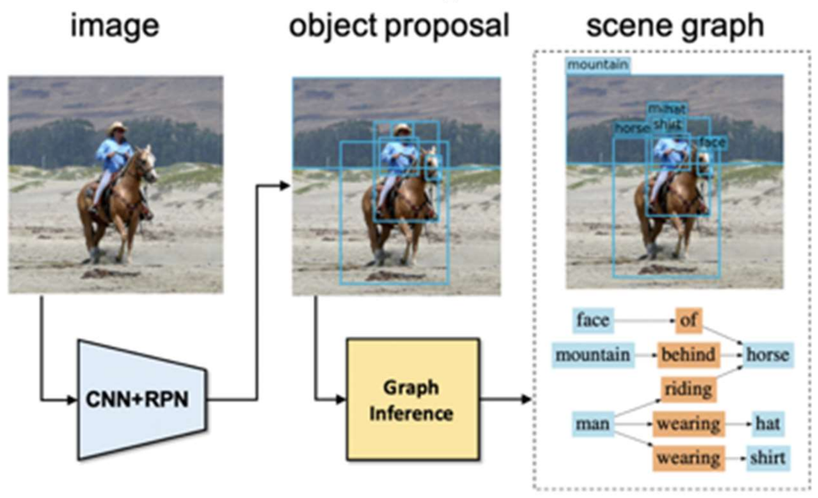
Having understand what types of analysis that GNN can perform, you must be wondering what are the real things that I can do with graphs. Well, this section will give you more insights into GNN's real-world applications.

GNN in Natural Language Processing

GNN is widely used in Natural Language Processing (NLP). Actually, this is also where GNN initially gets started. If some of you have experience in NLP, you must be thinking that text should be a type of sequential or temporal data which can be best described by an RNN or an LTSM. Well, GNN approaches the problem from a completely different angle. GNN utilized the inner relations of words or documents to predict the categories. For example, the citation network is trying to predict the label of each paper in the network given by the paper citation relationship and the words that are cited in other papers. It can also build a syntactic model by looking at different parts of a sentence instead of purely sequential as in RNN or LTSM.

GNN in Computer Vision

Many CNN based methods have achieved state-of-the-art performance in object detections in images, but yet we do not know the relationships of the objects. One successful employment of GNN in CV is using graphs to model the relationships between objects detected by a CNN based detector. After objects are detected from the images, they are then fed into a GNN inference for relationship prediction. The outcome of the GNN inference is a generated graph that models the relationships between different objects.



Scene Graph Generation. Figure from D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proc. of CVPR*, 2017

Another interesting application in CV is image generation from graph descriptions. This can be interpreted as almost the reverse of the application mentioned above. The traditional way of image generation is text-to-image generation using GAN or autoencoder. Instead of using text for image description, graph to image generation provides more information on the semantic structures of the images.

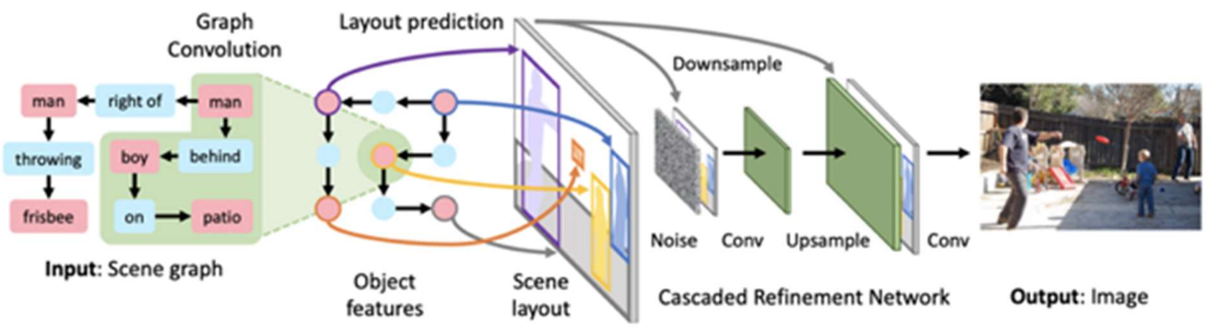


Image generated from scene graphs. Figure from J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," in *Proc. of CVPR*, 2018

The most interesting application I would like to share is zero-shot learning (ZSL). You can find this post for a comprehensive introduction to ZSL. In short, ZSL is trying to learn to classify a class given **NO** training samples (of the target classes) at all. It was quite challenging because if no training samples were given, we need to let the model "think" logically to recognize a target. For example, if we were given three images (as shown in the figure below) and told to find "okapi" among them. We may not have seen an "okapi" before. But if we were also given the information that an "okapi" is a deer-face animal with four legs and has zebra-striped skin, then it is not hard for us to figure out which one is "okapi". Typical methods are simulating this "thinking process" by converting the detected features into text. However, text encodings are independent among each other. It is hard to model the relationships between the text descriptions. In other hard, graph representations well model these relationships, making the machine to think in a more "human-like" manner.

Figure from X. Wang, Y. Ye, and A. Gupta, "Zero-shot recognition via semantic embeddings and knowledge graphs," in *CVPR 2018*

GNN in Other Domains

More practical applications of GNN include human behavior detection, traffic control, molecular structure study, recommender system, program verification, logical reasoning, social influence prediction, and adversarial attack prevention. Below shows a graph that models the relationships of people in a social network. GNN can be applied to cluster people into different community groups.

Graph of Social Network. Image from GDJ, via Pixabay

Conclusion

We went through some graph theories in this article and emphasized on the importance to analyze graphs. People always see machine learning algorithm as a "**black box**". Most machine learning algorithms only learn from the features of training data but there is no actual logic to perform. With graphs, we might be able to pass some "logics" to the machine and let it "think" more naturally.

GNN is still a relatively new area and is worthy of more research attention. It is a powerful tool to analyze graph data. Yet it is not limited to only problems in graphs. It can be easily generalized to any studies that can be modeled by graphs. And graph modeling is a natural way to analyze a problem.

References:

1. F.Scarselli, M.Gori, “The graph neural network model,” *IEEE Transactions on Neural Networks, 2009*
2. T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *Proc. of ICLR*, 2017.
3. Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, Philip S. Yu, “A Comprehensive Survey on Graph Neural Networks”, arXiv:1901.00596
4. D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, “Scene graph generation by iterative message passing,” in *Proc. of CVPR*, vol. 2, 2017
5. J. Johnson, A. Gupta, and L. Fei-Fei, “Image generation from scene graphs,” in *Proc. of CVPR*, 2018
6. X. Wang, Y. Ye, and A. Gupta, “Zero-shot recognition via semantic embeddings and knowledge graphs,” in *CVPR 2018*