Graph Neural Network for High-dimensional Data

Rahul Bhadani

Traditionally, Artificial Neural Networks (ANN) have employed linear relationships in the given dataset of interest to find patterns, do model-fitting make predictions, and perform statistical inferences. However, ANN works with datasets such as matrices, vectors, and linear data structure and are not suited for datasets with a hierarchical structure such as trees, heaps, graphs, hypergraphs, hash tables, etc.

For a hierarchical data structure such as a graph, graph neural networks (GNN) are well suited to perform learning. GNN has been applied to a wide variety of applications such as anomaly detection, clustering, classification, and link prediction in the domain of finances, cyber-security, bioinformatics, and transportation to name a few. In this article, we will explore the basics of GNNs and how they can be used to analyze high-dimensional data. In fact, in any domain where there is a possibility of a network (e.g. transportation network, biological network, social network, etc.), GNN can be utilized.

1 Graphs

A graph G is a data structure described by a set of edges E and vertices V (also known as nodes): G = (V, E). They can be directed or undirected. A graph is often represented by an Adjacency matrix, A. If a graph has N nodes, then A has a dimension of (N×N). Alternatively, a graph can be represented by sparse edge representation where we only need the list of edges, each edge is represented by a tuple (u, v) where u and v denote vertices or nodes. We sometimes need 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×F).

2 Why graphs are difficult to analyze?

Graphs can be difficult to analyze because they have many complex and interrelated relationships between the nodes and edges.

- 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.
- A graph does not have a fixed form.
- They are difficult to visualize when we have a large graph.
- Graphs can have different types of nodes and edges, with different attributes and properties. This heterogeneity can make it difficult to analyze the graph as a whole.

3 Then why we should use graph data structure?

Even though we clearly see the challenges of analyzing graphs, they are still amazing for understanding and analyzing data since they can capture complex and non-linear relationships that are usually not represented by linear data structures.

- 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.
- Graphs can solve more complex problems by simplifying the problems into simpler representations or transforming the problems into representations from different perspectives.

4 Working Principle of GNN

The underlying idea behind GNNs is that the characteristics of a node are closely tied to those of its neighboring nodes and the connections between them. To understand this, consider that if we were to remove the neighboring nodes and connections from a single node, it would lose all of its context and meaning. Thus, it is through the relationships and connections with its neighbors that a node's identity is determined.

5 Embeddings in Graph Learning

Since graphs are unstructured and any representation of a graph can lead to very high dimensional matrices, it is important to calculate low-dimensional representations. Such representations are called embeddings. Since usually a graph contains thousands of nodes, we are interested in calculating low-dimensional vector representations of nodes. Some methods to calculate node embeddings are (i) Message Passing; (ii) Random Projection; (iii) Node2Vec. Out of them, the most popular one is Message Passing.

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 the node embedding. One task of all GNNs is to determine the "node embedding" of each node, by looking at the information on its neighboring nodes.

6 Node Classification Problem

In the node classification problem setup, each node v is characterized by its feature x_v and associated with a ground-truth label t_v. Given a partially labeled graph G, the goal is to leverage these labeled nodes to predict the labels of the unlabeled. It learns to represent each node with a d-dimensional vector (state) h_v which contains the information of its neighborhood. Specifically,

$$h_v = f(x_v, x_{co}[v], h_{ne[v]}, h_{ne[v]}) \tag{1}$$

where $x_{co}[v]$ denotes the features of the edges connecting with v, $h_{ne[v]}$ denotes the embedding of the neighboring nodes of v, and $x_{ne[v]}$ denotes the features of the neighboring nodes of v.

Since we are seeking a unique solution for h_v, we can apply Banach's fixed point theorem and rewrite the above equation as an iterative update process. Such operation is often referred to as message passing or neighborhood aggregation. Ultimately, it leads to

$$H^{t+1} = F(H^t, X) \tag{2}$$

where H and X denote the concatenation of all the h and x, respectively. The output of the GNN is computed by passing the state h_v as well as the feature x_v to an output function g.

$$o_v = g(h_v, x_v) \tag{3}$$

Both f and g here can be interpreted as feed-forward fully-connected Neural Networks. The L1 loss can be straightforwardly formulated as the following:

$$loss = \sum_{i=1}^{p} (t_i - o_i) \tag{4}$$

which can be optimized via gradient descent.

7 What other task we can perform with GNN?

Apart from node classification, we can perform several other tasks with GNN such as:

- Link Prediction: the task is to understand the relationship between entities in graphs and predict if two entities have a connection in between.
- Graph Classification: the task is to classify the whole graph into different categories. It is similar to image classification but the target changes into the graph domain.

8 Use cases of GNN

GNNs have been used in various domains for achieving novel tasks as well as outperforming state-of-the-art. A few cases are below:

- GNNs have been used to learn physical models of complex systems of interacting particles
- In recommender systems, the interactions between users and items can be represented as a bipartite graph and the goal is to predict new potential edges (i.e., which items could a user be interested in), which can be achieved with GNNs
- Prediction of protein-protein and protein-ligand interactions
- Prediction of quantum molecular properties
- Generation of novel compounds and drugs

9 References

- https://arxiv.org/pdf/1911.02928.pdf
- https://www.mdpi.com/2078-2489/13/8/396
- https://web.archive.org/web/20220525200720/https://persagen.com/files/misc/scarselli2009graph.pdf
- https://arxiv.org/pdf/1812.08434
- https://arxiv.org/pdf/1611.08097.pdf