Graph Neural Network and its Applications

Juncheng Dong
Computer Science
Duke University
juncheng.dong@duke.edu

Abstract

Graph data and deep learning on graph have been of increasing interest and excitement. As a result, graph neural network(GNN), the study of how to apply deep learning on graph data, has been a popular topic. This paper is a review of graph neural network. Unlike most recent reviews on graph neural network, this review focus on applications of graph neural network in different areas including computer vision, chemistry, social science etc. The review intends to construct an overview starting from graph-related tasks and proposes two different perspectives to understand GNN. Then it will discuss about how researchers in different areas transforms their area-specific problems to graph-related problem, then apply graph neuron network to solve them.

1 Introduction

Deep learning has achieved great success in many tasks of numerous areas such as object detection and video processing in computer vision, speech recognition and translation in nature language processing, recommendation based on items' features etc. While all these achievements are exciting, most data studied by deep learning is in Euclidean space where the data is regular and grid-like. We can consider image as vectors on grid and text as vectors on a line.

However, non-euclidean data such as graphs also contains valuable information. For deep learning on other non-Euclidean data such as mesh and manifold, [31] provides a comprehensive review. Graph is the natural data structure to represent objects and their relationships thus it can represent complex systems. For example, in social science, social network can be represented as a graph of users and their following relationship; publications can be represented as papers and their citation relationships; users' purchase history can be represented as a graph of user and item graph. In Chemistry, relationship between proteins (or any entities) can be represented as an interaction graph and all molecules themselves can also be represented as a graph where nodes are atoms and edges are chemical bonds. Graph is also crucial data type in many other areas such as Physics and knowledge representation.

Due to ubiquity and importance of graphs, many researches have studied about how to apply all the deep learning tools which are suitable for Euclidean data on graph which is non-Euclidean. Hence graph neural network is the study of applying neural network and deep learning on graph data. However, adapting deep learning methods to graph data is non-trivial due to complexity of graph data. First, there is no natural ordering of nodes in a graph while in images we have clear idea of relative positions of pixels and in text we have clear idea of previous and next. Second, graph can be irregular which means that the number of neighbors of nodes can be very different. This makes methods like convolution and pooling in Convolutional Neural Network difficult to operate on graph. A more implicit problem is that it's obvious that nodes that are connected together can provide some relational bias to each other, but extracting meaning information from neighbors or from further nodes in graph can be difficult. Many researchers have made great progress on solving these difficulties.

While all these progress deserves detailed discussion, there are already many existing review paper of GNN that focusing on the technical details of various Graph Neural Network. This review paper takes a different perspective. Instead of reviewing different types of GNNs about their similarities and differences, this paper focus on applications of GNNs in different areas including computer vision, natural language processing, chemistry, social science etc. We first introduce notations and common graph-related tasks. Then it gives a short review of history and taxonomy of GNN. Two perspectives about GNN will be proposed to help readers to understand the function of GNN conceptually. Then some benchmark data sets will be presented. Following the example, the paper focus on how GNNs are used in different areas, specifically how the researchers of different areas transformed their tasks to graph-related tasks and then use GNNs to solve them.

2 Notation and Input to GNN

In common settings of GNN, we are given a graph represented as $G = (V, E)$ where V is the set of nodes and E is the set of edges. We use v_i denote single node in V and e_{ij} denote edge connecting v_i and v_j. In this paper, we assume all the graphs are undirected. But all the methods in this paper can be easily adapted to directed graph. The graph structure is usually represented as an adjacency matrix $\mathbf{A}$ where $A_{ij} = 1$ if there is an edge between v_i and v_j, otherwise $A_{ij} = 0$. Very often we also have node attributes (i.e. for each node v_i, we would have a feature vector x_i). So we have, as input to GNN, adjacency matrix $\mathbf{A}$ denoting graph structure and feature matrix $\mathbf{X} \in R^{n\times d}$ where n is the number of nodes in graph and d is the number of node features. In all literature about GNN, the node features are assumed to be given. So node feature collection is a design choice for models. For example, in social networks, node features can be user profile information, extracted visual feature of users' avatars, vector representation of users' text or combination of any of the previous features as long as the features are homogeneous for all nodes in the graph. An interesting future direction of GNN is to develop GNN with trainable node feature extraction so that the node feature collection can be closely related to the task and trainable end-to-end. Other than adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$, another concept is also important. A normalized graph Laplacian matrix L is defined as $L = I_n - D^{-1/2}AD^{-1/2}$ where I_n is identity matrix of size n and D is a diagonal degree matrix where $D_{ii} = \sum_j A_{ij}$. L is a real valued, symmetric, and positive semidefinite matrix. Hence L can be diagnolized as $L = U\Lambda U^T$ where Λ is a diagonal matrix and columns of U form an orthonormal space. This orthonormal space U will be important in following section about a kind of GNN called spectral-based convolutional graph neuron network.

3 Graph-related tasks

In this section, we introduce common graph-related tasks. The reason for this section is that while GNNs have wide applications in many different areas, these applications will first transform various of machine learning and deep learning tasks in different areas into graph-related tasks in order to employ GNNs. Our next sections will provide a detailed discussion of how researches of different areas transform their tasks into graph-related tasks. In this section, we will provide a taxonomy of graph-related tasks. Based on the objects of interest, we can categorize graph-related tasks into broadly two categories- node-level tasks and graph-level tasks.

3.1 Node-level tasks

Node Classification/Regression Node classification and node regression may be the most common graph-related tasks. In node classification/regression, we are given a graph in which some nodes are labelled and we want to predict the label of unlabelled nodes

Link prediction In link prediction problem, we are given a graph and we want to predict whether there is an edge between two node v_i and v_j. Of course in more complicated scenarios, other than existence of edges, we want to also predict the weight and the type of edges.

Node embedding/representation learning Node embedding is very closely related to GNN. Node embedding aims to find a vector representation of nodes that can contain both the nodes' own features as well as topological information. Node embedding was considered as an unsupervised learning task. But in GNN, node embedding is usually part of the model and trained end-to-end with supervised-learning task

3.2 Graph-level tasks

Graph Classification In graph classification problem, we are given a set of graphs and usually features for each node in each graph. Some of the graphs are labelled and we want to predict the labels of the unlabelled graphs.

Graph Generation Graph generation is another very important graph-related task. When given a graph or a set of graphs, we want to generate realistic or similar graphs with the same characteristics to the given graph

Graph embedding/representation learning Similar to node embedding, graph embedding aims to find a vector representation for graph that can contain both all of its nodes' information as well as the topological information in graph. Graph embedding is also closely related with GNN because GNN usually includes graph embedding as part of the model when performing graph classification or graph generation tasks.

4 History and Taxonomy of GNN

There are many taxonomies of GNN. GNN can be broadly categorized into *recurrent graph neural network*, *convolutional graph neural network* and *graph autoencoder*.

4.1 Recurrent Graph Neuron Network

The first elaborated work about deep learning on graph, graph neural network (GNN*), was proposed in 2009 by Scarselli et al [32]. It was before the popular age of deep learning and is considered as the first graph neural network. it iteratively update the representation for each node in the graph until the representations converge to a fixed value. To achieve this, it employs a regularized recurrent neural network. Some modifications were made to improve its training efficiency [35]. However, the constraint that the node representations converge to fixed value limits the performance. Gated Graph Neural Network [34] relaxes the constraint and employs a gate recurrent unit [33] as update function. This line of work uses recurrent neural network to update the node representation (i.e they use the same set of parameters to iteratively update node representation). For this reason, we call them recurrent graph neural networks (RecGNNs) [36].

4.2 Convolutional Graph Neuron Network

Another line of work originated from convolutional neural network. They adapts convolution on Euclidean data such as graph to non-Euclidean data such as graph. For this reason, we call them convolutional graph neural networks (ConvGNNs) [36]. ConvGNNs can be divided into two categories - spectral-based and spatial based. Interestingly, spectral-based models and spatial-based models are connected by arguably the most popular GNN in recent years - Graph Convolutional Network [39].

Spectral-based Spectral-based ConvGNN was first proposed by Bruna et al [37] in 2013. It's closely related to spectral graph theory and graph signal processing. Spectral-based ConvGNNs refines convolution on graph as performing graph Fourier transform to project node features (also called graph signals in graph signal processing) to the graph space by choosing a filter g. Different filters g will create different convolution. In notation section, we have introduced normalized graph Laplacian matrix L and the orthonormal space U we have by diagonalizing L as $L = U\Lambda U^T$. Suppose the node feature matrix is $X \in R^{n \times d}$. So for each node, we have a feature vector of size d. Then a graph signal is a column in feature matrix X. In other words, a graph signal x is vector of size n that has 1 dimension of node features. Then spectral graph convolution on graph signal x with filter g is defined as

$$x \star g = U(U^T x \odot U^T g)$$

where U is the orthonormal space of graph Laplacian matrix L, g is the filter for convolution and $\odot$ is element-wise multiplication. While this has a solid theoritical function in graph signal processing, eigen-decomposition which is required for calculation of U takes $O(n^3)$ which takes too long for large graph. On the other hand, any slightest change of graph, even only adding one node or adding an edge, will change U drastically. Hence U need to be calculated every time the graph changes and the learnt filter g cannot be applied to convolution on other graph. To overcome these problems, later works on spectral-based graph convolution focus on approximation for filter g without calculation of U. For example, Chebyshev Spectral CNN (ChebNet) approximates the filter g by Chebyshev polynomials [3]. To elaborate more but without too much details, precision of approximation is controlled by a hyperparameter K. When $K = 1$ and parameter sharing is enforced, ChebNet becomes Graph Convolutional Network (GCN) which is the bridge between spectral-based and spatial-based ConvGNNs.

Spatial-based Spatial-based ConvGNNs define convolution very differently. The first spatial-based ConvGNNs was proposed concurrent to GNN* by Micheli [38] in 2009. Convolution in spatial-based ConvGNNs can be viewed as nodes communicating with its neighbors and updating its own presentation based on the message received from neighbors. In 2016, Kipf and Welling [39] proposed Graph Convolutional Network (GCN) which closes the gap between spectral-based method and spatial-based method by showing that spatial-based method can be considered as a first order approximation of spectral method, thus preserving locality. Based on different methods of passing messages, many different spatial-based ConvGNNs have been proposed. To name a few, Diffusion Convolutional Neuron Network (DCNN) models message passing between nodes as a diffusion process of information on the graph. It assumes that amount of information send to neighbors is proportional to the transition probability from current node to its neighbor [2]. In GCN, nodes take message from its neighbors then normalize the summed message by dividing it with the number of neighbors. In GraphSage, nodes samples fixed amount of neighbors to receive message.

4.3 General Framework for Message Passing GNN

Most of message passing neural networks, both RecGNN and ConvGNN, follow some conventions. Graph convolution by message passing can be considered as each node in graph has a hidden state and the hidden states are updated simultaneously for all nodes together for many iterations. Here we present a general framework for GNN that performs graph convolution by message passing. This framework includes four steps for message passing: initialization, message collecting, message merging, state update. In the following paragraphs, we use x_v to denote for node feature of node v and h_v^i to denote hidden state of node v at iteration i.

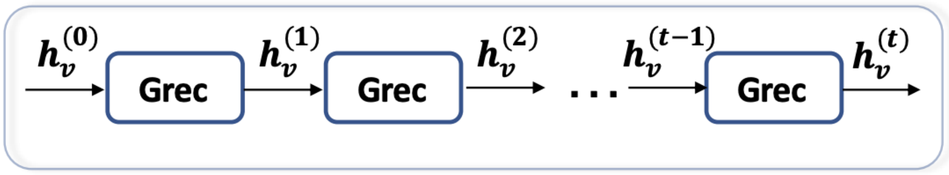
Initialization The hidden states for all nodes are usually set to their feature vectors at first. Hence $h_v^0 = x_v$ for all v.

Message Collecting In message collecting step, all nodes in graph send its message, usually just the hidden states, to their neighbors. Reciprocally all nodes will receive sent messages from its neighbors. Then an aggregating function, usually just a sum function, will aggregate all the received messages. But for generality, we represent this message collecting step with two function, message function $m(.)$ that takes hidden state of a node then generates the message to be sent and aggregating function $agg(.)$ that takes in all the received message from a node's neighbor. Hence for any node v in graph, the aggregated information from neighbors for v is $a_v = agg(m(u)/u \in N(v))$

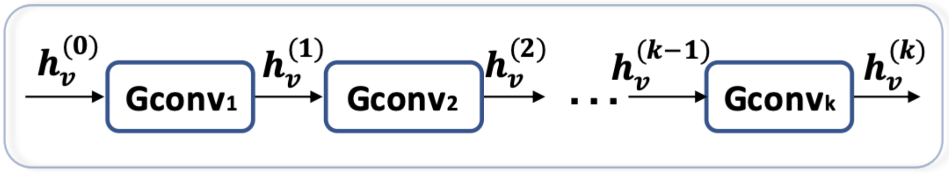
Message Merging In message merging step, the node will take the aggregated information a_v and combine it with its own hidden state h_v to update the hidden state. So we have merging function q that takes a_v and h_v.

Update In the last step, an non-linear update will be performed on each node. We denote the non-linear update function as f. f can a simple ReLU function or can be more complicated function approximators such as multi-layer perceptron. In the last step, we have $h_v^{t+1} = f(q(h_v^t, a_v)))$

This update process is repeated for K times where K is a hyper-parameter we can choose. At the end, we will have the final representation for nodes in graph. The biggest difference between RecGNN and ConvGNN is that RecGNN uses the same set of parameters in all iterations while ConvGNN uses different sets of parameters for each of the iteration. In the visual below, we can see that RecGNN uses the same layers while ConvGNN uses different layers for graph convolution.



(a) Recurrent Graph Neural Networks (RecGNNs). RecGNNs use the same graph recurrent layer (Grec) in updating node representations.



(b) Convolutional Graph Neural Networks (ConvGNNs). ConvGNNs use a different graph convolutional layer (Gconv) in updating node representations.

4.4 Graph Autoencoder

The most notable difference between RecGNNs/ConvGNNs and graph autodencoder is that graph autoencoders are employed in unsupervised learning tasks such representation learning for nodes or graphs and graph generation tasks. Some famous graph autoencoder include GAE which utilized GCN as its encoder and use inner product as decoder to reconstruct whether there is an edge between two nodes [1]. Similar to relationship between autoencoder and variational autoencoder, graph variational autoencoder tries to learn distribution of representation for graph. For example, Variational Graph Autoencoder (VGAE) assumes a Gaussian prior for representation of all the nodes in the graph and uses GCN as encoder as well as inner product as decoder for existance of edge between two nodes [1].

5 Two perspectives of GNN

Although there are many various kind of GNNs for various of graph-related tasks, they have something in common. And there are two perspectives to look at as well as understand GNN.

5.1 GNN as representation learning function

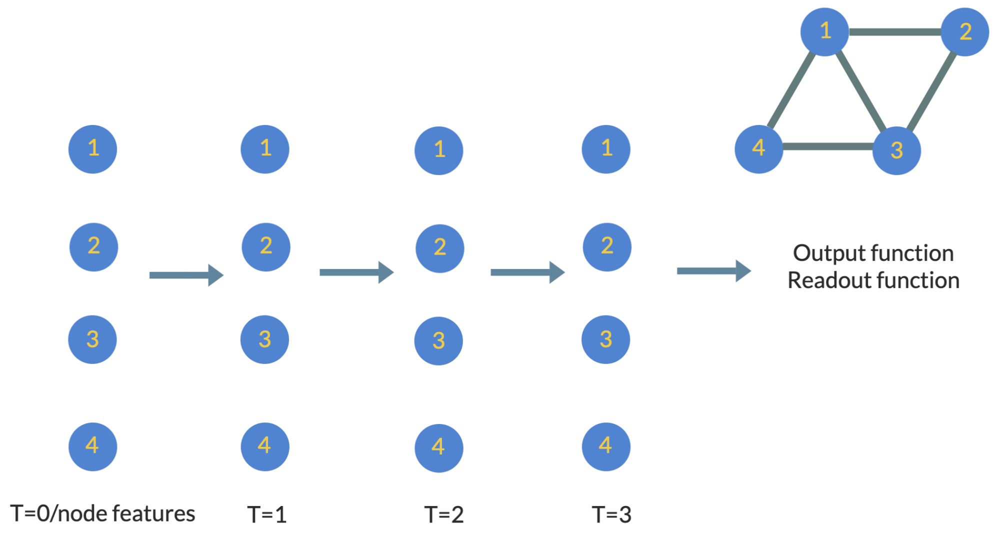
The value of graph data lies in the relationships between nodes. When it's impossible to collect all possible information, connection between nodes will tell us more information. For any node, if the labels of its neighbors are known, then its neighbors' information should be valuable to know its own label. This is the idea of relational bias, to borrow information from other objects in the system. This also gives us a new to define clossness of two data points. Usually closeness between data points are measured by the Euclidean distance between their feature vectors. But for nodes in the same graph, the closeness can be measured by their distance on graph, this solves the problem of insufficient data collection - two very different data points have similar features because only similar features are collected. So no matter how different are the structures between GNNs, they all need to find representation for nodes in graph by combining nodes' initial features with information from other nodes in the graph as well as topological information. This representation learning is done by layers of graph convolution, either by RecGNN or by ConvGNN. Hence variety of GNNs comes from difference in design of graph convolution, the process of extracting information from the graph. But all GNNs, through different graph convolutions, will always have a final representation for all the nodes in the graph. Then later tasks are performed on these extracted final representation. If the graph-related task is node-level, GNN will have an output function on nodes. If the graph-related task is graph-level, GNN will first have a readout function to compute representation for the whole graph, then an output function on graphs. Hence the common components of GNNs are as following:

Node-level tasks: representation learning on nodes+ output function on nodes

Graph-level tasks: representation learning on nodes + readout function on final representation of nodes + output function on graph

5.2 Feed-Forward Layers of Node Representations

Because almost in all GNNs, the final representations of nodes are computed by iteratively updating each node's hidden state. GNN can be considered as layers of nodes'representations. In layer 0, we start with nodes' features.In layer 1, we take in layer 0's information and compute the updated hidden states for all the nodes. This process is repeated until the last graph convolution. Then the output function or readout function will operate on the final representation of nodes. Following is such a computation graph, in each layer there are n neurons where n is the number of graphs. Understanding such computation graphs will give readers a clear idea of what GNNs are doing.



6 Benchmark Data sets

In machine learning and deep learning, data is, if not more, as important as the learning algorithms. There are some well-organized and well-annotated graph data sets that is crucial for comparing performance of various GNNs as well as testing new structure of GNN. These benchmark data sets can be divided into four categories: Citation Network, Biochemical Network, Social Network and Traffic Network.

Citation Network A citation network consists of papers and publications. Two publications are connected if one cites the other. Popular citation networks include Cora, a network of 2708 machine learning publications grouped into 7 classes, Citeseer, a network of 3327 publications grouped into 6 classes and DBLP, a network of millions of publications. Each publication in these citation networks has hand-engineered features.

Biochemical Network Biochemical data sets contain graph representation of molecules and chemical compounds as well as their properties. For example, NCI-1/9 data sets contain 4110/4127 chemical compounds, each labelled as whether it is active to hinder the growth of human cancer cells. The QM9 data set contains 133885 molecules,each of them is annotated with 13 physical properties. Another type of biochemical network is protein-protein interaction network which contains nodes as proteins and nodes are connected if they interact. The PPI data set contains 24 such interaction network.

Social Network Social network may be the most famous type of network. Social network is formed by users and their interactions. The BlogDatalog data set is a network consisting users as nodes and two users are connected if they have social relationship such as following. The Reddit data set is a network of more than 11 millions posts and two posts are connected if they are commented by the same user.

Traffic Network Traffic data set is also very common. METR-LA contains four months of traffic data collected by 207 sensors on the highways of Los Angeles County.

7 Taxonomy of Applications of GNN

All applications of GNN can be naturally divided into two scenarios: **structural** and **non-structural**. Structural scenario is when the graph used in the application is explicitly defined such as social network or molecule network. Non-structural scenario is when the graph used in the application is implicit and need to be first constructed. Applications in structural scenarios are often easier to transform into graph-related tasks while applications in non-structural scenarios requires expertise knowledge for constructing the required graph, and performance of graph-related task closely depends on construction of the implicit graph. In following sections, we will start from non-structural scenarios including computer vision, natural language processing to structural scenarios including traffic prediction, physical system, chemistry, recommendation system, social science and combinatorial optimization. It is crucial to point out that in non-structural scenarios, there is no convention for how to construct the graph, hence, the success of GNN mainly depends on whether the graph is constructed in a reasonable way guided by expertise knowledge. So construction of graph is as important as GNN itself for applications in non-structural scenarios. While in the structure scenarios, applications can follow many conventions for performing graph related tasks and there are many already existing GNN frameworks ready to use. With all these said, GNN has been widely applied in many different areas. Applications reviewed in following sections are far from exhaustive list. Readers are encouraged to find out how GNN can be adapted to their own area of expertise and help with area-specific problems.

8 Applications of GNN in Computer Vision

Scene Graph Generation While deep learning has achieved many success in tasks such as object classification and object detection, representing an image as a collection of objects isn't helpful for high-intelligence tasks such as interpreting the image content. Scene graph, a graph representing objects and their relationships, is a structured representation data that can be used to perform many high-intelligence tasks such as visual question answering [40] in which both the graph and the question are encoded as graphs, then the task is transformed to graph classification problem (whether this graph should be classified as "yes" or "no" according to the question). In order to perform scene graph generation, the application will first perform object detection, by regional proposal network, to find possible objects in the graph. Then it will construct a fully connected graph of these objects [41]. Then to construct the correct scene graph, the application is performing node and link prediction about the class of nodes (what object it is) and the class of edges (what relationship between two objects). To solve the redundancy of fully connected graph, [42] proposes relation proposal network to first prune unlikely edges (relationships) from the graph between scene graph generation to improve efficiency while [43] proposes a different method based on sub-graph extraction to further increase efficiency.

Image generation from scene graph We can also inverse the scene graph generation task to generating an image given scene graph [44]. Scene graph can also be the bridge between text and image because any descriptive text can be also construct as scene graph. Then we can generate image from text description by first converting the text to scene graph.

Few-Shot/Zero-shot Learning Few-shot learning is another challenging problem in computer vision. Many creative applications with GNN have been proposed. They can be divided into two categories. One is using knowledge graph of objects to help with the task while the other directly construct a graph of images and perform classification on the graph. Humans learn by understanding the relationship between objects and thus can transfer knowledge they have on one object to another object under reasonable assumptions. Knowledge graph can be the key for transfer learning on machines. [30] constructs a knowledge graph of objects to build classifier with trained classifiers. A regular object classification model consists of convolution layers which extracts features then a fully-connected layer for classification on extracted features. [30] proposes to use one layers logistic regression as the fully-connected layer. Then it constructs a knowledge graph in which each node stands for a kind of object and two nodes are connected if they are close in semantic meaning (node representing horse should be connected with node representing zebra). The initial node features are the word embedding for that object. And labels on nodes are the parameters of trained logistic regression in object classification model. Then the problem of few-shot learning is transformed to node regression problem where we want to find the labels (parameters for trained logistic regression) for those unlabelled objects in the knowledge graph. After the labels are predicted, we can use the already trained convolution layer and the predicted parameters for logistic regression as the object classification model for that object. Knowledge Graph is also helpful to solve unseen label problem. [29] uses knowledge graph to solve multi label zero-shot learning problem. Another way to use graph is to build a graph of images so the similarity between images can be utilized for few-shot learning [28]. It first construct a weighted fully connected graph of images in data set. Then new representations of images are calculated by graph convolution. These new representations are used for classification of unseen images.

Human Object Interaction Human Object Interaction (HOI) can also be transformed to a graph-related task. Given a fully-connected HOI graph. [27] uses iterative message passing to update hidden state of nodes and calculate how likely an edge should exist between two nodes by a link function that takes in hidden representation of two nodes and returns likelihood of edge between these two nodes. At the end of iterations, node classification is performed on the final hidden state of nodes to predict the action of human in this graph.

Semantic Segmentation GNNs are also widely used in semantic segmentation tasks. There are two different approaches for 2d(RGB) image and 3d(RGBD) image. For 2d image, an image is divided to many equal size super-pixels [26]. These super-pixels form a graph when two adjacent super-pixels are connected to each other. Hence these super-pixels form a grid-like graph. Convolutions are done on the original image to extract features for each of these super-pixels. Then graph convolution is performed on the super-pixel graph. At the end of graph convolution, final representations for super-pixels are used for node classification (what does this super-pixel represent in semantic segmentation task). However, these super-pixels are difficult to define on 3d images. Hence another method needs to be proposed. [25] constructs k-nearest-neighbor graph for the image based geometric position of pixels and uses the RGB feature as node features. Then graph convolution is performed on this knn graph and node classification is performed on final representation of nodes.

Point Cloud Point cloud is the data collected by LiDAR devices and is used widely in many scenarios like robotics and automatic driving. Two most common tasks performed on point cloud are point cloud classification which is classification the whole point cloud and point cloud segmentation which is to find the semantic meaning of each sub-part of point cloud. Point cloud problem is solved in a similar way to 3d images by first constructing a k-nearest-neighbor graph then perform graph classification or node classification on the knn graph. However, the number of points in point cloud is much more than the number of pixels in an image. Thus, [24] first constructs super-node graph based on the knn graph generated from the point cloud, then performs graph related tasks on the super-node graph. Most graphs constructed from image or point cloud is static which means the structure of the graph remains the same. This assumption may limit the performance of GNN. [23] proposes a framework that iteratively update not only hidden state of nodes in graph but also hidden state of edges as well as structure of the graph. The hidden state of edges depends on the hidden state of nodes and vice versa. Then at the end of each iteration, the structure of graph is updated based on the hidden state of the nodes. It also designs a framework that can solve point cloud classification and segmentation at the same time.

9 Applications of GNN in Natural Language Processing

Document Classification as node classification Suppose that documents form a graph or a network, then we can utilize the network structure to improve document classification. For example, a citation network is a network of publications and two publications are connected if one cites the other. If some of the documents in this network is labelled, then document classification problem becomes node classification problem. The steps are similar as other graph-related tasks. First graph convolution is performed on the network then the final representation is used for node classification. With such network, classification can be done without extracting information from the content [39].

Document Classification as graph classification Text classification problem is that given some text, we want to predict its class or label. One typical example is given a short sentence, predict whether it's positive or negative. One can also consider document classification problem as text classification since document is just very long text. To apply GNN to do text classification, a graph of words in text need to be constructed. The most common graph is co-occurrence graph in which two words are connected if they appear in the same sentence or in a window of any size [21]. Then document classification can be transformed to graph classification task on the word graph. Node classification and graph classification can be combined for document classification. [20] constructs heterogeneous graph contains two types of nodes. One type stands for words and the other stands for documents. Word nodes are connected to document nodes if the word appears in that document. And word nodes are connected with each other by the same standard of co-occurrence graph. In this way, we are performing node classification task again, but we can use graph convolution to extract information from content in the document to further improve the model.

Sentence Labelling Sentence labelling problem is that given a sentence, we want to predict a class or label for each word in the sentence. One way to construct a graph for sentence is simply representing the sentence as a chain where each word is only connected to previous and next words. Such graph is very simple, thus cannot capture all the information in the sentence. [19] represents the sentence the graph as syntactic dependency tree. It also uses gate on each edge in the syntactic dependency tree to regulate how much information should be passed along this edge when graph convolution is performed on this syntactic dependency tree. After the graph representation for a sentence is constructed, sentence labelling problem becomes node classification problem on graph and we can further apply a read-out function to find representation for the whole graph to perform text classification.

Neural Machine Translation While neural machine translation is typically solved by recurrent neural network, GNN can also be useful in this task. By constructing syntactic dependency tree for a sentence, GNN can utilized the syntax infomration directly. [18] applies GNN on syntactic dependency tree to perform syntax-aware neural machine translation task. [17] not only uses syatax information but also semantic information by utilizing predicate-argument structure of sentence.

AMR-to-Sentence The graph-to-sentence problem is that given a semantic graph of abstract words (Abstract Meaning Representation), we want to generate sentences with the same meaning of AMR. Traditional way to solve this problem is by sequence-to-sequence model which first represents the semantic graph as a sequence data. However, this will cause loss of information. With GNN, we can use GNN as the encoder and perform graph convolution to encode the representation of semantic graph. Two approaches have been proposed. [15] is based on graph LSTM while [16] is based on GRU.

10 Applications of GNN in Traffic Prediction

Traffic prediction is another popular application with GNN. Roads in a city can be considered as a network in which the nodes are cross-sections and the edges are roads. By constructing roads as network, we can perform traffic prediction on the graph. A traffic prediction problem can be considered as node classification on time series data where each node has a sequence of labels representing history traffic speed. Given past traffic data on one node, we want to predict this node's future traffic. Only using history data on one node, we can do time-series prediction. However, clearly roads' traffic is related to other adjacent road, so we should be able use neighbor nodes' information to help predict future traffic. In order to utilize neighbor node's information, graph convolution is first performed at each time step on data at that time step, then the convoluted data is passed into recurrent neural network to do future traffic prediction [14]. By combining relational information with history data, traffic prediction can achieve state-of-art performance [13].

11 Applications of GNN in Physical System

Due to graph's ability to represent objects and their relationships, graph is the most suitable representation of complex system such as physical system. By modeling objects as nodes in graph, objects' relationships as edges in graph, we can perform inference on object state as node classification, inference on relationship as link prediction and inference on state of the whole system as graph classification. To simulate and learn about physical systems, [12] proposes Interaction Network which represents any physical system as objects and their interactions. More importantly, it defines a node state update function and an interaction update function following physical dynamics. The node and interaction update function will iteratively update the states of the graph simulating working of a physical system. Hence Interaction Network can also be considered as a physical simulation engine. Any agents communication or interaction system can also be represented with a graph. Hence inference agents behaviors or system states can be transformed to node classification and graph classification problem. For example, [11] uses GNN to perform multi-agent predictive modeling. What is interesting is that physical system has a long history of being represented by graph and those deep learning methods applied on the physical system graph follows the GNN framework while the authors in physical system didn't mention their model's closeness to GNN. From this maybe we can claim that GNN is the intuitive and natural deep learning model to perform on graph data. [10] proposed Graph Network framework which can include most GNNs. Graph Network consists of three modules - node states update module, edge states update module and graph state update module. In each iteration, all the edge states are first updated, then the nodes states are updated based on the new edge states, then the new edge states and new node states are aggregated to update the graph state. Graph Network was first developed as a physical system and a physical engine. But it turns out that most GNNs can be considered as an implementation of Graph Network.

12 Applications of GNN in Chemistry and Biology

Graph structure is widely used in Chemistry and Biology. Specifically, graph is nature representation of molecule. On a molecule graph, nodes are atoms and edges are chemical bonds. We can use feature for atoms as node features. Before GNN and other study on applying deep learning directly on graph data, molecules were studied with a string representation of the molecule graph called SMILES. However, encoding the graph into text will inevitably lose information of the data. Hence, performing deep learning directly is a better solution than to first encode graph data to text sequence then apply recurrent neural nets on it.

Molecule Fingerprints Molecules consisting of different atoms and different chemical bonds have various physical properties such as stableness and various biochemical properties such as toxic or not. Molecule fingerprints is a vector that can describe molecule's properties. This can be considered as graph embedding problem.

Molecule Properties With molecule fingerprints, we can predict properties of this molecule. Given molecule graph, molecule properties prediction can be transformed to graph classification problem in which we want to predict labels (properties) for different graphs (molecules) or node classification problem if we are given a network of molecules where some of the molecules' properties are known. This is usually done by first graph convolution, then use a readout function to find the representation for molecule graph, then use the representation of molecule to perform prediction [9]. This framework is usually trainable end-to-end. With high accuracy, this application can speed up greatly all the lab procedures for biology and chemistry.

Protein Interaction Proteins and other compounds, organisms interact with each other very actively in our body. This complex interaction system can be modeled by a protein-protein interaction network. In this network, two proteins are connected if they interact with each other. Prediction about interaction between two proteins is transformed to link prediction problem on this interaction network. First graph convolution is performed to find embedding of proteins in the network. Then these embedding is the input to an output function that takes embedding of two proteins and predicts the likelihood of edge between them [39]. This whole framework, from graph convolution to output function for link prediction can be trained from end-to-end so that we can extract information valuable to link prediction by graph convolution. With high accuracy, this application provides a fast method to predict behaviors of new drugs and medicine, thus shortening the time required for lab procedures which can take months or even years.

Molecule Generation Molecule generation can be considered as a graph generation problem in which we want to generate graphs similar to the given graph according to various similarity requirements depending on subjects. This application is widely used to synthesize compounds, especially molecules with similar properties to known drugs [8]. There are two approaches for generating graphs. One is to generate the graph at once by generating the most likely adjacency matrix[8]. This can be thought as global method. The other approach is to break the graph generation in local choices, specifically, when to where to add nodes and edges [6]. We can not only generate similar molecules but also molecules with designed objects such as stableness. Graph generation can be considered as a sequences of actions to add nodes and edges. Then the generative model can be considered as a player in this graph generation process with a reward, often non-differential, to maximize at the end of process. Hence, [7] incorporates reinforcement learning with GNN to generate goal-directed molecule graphs. The whole framework is trained end-to-end by policy gradient. As a very complicated task, graph generation is often an intersection between GNN and other deep learning methods. For example, [8] adopts generative adversial network to generate more realistic graphs while [7] adopts reinforcement learning to handle non-differential and complicated objectives.

13 Applications of GNN in Recommendation System

Old recommendation system was based on single user's shopping history. The recommendation system would recommend items that is the most similar to what the user has liked or purchased. It can be considered a typical classification problem given features of an item, we want to predict that item's class (whether it is liked by users or not). However, with increasing amount of users' action history data. New generation recommendation systems are mostly based on graphs (graph-based recommendation is also called collaborative filtering in other literature). The intuition behind this is simple: if user A and user B like the same product, then it is also very likely for user A to like other items liked by user B. These relational data can help improve recommendation. With users' history data, we can construct a network consisting of items and users. It's a bipartite graph where users are not connected with users and items are not connected with items. Then whether user will any item is transformed to a link prediction between user nodes and item nodes. To predict links between nodes, graph convolution is first performed on the user-item graph. Then the output function takes final representation for a user and an item predict whether there should exist an edge between them. The output function and graph convolution function is trained end-to-end [4]. One difficulty with recommendation system is the number of items and users in the graph. It can contain billions of items and tens of millions of users. In order to perform graph convolution on large scale network, [4] also proposes to use sampling methods like random walk during graph convolution and other methods like map-reduce to better adapt to network size.

14 Applications of GNN in Combinatorial Optimization

There are many famous NP-hard combinatorial optimization defined on graph such as travelling salesman problem and vertex cover problem. Due to equality of all NP-complete problems, if we can solve one of the NP-hard problem, we solve all of the NP-complete problems. Most of the combinatorial optimization problems on graph can be transformed to a graph classification/regression problem. For example, given vertex cover problem, we can transform it to the graph classification problem that given this graph, we want to classify this graph to one of two classes 0 or 1. 0 means there is no vertex cover of size k in this graph and 1 means there exists vertex cover of size k. We can also form the vertex cover problem as a regression problem where we directly regression on the size of minimum vertex cover. However, these are two problems. The first problem is that we cannot have large amount of data for training because finding the correct answer on graph is NP-hard. If we can collect large amount of data for training, then we should already have an efficient algorithm for these NP-hard problems. The second problem is that even we can know the yes/no answer or the minimization value, we cannot have the exact solution for that minimization value. For example, even if GNN correctly finds the size of minimum vertex cover, it cannot return the set of vertices that form this minimum vertex cover. On the other hand, there is only very a few work on limitation of representation power of GNN. Despite these difficulties, some great achievements have been made in this area. For example, [5] tries to solve quadratic assignment problems which is the problem to measure the similarity of two graphs.

15 Discussion of Power and Limits of GNN

There are only a few dedicated works on the theoretical aspects of GNNs. Here we briefly discuss about merits of GNN as well as its drawback.

Obviously GNN enables us the perform deep learning on graph data directly without pre-processing such as converting graph data to sequence data. This improves performance of all graph-related tasks and since graph data is ubiquitous, by GNN we can extract valuable information from graph data which we cannot before GNN. By either using explicit graph or constructing graph, we can utilized relational bias through GNN which decreases the amount the data required for learning. For example, in node classificaiton task, even if only 5% of the total nodes are labelled, GNN can still give amazing results [39].

However, GNN is not efficient comparing to normal multi-layer perceptrons (MLP). If we ignore all the data structure and only use node features, we are back to the traditional deep learning with MLP which is significantly faster but performs worse since it is not using any relational information. Hence this is a trade-off that we have to consider. While GNN seems promising on many graph-related tasks, GNN on complicated graph such as dynamic graph on which the graph structure changes over time and heterogeneous graph on which there are many types of nodes and edges remains an open and challenging problem.

16 Conclusion

As an uprising area of deep learning, GNN has received more attention in recent years and number of applications involving GNN has increased sharply. While GNN has promising behaviors, many problems remains for us to solve. We have reviewed graph neuron network from a different perspective. Instead of focusing on technical details, we try to construct an overview of graph-related tasks, what GNN is doing conceptually and how researcheres in different areas transform their area-specific problem to graph-related tasks and use GNN to solve them. In this paper, we have introduced common graph-related tasks, benchmark data sets, taxonomy of both GNN and applications of GNN as well as detailed walk-through of how researches in computer vision, nature language processing, social science, physics, chemistry apply GNN to solve their area-specific problems. Finally we have discussed briefly about merits and limits of GNN.

Acknowledgement

This review paper is initiated as a research project for deep learning course as well as a tutorial for readers who are not familiar with GNN. It is also intended to be the first part of a master thesis. The author appreciates all the help and patience from his instructor of course as well as advisor Prof. Vahid Tarokh and Dr. Mohammadreza Soltani.

References

[1] T. N. Kipf and M. Welling, *Variational graph auto-encoders* NIPS Workshop on Bayesian Deep Learning, 2016.

[2] J. Atwood and D. Towsley, *Diffusion-convolutional neural networks* in Proc. of NIPS, 2016, pp. 1993–2001.

[3] M. Defferrard, X. Bresson, and P. Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering* in Proc. of NIPS, 2016, pp. 3844–3852.

[4] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, *Graph convolutional neural networks for web-scale recommender systems* in Proc. of KDD. ACM, 2018, pp. 974–983.

[5] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna, *Revised note on learning quadratic assignment with graph neural networks* in IEEE DSW 2018. IEEE, 2018, pp. 1–5.

[6] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, *Graphrnn: Generating realistic graphs with deep auto-regressive models* in ICML 2018, 2018, pp. 5694–5703.

[7] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, *Graph convolutional policy network for goal-directed molecular graph generation* in Proc. of NeurIPS, 2018.

[8] N. De Cao and T. Kipf, *MolGAN: An implicit generative model for small molecular graphs* ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models, 2018.

[9] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, *Neural message passing for quantum chemistry* in Proc. of ICML, 2017, pp. 1263–1272.

[10] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner et al., *Relational inductive biases, deep learning, and graph networks* arXiv preprint arXiv:1806.01261, 2018.

[11] Y. Hoshen, *Vain: Attentional multi-agent predictive modeling* in NIPS 2017, 2017, pp. 2701–2711.

[12] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende et al., *Interaction networks for learning about objects, relations and physics* NIPS 2016, 2016, pp. 4502–4510.

[13] B. Yu, H. Yin, and Z. Zhu, *Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting* in Proc.of IJCAI, 2018, pp. 3634–3640.

[14]] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, *Gaan: Gated attention networks for learning on large and spatiotemporal graphs* in Proc. of UAI, 2018.

[15] L. Song, Y. Zhang, Z. Wang, and D. Gildea, *A graph-to-sequence model for amr-to-text generation* in Proc. of ACL, 2018.

[16] D. Beck, G. Haffari, and T. Cohn, *Graph-to-sequence learning using gated graph neural networks* in Proc. of ACL, 2018.

[17] D. Marcheggiani, J. Bastings, and I. Titov, *Exploiting semantics in neural machine translation with graph convolutional networks* in Proc. of NAACL, 2018.

[18] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima'an, *Graph convolutional encoders for syntax-aware neural machine translation* in Proc. of EMNLP, 2017, pp. 1957–1967.

[19] D. Marcheggiani and I. Titov, *Encoding sentences with graph convolutional networks for semantic role labeling* in Proc. of EMNLP, 2017, pp. 1506–1515.

[20] L. Yao, C. Mao, and Y. Luo, *Graph convolutional networks for text classification* arXiv preprint arXiv:1809.05679, 2018.

[21] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, *Large-scale hierarchical text classification with recursively regularized deep graph-cnn* in WWW 2018, 2018, pp.1063–1072.

[22] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks* in Proc. of ICLR, 2017.

[23] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, *Dynamic graph cnn for learning on point clouds* ACM Transactions on Graphics (TOG), 2019.

[24] L. Landrieu and M. Simonovsky, *Large-scale point cloud semantic segmentation with superpoint graphs* in Proc. of CVPR, 2018.

[25] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, *3d graph neural networks for rgbd semantic segmentation* in CVPR 2017, 2017, pp. 5199–5208.

[26] X. Liang, X. Shen, J. Feng, L. Lin, and S. Yan, *Semantic object parsing with graph lstm* ECCV 2016, pp. 125–143, 2016.

[27] S. Qi, W. Wang, B. Jia, J. Shen, and S.-C. Zhu *Learning human object interactions by graph parsing neural networks* arXiv preprint arXiv:1808.07962, 2018.

[28] V. Garcia and J. Bruna, *Few-shot learning with graph neural networks* arXiv preprint arXiv:1711.04043, 2017.

[29] C. Lee, W. Fang, C. Yeh, and Y. F. Wang, *Multi-label zero-shot learning with structured knowledge graphs* in Proceedings of CVPR, 2018, pp. 1576–1585.

[30] X. Wang, Y. Ye, and A. Gupta *Zero-shot recognition via semantic embeddings and knowledge graphs* in CVPR 2018, 2018, pp.6857–6866.

[31] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, *Geometric deep learning: going beyond euclidean data*. IEEE SPM 2017, vol. 34, no. 4, pp. 18–42, 2017.

[32] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, *The graph neural network model*. IEEE Transactions on Neural Networks, vol. 20, no. 1, pp. 61–80, 2009.

[33] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*. in Proc. of EMNLP, 2014, pp. 1724–1734.

[34] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, *Gated graph sequence neural networks* in Proc. of ICLR, 2015.

[35] C. Gallicchio and A. Micheli, *Graph echo state networks* in IJCNN. IEEE, 2010, pp. 1–8.

[36] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, *A comprehensive survey on graph neural networks* arXiv preprint arXiv:1901.00596, 2019.

[37] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, *Spectral networks and locally connected networks on graphs* in Proc. of ICLR, 2014.

[38] A. Micheli, *Neural network for graphs: A contextual constructive approach* IEEE Transactions on Neural Networks, vol. 20, no. 3, pp.498–511, 2009.

[39] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks* in Proc. of ICLR, 2017.

[40] D. Teney, L. Liu, and A. V. Den Hengel, *Graph-structured representations for visual question answering* in Proceedings of CVPR, 2017, pp. 3233–3241.

[41] 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.

[42] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, *Graph r-cnn for scene graph generation* in Proc. of ECCV. Springer, 2018, pp. 690–706

[43] Y. Li, W. Ouyang, B. Zhou, J. Shi, C. Zhang, and X. Wang, *Factorizable net: an efficient subgraph-based framework for scene graph generation* in Proc. of ECCV. Springer, 2018, pp. 346–363.

[44] J. Johnson, A. Gupta, and L. Fei-Fei, *Image generation from scene graphs* in Proc. of CVPR, 2018.