Survey on Graph Neural Networks

Angela Kormushoska

***Abstract*—With the rise of Machine Learning tasks, different models have been researched and implemented to deal with Euclidean data. Euclidean data can be represented as a grid graph, such as images, videos and texts. Images and videos have fixed-sized grid graphs, and texts are represented as a line graph, because they are sequences. However, graphs are more complex, they don't have a fixed size and have unordered nodes with different numbers of neighbors. This makes graphs hard to analyze with the models that already exist for euclidean data, such as CNNs. GNNs are neural networks that solve this problem, they work with non-euclidean data or graphs. They have many applications, some of which are social networks, computer vision, natural language processing, chemistry and recommender systems. This paper will provide an overview of GNNs, how they work and differ from other networks, their different categorizations and frameworks, as well as some different usages in various fields and some problems they have.**

***Index Terms*—Graphs, CNN, GNN, message passing, node embeddings, RGNNs, CGNNs, GAEs, STGNNs, graph classification, node classification.**

I. Introduction

Deep Learning (DL) has seen many improvements, like ability to capture hidden patterns and influence statistical properties of data, since the introduction of recurrent neural networks (RNNs) and convolutional neural networks (CNNs), especially with machine learning (ML) tasks, such as computer vision, natural language processing and speech recognition [3]. Other than that, these ML models have been used in agriculture, commerce, finance, medicine and so on [3]. Euclidean data is data that can be represented as a grid graph. Non-euclidean data on the other side is concentrated on graphs, which have no fixed size and no fixed number of nodes. These models have certain limitations and since they can only be applied to euclidian data, they cannot work with graphs. That is where GNNs take over, since they work with non-euclidian data, where convolutions as in CNNs are difficult to implement. On Fig. 3 it is represented how traditional models and GNNs differ. GNNs successfully do the tasks which CNNs cannot and further expand the research about themselves [3]. GNNs are present in our everyday lives and have been used and proposed for various tasks, such as node classification, link prediction and graph clustering, which will be discussed in Section II and VII [13]. Furthermore, they are heavily applied in a variety of different branches, such as computer vision, recommender systems, chemistry, traffic, chemistry, medicine, pharmacy and more [22], [3].

II. Graphs

A. Formal Definitions

Graph Neural Networks (GNNs) consist of graphs and neural networks. Graphs are used to represent social netwroks or information networks [5]. Formally, in computer science, a Graph G is a data structure, that consists of two components: nodes or vertices (V) and edges (E). This could also be represented as $G = (V, E)$, where E represents the edges between a set of nodes v_1,$v_2 \in V$. An adjacency matrix A, with dimensions $(n \times n)$, where n indicates the number of nodes, is used to represent graphs. If $e_{ij} \in E$, then $A_{ij} = 1$ and if $e_{ij} \notin E$, then accordingly $A_{ij} = 0$ [22].

B. Node features and Edge Attributes

Nodes have features, whose purpose is to describe the nodes, for example describing the user profile with the traits of the user, such as age, occupation, hobbies etc. A feature matrix is a matrix of the features of the dataset that is being used, where each row is a node from the graph, with as many columns as features. If a node has f many features, then the feature matrix X would have the dimension of $(n \times f)$. Nodes in graphs are often associated with feature vectors. The node feature matrix $X \in R^{n \times d}$, where $x_v \in R^d$ represents the feature vector of a node v. In addition to node features, some graphs have edge attributes X^e, in comparison to node attributes X. $X^e \in R^{m \times c}$ represents the edge feature matrix, where $x^e_{u,v} \in R^c$ is the feature vector of an edge (v, u) [1]. It is important to note that node features, along with the graph structure are a part of the learning process of GNNs [5]. A disadvantage of node features in GNNs is that for graphs with no features, GNNs cannot be used on them [5].

C. Example

Fig. 1 shows a graph that represents different nodes, such as Terry, Henry, Kate, Gary etc. and undirected edges between them, which means the "relation" between them is unknown in this example. They could be colleagues, friends, family etc. The feature nodes here are represented as the person's likes. For example, Bill's interests include arts, comics, games and sports. This set of likes will be transformed into a feature vector in the graph, thus helping the GNNs find similar interests in the other candidates in order to find a better match.

III. Convolutional Neural Networks

A. Working Principles of CNNs

Convolutional Neural Networks (CNNs) have been linked to image processing, voice recognition, face detection and video recognition [1]. CNNs have convolutional, pooling and fully-connected layers [5]. If an image is taken as an input, the input layer will take the pixels of the image as values [16]. The convolutional layer will then apply convolution on the input, which is a mathematical operation used to produce a feature

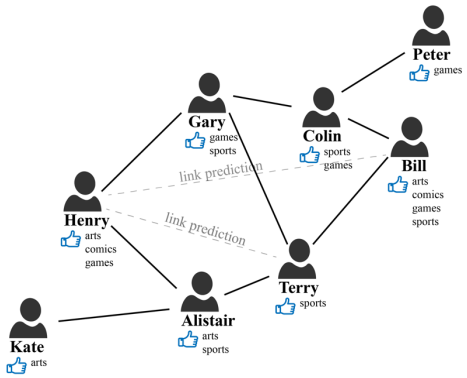


Fig. 1. A social network illustrated as a graph with feature nodes [23]

map. To get this feature map, a filter or a kernel is applied to the input image, which is a $(n \times n)$ matrix, thus multiplying the pixel values of the input, also a $(n \times n)$ matrix, with the values of the filter. The results of this matrix multiplication are written in the feature map. This process continues for every $(n \times n)$ matrix of the input data. The output of this layer will be a competed feature map or the matrix acquired by multiplying the input values with the filter values. This is then passed onto the next convolutional layer as input, where a filter will be used again to determine it's layers output. More complex filters are located deeper in the network, that recognize high-level features of the image, such as faces and other complex patterns. The early convolutional layers detect low-level features, such as edges and shapes. In the pooling layer, the output from the convolutional layer is received and it is compressed or decreased in size, using a pooling filter. There are mainly two different types of pooling, max pooling and average pooling. Max pooling returns the maximum value for each value of the feature map, whereas average pooling returns the average value [16]. The output is now flattened to a column vector and passed down to the fully connected layer. This layer is similar to the traditional neural networks, where every neuron of one layer is connected to every neuron of the next layer. The vector goes through the layer and the image classification takes place.

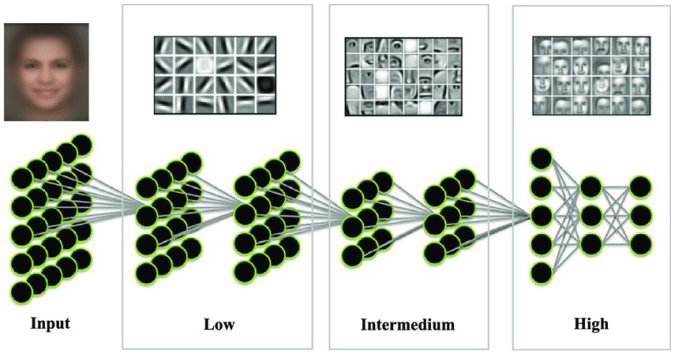


Fig. 2. Face recognition with a Convolutional Neural Network [14].

Fig. 2 shows how the CNN's architecture is implemented for example in face recognition applications. This CNN contains several hidden layers [14]. The input is the raw data, or the picture of a face. In the first convolutional layer, the CNN recognizes edges, lines or dots, some very detailed facial features [14]. In the intermediate layer, some larger features of the face are already recognizable, such us eyes, nose, mouth, brows etc. In the last layers, the network is already able to recognize the whole face along with its features, which helps with face recognition in apps. An important part of the CNNs are also the receptive fields, which place the input neurons in a square to connect to one neuron in the hidden neurons [14].

B. CNNs vs. GNNs

CNNs and GNNs work in a similar fashion. CNNs can actually be seen as GNNs ancestors in the field of graph processing.

CNNs and GNNs are both deep neural networks, that have been important for tasks in deep learning. The advancements of CNNs have resulted in the discovery of GNNs [22]. They share some similarities, but also have some key differences. Their main similarity is that they work in many layers in order to achieve more information about the acquired image or graph, such as any neural network. The earlier layers have less information about the image in CNNs. Similarly, the lower amount of messages aggregated in GNNs, the lower information acquired about the neighborhood and thus the target node.

The main difference between GNNs and CNNs are the structures of the graphs. As shown in Fig. 3, CNNs work with graphs in Eulcidean Space, whereas GNNs work in Non-Euclidean Space [15]. Euclidean data can be anything that can be represented as a grid graph, such as images, videos or a line graph, such as for texts, as seen in Fig. 3 [22]. Furthermore, the nodes of a CNN graph have a fixed size and are ordered. In contrast to that, graphs in GNN are unordered and do not have a fixed size [22]. This includes data such as social networks, user profiles, molecule or compound graphs etc [22].

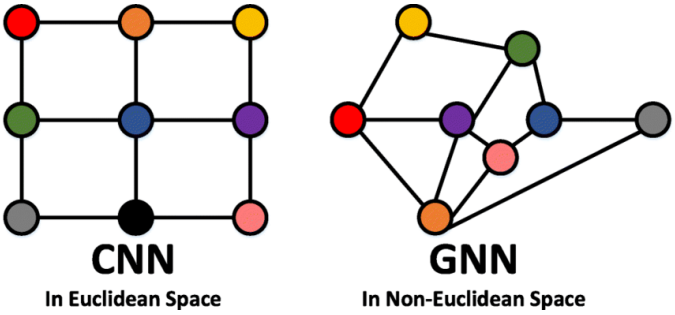


Fig. 3. Differences between CNNs and GNNs [15].

IV. Working Principles of GNNs

A. GNN Architecture

The goal of GNNs is to create representations of nodes and any feature information, where the nodes depend on the graph. GNNs use a form of neural message passing. With it, the nodes of the graph exchange vector messages and then update the neural network parameters [9]. The message-passing framework is important for GNNs. i is the iteration in which the

embedding and function currently are. So with every iteration in the GNN, the function aggregates the set of embeddings of the node's neighbors from it's neighborhood and generates a new message based on the information that it has aggregated. It starts with the first iteration $(i = 1)$, where the node contains information only about itself. In the second iteration $(i = 2)$, the node embedding receives information from its immediate neighbors of the graph (passing of messages between nodes). In the following iteration $(i = 3)$, the node has acquired information from it's neighbor's neighbors. As the iterations continue, every node will aggregate information from the local neighborhood. With the last iteration i, every node and its features have average information about its i-hop neighborhood, which is called average neighborhood information [9]. This cycle goes on until a stable equlibrium is reached and the loss function is low enough. A loss function is the squared difference between expected and predicted value of the GNN and it is used to measure how accurate the predicted value is [9]. This method is really important in classification and labeling problems [9].

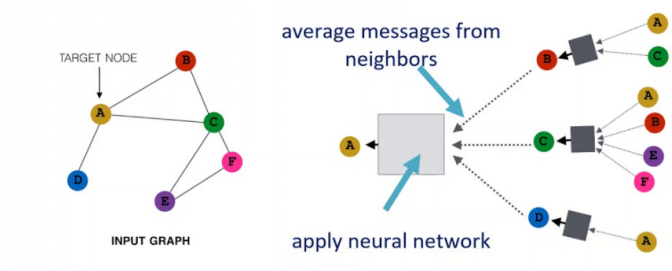


Fig. 4. General Idea of how GNNs work and how average neighborhood information is received [17].

As illustrated on Fig. 4, the GNN works as follows: a target node A from the input graph needs to receive the required information or messages from all the nodes in the graph. In the first iteration, A obtains the information from B, C and D, since these nodes are the direct neighbors of A, as illustrated in the input graph. In the next iteration, A obtains the information from it's neighbor's neighbors. In this case, B obtains information from A and C, which are it's own neighbors. Therefore, A collects this data and the messages have been passed. It works the same way for nodes C and D. In the end, all messages have been passed and the end has been reached, so the iterations are done and the target node A has received the required information. This is an example for a two-layer version of a message-passing protocol. That means there have been 2 layers of aggregation, namely between A and it's neighbor and their respective neighbors. That's how a single node aggregates all the messages from it's neighborhood.

The information that the node embeddings encode can be structural or feature-based [9]. The structural information is extracted by the graph's structure, and the feature information is acquired from the node features [9]. The structural information can be useful for many tasks, such as for molecular graphs analysis, where the information can be used to measure different atom types or benzene rings [9]. This means, that after i iterations of the message passing, the embedding of a node might have (structural) information about all the nodes in the local neighborhood. As for the feature-based information, assuming that the nodes have features, which can be used to start the GNN message passing. That means, that after several iterations, the node embeddings have encoded information about all the features in their local neighborhood.

$$h_u^{k+1} = UPDATE^{(k)}(h_u^{(k)}, AGGREGATE^{(k)}(h_v^{(k)}, \forall v \in N(u)) = UPDATE^{(k)}(h_u^{(k)}, m_{N_u}^{(k)})$$

In this formula the message passing is represented, which includes the update and aggregate functions [9]. $h_u^{(k)}$ represents the hidden embedding, that corresponds to each node $u \in V$ in the graph and is generated during message passing. This hidden embedding is then updated with the update function with the information aggregated the aggregation function from $u's$ neighborhood $N(u)$ [9]. k denotes the iteration in which the embedding and function currently are. So with every iteration k in the GNN, where u is the node and N is the neighborhood, the function aggregates the set of embeddings of u's neighbors from it's neighborhood $N(u)$ and generates a new message $m_{N(u)}^{(k)}$ based on the information that it has aggregated. Then, the update function completes its task by combining the previously acquired message $m_{N(u)}^{(k)}$ with the previous embedding $h_u^{(k-1)}$ of the node u, so it can generate the new (updated) embedding $h_u^{(k)}$. For $k = 0$, the initial embeddings are set to the input features for all the nodes in the input graph, which means $h_u^{(0)} = x_u, \forall u \in V$ [9]. After the k-th iteration, the output of the final layer can be used to define the embeddings for each node $z_u = h_u^{(k)}, \forall u \in V$.

$$h_u^k = \sigma(W_{self}^{(k)} h_u^{(k-1)} + W_n eigh^{(k)} \sum_{v \in N(u)} h_v^{(k-1)} + b^{(k)})$$

This formula is a simplification of the original GNN models initialised by Maerkwirth and Lengauer (2005) and Scarselli et al. (2009) [9], [18]. Here, $W_{self}^{(k)}$,$W_{neigh}^{(k)}$ are trainable matrices with parameters in them and σ is an elementwise non-linearity [9], [18]. $b^{(k)}$ is a bias term that is often omitted, but it can be important to achieve strong performance [9], [18]. The formula represents the sum of the messages from the neighbors. After that, using a linear combination, the neighborhood information is combined with the previous embedding of the node, and is followed by an elementwise non-linearity [9], [7]. Similar to the formula in Fig. 5, the basic GNN can be calculated, using the update and aggregate functions.

As seen in the previous formulas, $m_{N(u)}$ is used to define the aggregated message from u's graph neighborhood [9], [18]. Here, the sum of all aggregated messages from the node embeddings in the graph are represented, which are later updated with every new node embedding and message passing.

B. Example

An example for this would be a dating app, or looking for a match for a partner who has a user profile, such as the other users. All the users in the app have different hobbies, interests, their occupation and age, which could be represented

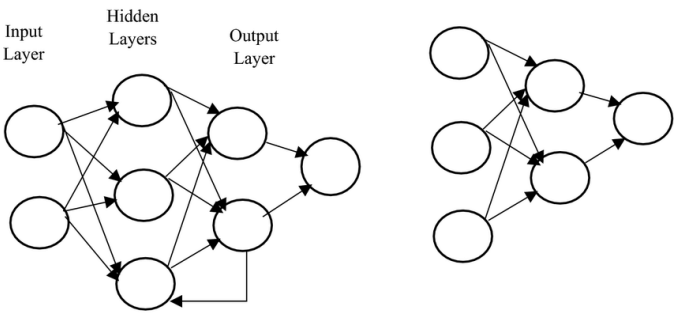
as a feature vector of the features of the node, where the node is the user of the app, as seen in Fig. 1. If we want to find a match for Bill for example, whose interests are stored in the feature vector, we aggregate the information from his neighbors, namely Terry and Colin and pass those messages to Bill. The node embedding is then updated and the process continues, where the information is aggregated via message passing from Alistar, Gary and Peter. Their feature vectors are located in the message and thus Bill receives that information and the node is once again updated and so on. The goal is to map nodes to d-dimensional embeddings such that nodes with similar network neighbors, in our case Bill, are embedded close together, so we can find the perfect match for him based on his different features.

V. Categorization and Training

There are different types of GNNs, that could be categorized in four different types: reccurent graph neural networks (RecGNNs), convolutional graph neural networks (ConvGNNs), graph encoders (GAEs) and spatial-temporal graph neural networks (STGNNs)[22]. In the following these classifications will be briefly explained.

A. Categorization of GNNs

Recurrent graph neural networks (RecGNNs) have been recognized as the first neural networks that were related to GNNs [22]. They apply recurrent neural architectures in order to learn node representations. They make an assumption that messages will be exchanged between nodes and their neighbors until a stable equlibrium has been reached. RecGNNs inspired the making of convolutional graph neural networks (ConvGNNs) [22]. Example on Fig.5.



(a) Recurrent neural network **(b)** Non recurrent neural network

Fig. 5. Difference between a RNN and a non-recurrent neural network [11].

Convolutional graph neural networks (ConvGNNs or CGNNs) are the followers of RecGNNs [22]. They perform convolution on graph data, instead of grid data like CNNs [22]. The main idea is to generate a node's representations or features by aggregating it's own features as well as it's neighbor's features, making up a new node feature that will be represented for the same node. Different from RecGNNs, but similar to CNNs, ConvGNNs achieve high-level node representations by adding more graph convolutional layers [22]. ConvGNNs are known for different and many complex GNN models and have been used for node and graph classification [22]. Illustrative explanation of this architecture on Fig. 6.

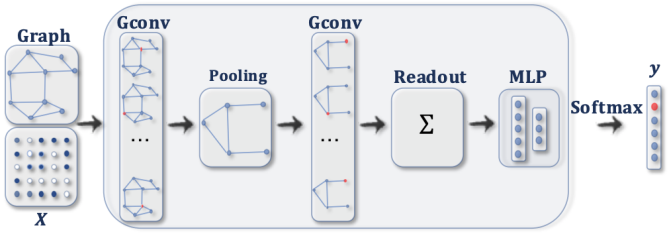


Fig. 6. The architecture of a CGN [22].

Graph autoencoders (GAEs) reconstruct graph data from the messages or encoded information by encoding nodes or graphs into a latent vector space [22]. By using GAEs, we could learn graph generative distributions, as well as network embeddings. When using graph generation, the output may be the whole graph or a step-by-step generating of nodes and edges. As for network embedding, reconstruct graph structural information like for example the graph adjacency matrix, in order to learn latent node representations. This could easily be seen on Fig. 7.

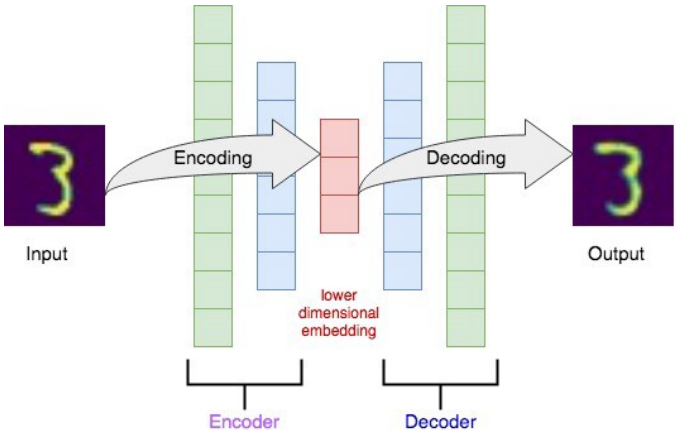


Fig. 7. Architecture of an AE, including the encooder and decoder [10].

Spatial-temporal graph neural networks (STGNNs) use spatial-temporal graphs to learn hidden patterns in the graph. These type of GNNs are important and used in many different areas, such as traffic speed forecasting, human action recognition and driver maneuver anticipation etc [22]. Spatial and temporal dependencies are the key words in STGNNs and the aim is for both of them to be applied at the same time. Different ways to capture or model the spatial and temporal dependencies exist, namely with RNNs for spatial and CNNs for temporal dependency [22], as seen of Fig. 8.

B. History of GNNs

In 1997, Sperduti et al. first used neural networks on acyclic graphs, which started the early studies for GNNs

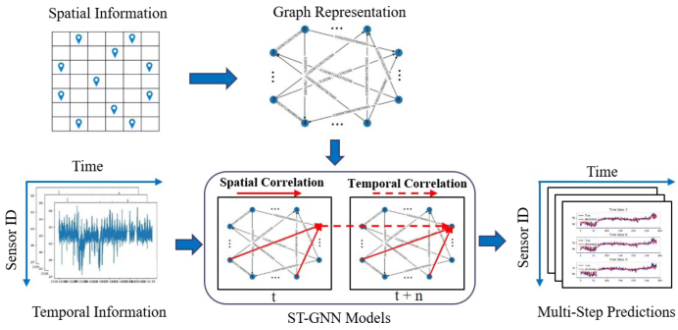


Fig. 8. A spatial-temporal GNN model for traffic prediction, that produces multi-step predictions collecting the temporal and spatial information of a graph [4].

[22]. However, they were formally addressed and introduced by Gori et al. in 2005 and Scarselli et al. in 2008 [13], including Gallicchio et al. in 2010 [22]. These early papers are concentrated on recurrent graph neural networks (RecGNNs). Later on, convolutional graph neural networks (ConvGNNs) are developed, following the success of CNNs. There are two main streams of ConvGNNs: spectral-based and spatial-based approaches. Bruna et al. did the first paper on spectral-based ConvGNNs in 2012 [22], and that shaped the path for many betterments and additions on the spectral-based ConvGGNs [22]. Spatial-based ConvGNNs emerged in 2009, when Micheli et al. inherited ideas of message passing from RecGNNs [22]. In the recent years, GNNs have started to be more researched, thus provoking the development of many new GNNs. These include Graph Autoencoders (GAEs) and Spatial-Temporal Graph Neural Networks (STGNNs), which can be essentially constructed with RecGNNs, ConvGNNs or something similar that is helpful for representing graphs.

C. Outputs

The outputs of the graphs can differ according to the tasks the GNN has been given. On the other side, the inputs of GNNs are the graph structure and node content information. The three different types of outputs can be: node-level, edge-level and graph-level outputs [22]. In the following they will all be briefly discussed.

Node-level outputs, relate to the node regression and node classification tasks of GNNs [22]. Using information propagation or graph convolution, RecGNNs and ConvGNNs can have high-level node representations as outputs, due to the many layers used in order to achieve better precision [22]. In order to achieve this, GNNs have a multi-perceptron and a softmax layer as the output layer, so that node-level tasks could be performed [22].

Edge-level outputs means that the tasks about edge classification and link prediction are of the highest importance. Taking as input the hidden representations of the two nodes of the GNN, a similarity function or neural network can be used to predict labels/connections edge strength [22].

Graph-level output is related to graph classification. GNNs are combined with readout and pooling operations, in order to achieve a compact representation on the graph level [22].

D. Training Frameworks

Depending on the type of classification, there can be also different types of training. Usually, most of the GNNs can learn semi-supervised (for node-level classification) or supervised (for graph-level classification). Depending on the task and available information, some GNNs could also be trained unsupervised.

Supervised learning: This is used for graph-level classification, which aims to predict the class labels for the whole graph [22]. This is executed by using different types of layers, such as graph convolutional layers, graph pooling layers and readout layers. As previously stated, graph convolutional layers help with the output of high-level and precise node representations in node-level outputs, whereas graph pooling layers execute the down-sampling part. Down-sampling means that the input graph is ”downsampled” into a sub-structure [22]. And finally the readout layer collects the node representations of all the graph into a graph representation by summarizing the hidden representations of sub-graphs, thus the graphs are classified. The framework for graph classification is built from a multi-layer perceptron and a softmax layer [22]. ConvGNNs for example use supervised learning with pooling and readout layers for graph classification [22]. A visual illustration can be seen on Fig. 9.

Semi-supervised learning: Used for node-level classification. In a ConvGNN, some nodes in a single network might me labeled, and others not. This leaves space for the ConvGNN to do some semi-supervised learning, by making and learning a robust model, which can successfully classify the unlabeled nodes [12]. Graph classification is used to deal with unlabeled data during training. node classification is used Multi-class classification in an end-to-end framework can be achieved by putting together one after another several convolutional layers and then adding a softmax layer [12].

Unsupervised learning: Used for graph embedding. In unsupervised learning, there are no class labels available in the graphs, so algorithms that could learn and be trained in a different way are needed. Such algorithms can extract edge-to-edge level information in two ways. One way is to use and autoencoder framework, where the graph convolutional layers can be used. In those layers, the graph can be reconstructed using a decoder after the convolutional layers embed the graph into the latent representation [22]. The other way is to learn how to make a difference between positive and negative node pairs [22]. The negative node sampling approach means that the GNN learns to sample a portion of nodes as negative pairs. The positive node approach is to identify existing node pairs in the graph as positive pairs [22].

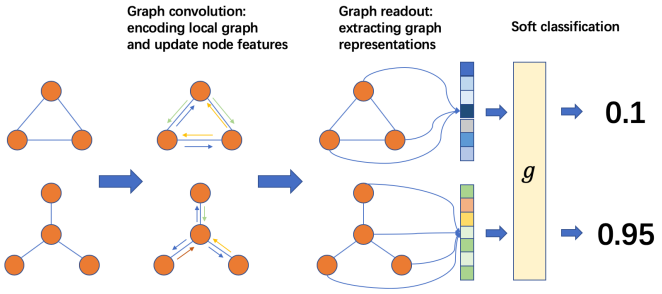


Fig. 9. Graph Classification for supervised learning [19].

VI. Problems

Although GNNs have been praised for doing what CNNs cannot do and have gained popularity in graph structured data, they still have some drawbacks, which will be discussed in this section. Some problems include limitations in the graph structure, noise limitations and graph isomorphism network (GIN) [2] .

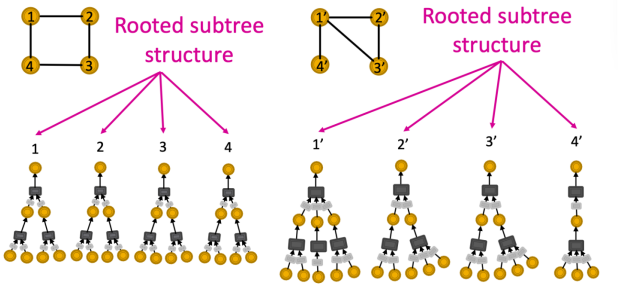


Fig. 10. Graph Structure Limitations of GNNs, different graph structures that some GNNs fail to distinguish [6].

Fig. 10 illustrates how some GNNs, mainly GCNs fail to distinguish graph structures, making it a problem for the graph structure identification [24]. This means that the graphs with the same amount of nodes and with node features that are uniform, but have different edge connections and combinations, will have a different structure - which would not be otherwise recognized in some GNNs [2]. The study by Xu et. al. [2] showed that some popular GNNs such as GCNs cannot differentiate certain and simple graph structures, which would lead to underfitting the training set. This is essentially known as the graph isomorphism problem [8], since the two graphs shown on Fig. 8 are isomorphic because of their structure. Related to this problem are symmetry detecting problem and general combinatorial structures [8]. Because of the wide usage of graphs and GNNs in general, this problem of the isomorphism is really important and still not solved, although there have been some attempts to solve it [8]. There is no polynomial algorithm that solves this case.

GNNs have another drawback, which is the noise vulnerability. This means that GNNs are not robust to noise in the graphs and will change the outcome of their predictions based on a slight noise in the graph [7]. This slight noise may be a node petrubation, edge addition, edge deletion etc. and will make a big difference in the outcome. As seen on Fig. 11

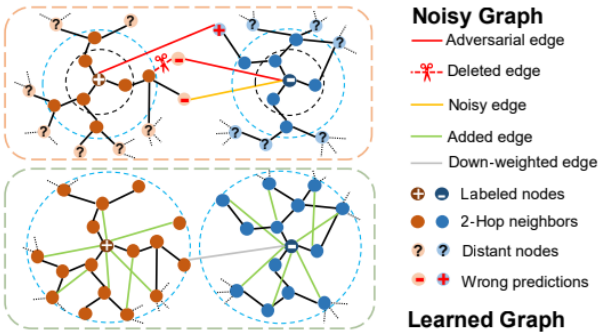


Fig. 11. Noisy graph and learned graph. Removing noise edges and densifying the graph to achieve better performance [6].

for example, attacks add or delete edges to the graph, they are marked with red on the illustration. By doing this, they connect nodes with different features, which causes noise or errors to the node representations [6]. These kind of attacks happen in different areas and applications of GNNs, such as recommender systems, social networks, search engines and so on [7].

VII. Applications

GNNs have been used in many different areas in the everyday world. Since they work with graphs, they additionally work with complex graph related structures and applications using those. They are used in data sets, evaluation and open-source implementations and have of course practical applications [22]. Data sets include biochemical graphs, social networks, citations networks etc [22]. As mentioned before, evaluation and open-source implementations include node classifications and graph application, which are the most common tasks in RNNs and CGNs and can be solved directly by GNNs [22]. In open-source implementations, different base-line experiments in deep learning have been researched. The popular ones are geometric learning library in PyTorch or PyTorch Geometric and the Deep Graph Library (DGL) in PyTorch and MXNet [22]. These provide fast and easy implementation of GNNs in vastly used deep learning platforms. Despite these applications on graphs, some others tasks can be solved too with GNNs, such as node clustering, link prediction and graph partitioning [22]. In the more everyday areas, GNNs are used mostly and especially in computer vision, natural language processing, traffic, chemistry, recommendation systems, link predictions and others [22].

A. Computer Vision

GNNs have helped computer vision to improve in different fields, such as scene graph generation, point clouds classification and action recognition [22]. The implementation of these models brings performance improvement and more explainable decomposition to the models in computer science [19]. CNNs have also been used in computer vision, however they were inefficient with analysis of relationships between visual data [19]. Images are represented as spatial maps, whereas videos are represented as spatio-temporal graphs.

Edges in images or regions can be spatially and semantically dependent, while regions in videos represent relationships between the regions. Edges here are dependent on the regions and form dependant relationships between them and with the nodes, so it is really important to perceive, understand and reason the visual data accordingly and precisely. GNNs help in such tasks in the field of images and videos by extracting patterns from graphs so the right task could be executed [19]. In order for graphs to be analyzed, images and videos and other visions need to be represented as graphs. With images, GNNs are important for object detection and image classification [19]. In the field of videos, GNNs are implemented in video action recognition, temporal action and localization [19]. In other related work, such as cross-media, GNNs help with visual caption, visual question answering and cross-media retrieval [19]. There are other applications, such as few-shot image classification, human-object interaction, semantic segmentation, visual reasoning, question answering [22].

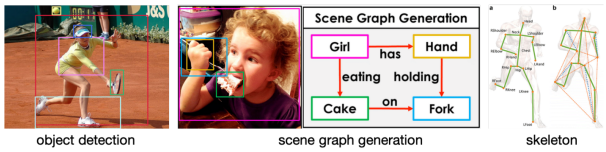


Fig. 12. Different applications of GNNs in Computer Vision [19].

B. Natural Language Processing

Deep Learning has become a big part of Natural Language Processing (NLP). Text is usually represented as a sequence of tokens, but there are some problems in NLP that can only be expressed through graphs, thus they take advantage of the recent developments and improvements of GNNs. For modeling text sequences, RNNs and CGNs have been used [20]. However, for tasks that work with non-Euclidean data, graphs have to be used. Some structures in NLP that could be expressed through graphs include sentence structural information in text sequence and semantic information in sequence data [20]. Thus, this data is able to use the relationships between entity tokes so it can learn a better and more informative representation [20]. Different GNNs have been used for different NLP tasks and have seen a considerable amount of success. Some of these tasks include: sentence classification, semantic role labeling, relation extraction, machine translation, question generation and summarization [20].

As seen on Fig. 13, different applications and tasks of GNNs can be categorized in NLP. This means, GNNs are used for graph constructions tasks, which then include static graph construction (dependency graph, constituency graph, AMR graph etc), dynamic graph construction, such as node embedding-based similarity metric learning and structure-aware similarity metric learning. There is also hybrid graph construction. Different types of graphs can be represented using GNNs for these tasks, which is a part of graph representation learning. There are GNNs for homogenous graphs (static/dynamic graph

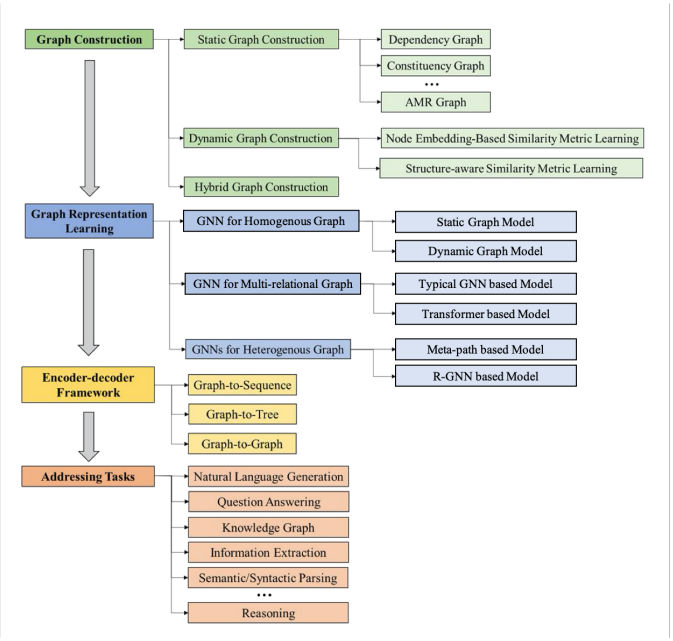


Fig. 13. Taxonomy of NLP tasks for GNNs [20].

model), multi-reltional graphs (typical GNN/transformer based models) and finally heterogeneous graphs (meta-path/RGNN based models). Furthermore, NLP has experienced progress in encoder-decoder frameworks, with tasks such as graph-to-sequence, graph-to-tree and graph-to-graph models. Lately, the addressing tasks or applications, which were mentioned before, take an advantage of GNNs in natural language generation, question answering, knowledge graph, information extraction, semantic/syntactic parsing, reasoning and many more.

C. Recommender Systems

GNNs also play a huge role in recommender systems and have been widely used. Using information propagation on graphs, GNNs can successfully establish correlations between users, items and associated features [21]. GNNs have a lot of advantages in recommender systems, such as: unifying structured data, modeling of high-order connectivity and multiple supervision signals [20]. GNNs also solve the problem of data sparsity in recommender systems [20].

D. Chemistry

Apart from the computer science world, chemistry uses GNNs as well. Molecules or compounds can be represented as graphs, and chemists use GNNs to study their structure [22]. In these graphs which represent molecules, atoms are represented as nodes and edges are the chemical bonds between them. Node/graph classification and graph generation are the three mostly used tasks in chemistry that help with the learning of molecular fingerprints, predicting molecular properties, intefering protein interfaces and synthesizing chemical compounds [22].

E. Others

The power of GNNs is not only recognized by and limited to these branches. Many new domains and tasks need GNNs and use them on an everyday basis [22]. GNNs have been widely used in program verification, program reasoning, social influence prediction, brain networks, event detection, adversarial attacks prevention and many more [22].

VIII. Conclusion

A. Future Work

Although GNNs are gaining popularity and being used in different branches, they still face challenges and have some drawbacks. However, with GNNs being needed more in different industries, the research to improve them and make them more accessible has been increased in recent years. Four future directions for GNNs would be model depth, scalability trade-off, heterogenity and dynamicity [22].

B. Summary

In this paper we provide an overview of GNNs. We further explain how graphs work and how GNNs use these graphs to work with non-euclidian data, in comparison to CNNs, that have been used for euclidian data. Furthermore, we look at the history of GNNs and how they derive, forming the rich taxonomy of GNNs. We additionally introduced some formulas to better explain the working principles of GNNs and stated the different frameworks and training principles of GNNs. Additionally, given the drawbacks, we discuss about the problems that GNNs face, but also shortly discuss the topic of future works. Since they are used in a variety of branches, we list about some of the many applications they have and the impact they have made.

References

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[2] Pranjal Awasthi, Abhimanyu Das, and Sreenivas Gollapudi. Beyond gnns: An efficient architecture for graph problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(6):6019–6027, Jun. 2022.

[3] Jiamin Chen Tengfei Lyu Raeed Al-Sabri Babatounde Moctard Oloulade, Jianliang Gao. Graph neural architecture search: A survey. *Tsinghua Science and Technology*, 27(4):17, 2022.

[4] Nam Bui Khac Hoai, Jiho Cho, and Hongsuk Yi. Spatial-temporal graph neural network for traffic forecasting: An overview and open research issues. *Applied Intelligence*, 52, 02 2022.

[5] Chi Thang Duong, Thanh Dat Hoang, Ha The Hien Dang, Quoc Viet Hung Nguyen, and Karl Aberer. On node features for graph neural networks, 2019.

[6] Hui Liu Suhang Wang Enyan Dai, Wei Jin. Towards robust graph neural networks for noisy graphs with sparse labels. 2022.

[7] James Fox and Sivasankaran Rajamanickam. How robust are graph neural networks to structural noise?, 2019.

[8] Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Commun. ACM*, 63(11):128–134, oct 2020.

[9] William L. Hamilton. *The Graph Neural Network Model*, pages 51–70. Springer International Publishing, Cham, 2020.

[10] Fanghao Han. Tutorial on variational graph auto-encoders. 2019.

[11] Kamaraj Kanagaraj, Arvind Chakrapani, and K. Srihari. A weight optimized artificial neural network for automated software test oracle. *Soft Computing*, 24, 09 2020.

[12] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[13] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6437–6449. PMLR, 18–24 Jul 2021.

[14] Xiang Li, Zhanli Liu, Shaoqing Cui, Chengcheng Luo, Chen-Feng Li, and Zhuo Zhuang. Predicting the effective mechanical property of heterogeneous materials by image based modeling and deep learning. *Computer Methods in Applied Mechanics and Engineering*, 347, 04 2019.

[15] Chia-Hung Lin, Yu-Chien Lin, Yen-Jung Wu, Wei-Ho Chung, and Ta-Sung Lee. A survey on deep learning-based vehicular communication applications. *Journal of Signal Processing Systems*, 93, 04 2021.

[16] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[17] Mohammad Sharique Salman. A classical graph neural network (gnn): Graphs tell stories. 2020.

[18] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[19] DGL Team. 5.4 graph classification. 2018.

[20] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. Graph neural networks for natural language processing: A survey, 2021.

[21] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey, 2020.

[22] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[23] Will Badr Yanwei Cui. Graph-based recommendation system with neptune ml: An illustration on social network link prediction challenges. 2022.

[24] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 11 2019.