

Applications of GNNs

As promised in the intro, I want to finish up with a few applications of GNNs.

Graph-structured data is present everywhere. The problems that GNNs resolve can be classified into these categories:

1. **Node Classification:** the task here is to determine the labeling of samples (represented as nodes) by looking at the labels of their neighbors. Usually, problems of this type are trained in a semi-supervised way, with only a part of the graph being labeled.
2. **Graph Classification:** the task here is to classify the whole graph into different categories. It's like image classification, but the target changes into the graph domain. The applications of graph classification are numerous and range from determining whether a protein is an enzyme or not in bioinformatics, to categorizing documents in NLP, or social network analysis.
3. **Graph visualization:** is an area of mathematics and computer science, at the intersection of geometric graph theory and information visualization. It is concerned with the visual representation of graphs that reveals structures and anomalies that may be present in the data and helps the user to understand the graphs.
4. **Link prediction:** here, the algorithm has to understand the relationship between entities in graphs, and it also tries to predict whether there's a connection between two entities. It's essential in social networks to infer social interactions or to suggest possible friends to the users. It has also been used in recommender system problems and in predicting criminal associations.
5. **Graph clustering:** refers to the *clustering* of data in the form of graphs. There are two distinct forms of clustering performed on graph data. Vertex clustering seeks to cluster the nodes of the graph into groups of densely connected regions based on either edge weights or edge distances. The second form of graph clustering treats the graphs as the objects to be clustered and clusters these objects based on similarity.

Let's go through some applications across domains where GNN can resolve various challenges.

GNNs in computer vision

Using regular CNNs, machines can distinguish and identify objects in images and videos. Although there is still much development needed for machines to have the visual intuition of a human. Yet, GNN architectures can be applied to image classification problems.

One of these problems is scene graph generation, in which the model aims to parse an image into a semantic graph that consists of objects and their semantic relationships. Given an image, scene graph generation models detect and recognize objects and predict semantic relationships between pairs of objects.

However, the number of applications of GNNs in computer vision is still growing. It includes human-object interaction, few-shot image classification, and more.

GNNs in Natural Language Processing

In NLP, we know that the text is a type of sequential data which can be described by an RNN or an LSTM. However, graphs are heavily used in various NLP tasks, due to their naturalness and ease of representation.

Recently, there has been a surge of interest in applying GNNs for a large number of NLP problems like text classification, exploiting semantics in machine translation, user geolocation, relation extraction, or question answering.

We know that every node is an entity and edges describe relations between them. In NLP research, the problem of question answering is not recent. But it was limited by the existing database. Although, with techniques like GraphSage (Hamilton et al.), the methods can be generalized to previously unseen nodes.

GNNs in traffic

Forecasting traffic speed, volume or the density of roads in traffic networks is fundamentally important in a smart transportation system. We can address the traffic prediction problem by using STGNNs.

Considering the traffic network as a spatial-temporal graph where the nodes are sensors installed on roads, the edges are measured by the distance between pairs of nodes, and each node has the average traffic speed within a window as dynamic input features.

GNNs in chemistry

Chemists can use GNNs to research the graph structure of molecules or compounds. In these graphs, nodes are atoms, and edges – chemical bonds.

GNNs in other domains

The application of GNNs is not limited to the above domains and tasks. There have been attempts to apply GNNs to a variety of problems such as program verification, program reasoning, social influence prediction, recommender systems, electrical health records modeling, brain networks, and adversarial attack prevention.

<https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications>

Graph Neural Network and Some of GNN Applications: Everything You Need to Know

The recent success of neural networks has boosted research on pattern recognition and data mining.

Machine learning tasks, like object detection, machine translation, and speech recognition, have been given new life with end-to-end deep learning paradigms like CNN, RNN, or autoencoders.

Deep Learning is good at capturing hidden patterns of Euclidean data (images, text, videos).

But what about applications where data is generated from non-Euclidean domains, represented as graphs with complex relationships and interdependencies between objects?

That's where [Graph Neural Networks](#) (GNN) come in, which we'll explore in this article. We'll start with graph theories and basic definitions, move on to GNN forms and principles, and finish with some applications of GNN.

Table of contents

[What is a Graph?](#)

[Graph Neural Networks](#)

[Graph Convolutional Networks](#)

[Applications of GNN](#)

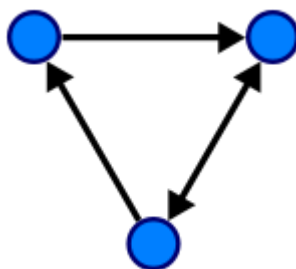
What is a Graph?

The most fundamental part of GNN is a Graph.

In computer science, a **graph** is a data structure consisting of two components: [nodes](#) (vertices) and edges.

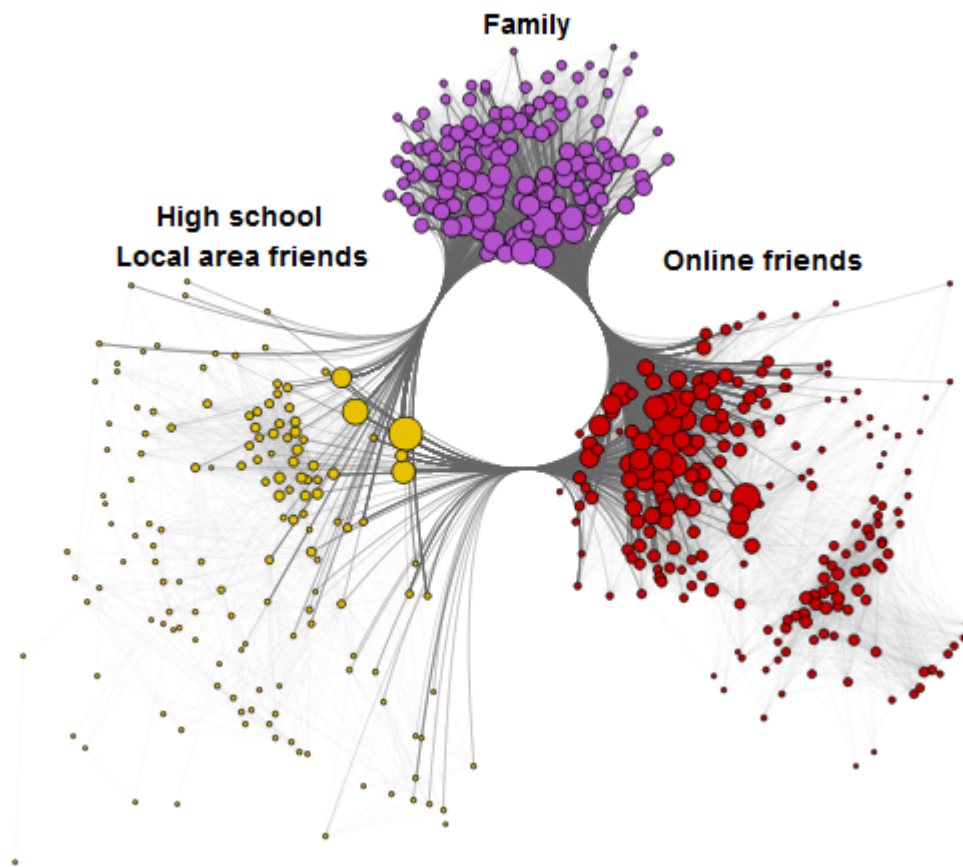
A graph G can be defined as $G = (V, E)$, where V is the set of nodes, and E are the edges between them.

If there are directional dependencies between nodes then edges are directed. If not, edges are undirected.



Directed Graph | [Source](#)

A graph can represent things like social media networks, or molecules. Think of nodes as users, and edges as connections. A social media graph might look like this:



[Source](#)

A graph is often represented by A , an adjacency matrix.

If a graph has n nodes, A has a dimension of $(n \times n)$.

Sometimes the nodes have a set of features (for example, a user profile). If the node has f numbers of features, then the node feature matrix X has a dimension of $(n \times f)$.

READ SOME PAPERS ABOUT GNN

[Top Research Papers from the ECML-PKDD 2020 Conference \(on Graph Neural Networks\)](#)

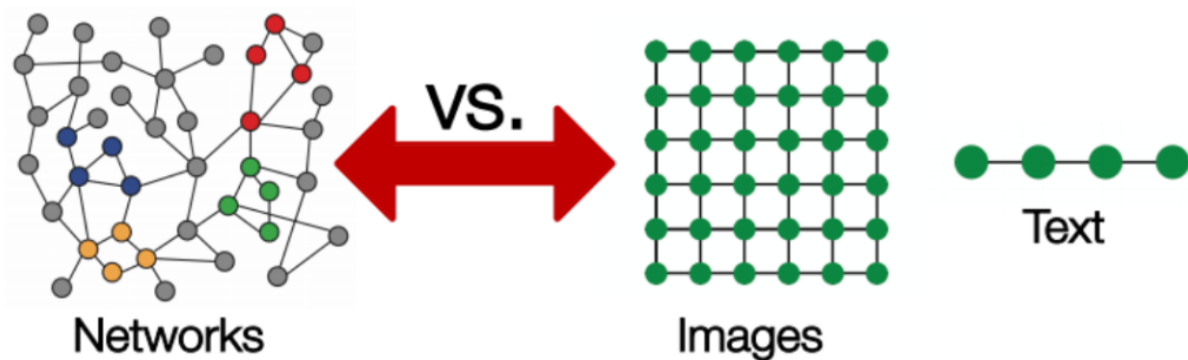
Why is it hard to analyze a graph?

Graph data is so complex that it's created a lot of challenges for existing machine learning algorithms.

The reason is that conventional Machine Learning and Deep Learning tools are specialized in simple data types. Like images with the same structure and size, which we can think of as fixed-size grid graphs. Text and speech are sequences, so we can think of them as line graphs.

But there are more complex graphs, without a fixed form, with a variable size of unordered nodes, where nodes can have different amounts of neighbors.

It also doesn't help that existing machine learning algorithms have a core assumption that instances are independent of each other. This is false for graph data, because each node is related to others by links of various types.



[Source](#)

Graph Neural Network

Graph Neural Networks (GNNs) are a class of deep learning methods designed to perform inference on data described by graphs.

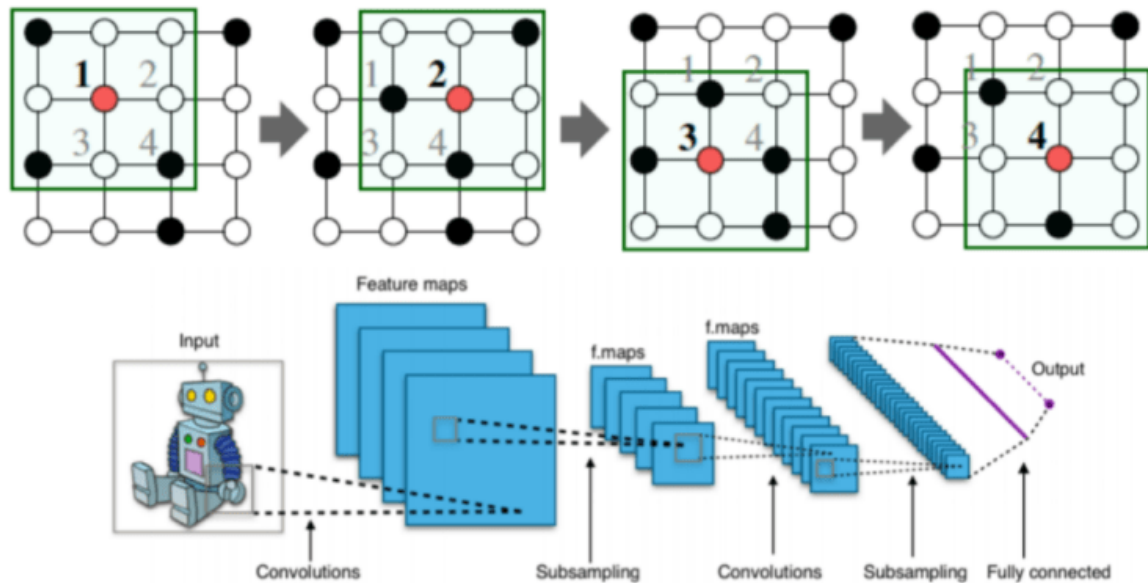
GNNs are neural networks that can be directly applied to graphs, and provide an easy way to do node-level, edge-level, and graph-level prediction tasks.

GNNs can do what Convolutional Neural Networks (CNNs) failed to do.

Why do Convolutional Neural Networks (CNNs) fail on graphs?

CNNs can be used to make machines visualize things, and perform tasks like image classification, image recognition, or object detection. This is where CNNs are the most popular.

The core concept behind CNNs introduces hidden convolution and pooling layers to identify spatially localized features via a set of receptive fields in kernel form.



CNN on an image / [Source](#)

How does convolution operate on images that are regular grids? We slide the convolutional operator window across a two-dimensional image, and we compute some function over that sliding window. Then, we pass it through many layers.

Our goal is to generalize the notion of convolution beyond these simple two-dimensional lattices.

The insight allowing us to reach our goal is that **convolution takes a little sub-patch of the image (a little rectangular part of the image), applies a function to it, and produces a new part (a new pixel).**

What happens is that the center node of that center pixel aggregates information from its neighbors, as well as from itself, to produce a new value.

It's very difficult to perform CNN on graphs because of the arbitrary size of the graph, and the complex topology, which means there is no spatial locality.

There's also unfixed node ordering. If we first labeled the nodes A, B, C, D, E, and the second time we labeled them B, D, A, E, C, then the inputs of the matrix in the network will change. Graphs are invariant to node ordering, so we want to get the same result regardless of how we order the nodes.

Basics of Deep Learning for graphs

In [graph theory](#), we implement the concept of Node Embedding. It means mapping nodes to a d- dimensional embedding space (low dimensional space rather than the actual dimension of the graph), so that similar nodes in the graph are embedded close to each other.

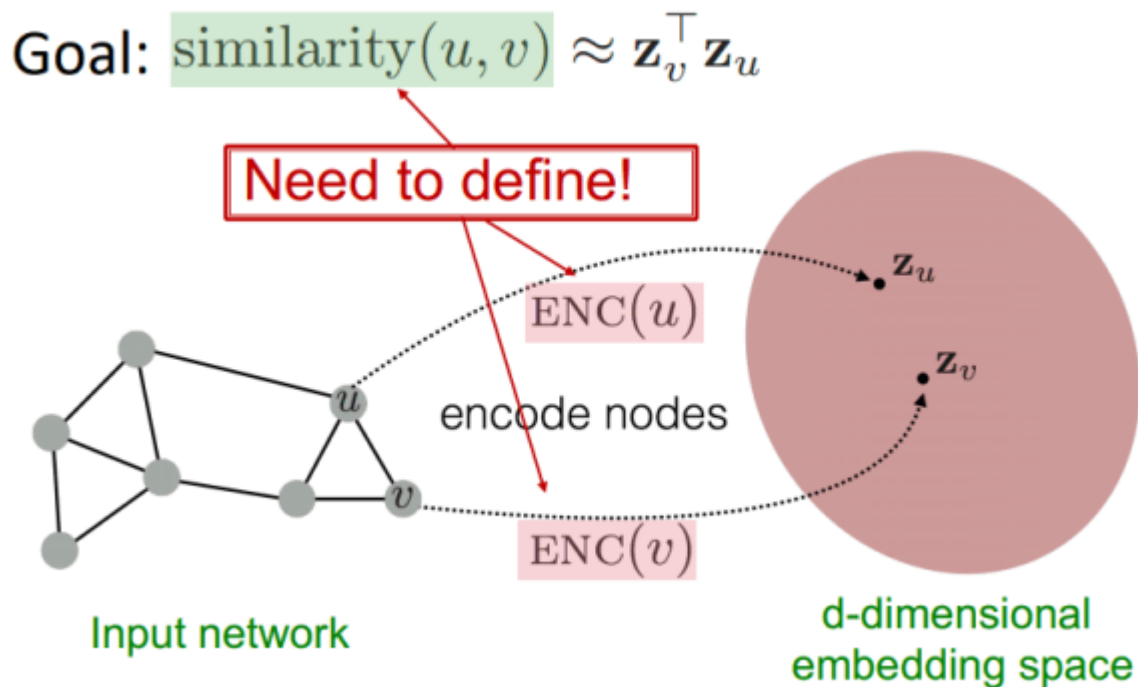
Our goal is to map nodes so that similarity in the embedding space approximates similarity in the network.

Let's define u and v as two nodes in a graph.

x_u and x_v are two feature vectors.

Now we'll define the encoder function $Enc(u)$ and $Enc(v)$, which convert the feature vectors to z_u and z_v .

Note: the similarity function could be Euclidean distance.



[Source](#)

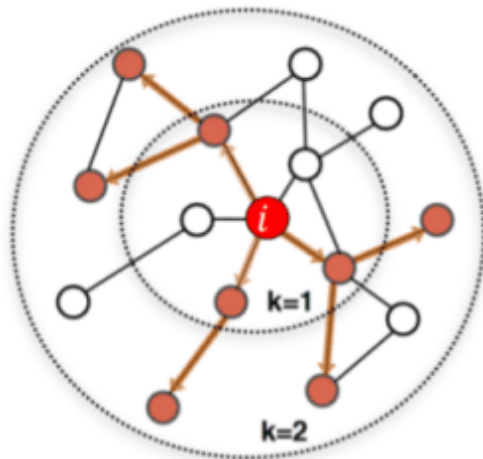
So the challenge now is how to come up with the encoder function?

The encoder function should be able to perform :

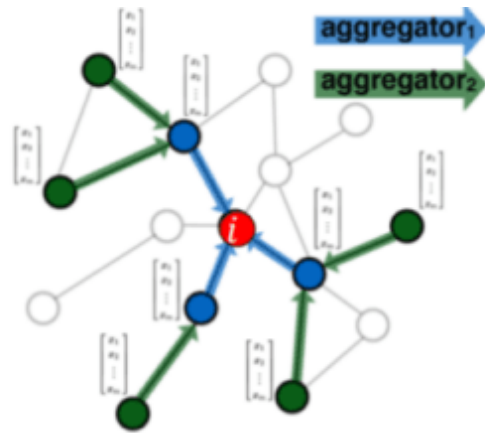
- Locality (local network neighborhoods)
- Aggregate information
- Stacking multiple layers (computation)

Locality information can be achieved by using a computational graph. As shown in the graph below, i is the red node where we see how this node is connected to its neighbors and those neighbors' neighbors. We'll see all the possible connections, and form a computation graph.

By doing this, we're capturing the structure, and also borrowing feature information at the same time.



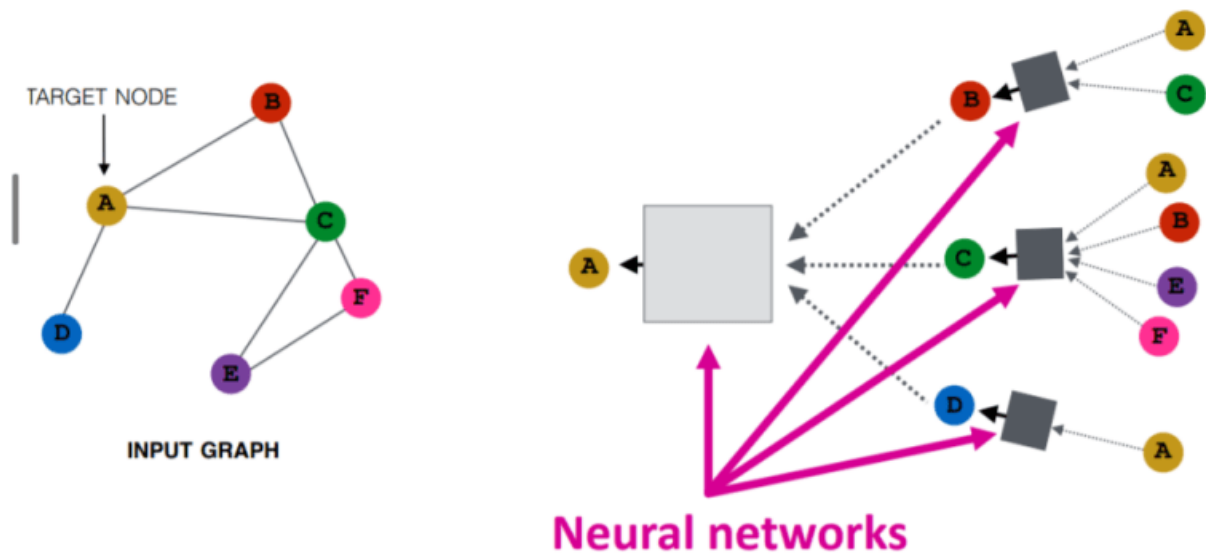
Determine node computation graph



Propagate and transform information

Neighborhood exploration and information sharing | [Source](#)

Once the locality information preserves the computational graph, we start aggregating. This is basically done using neural networks.

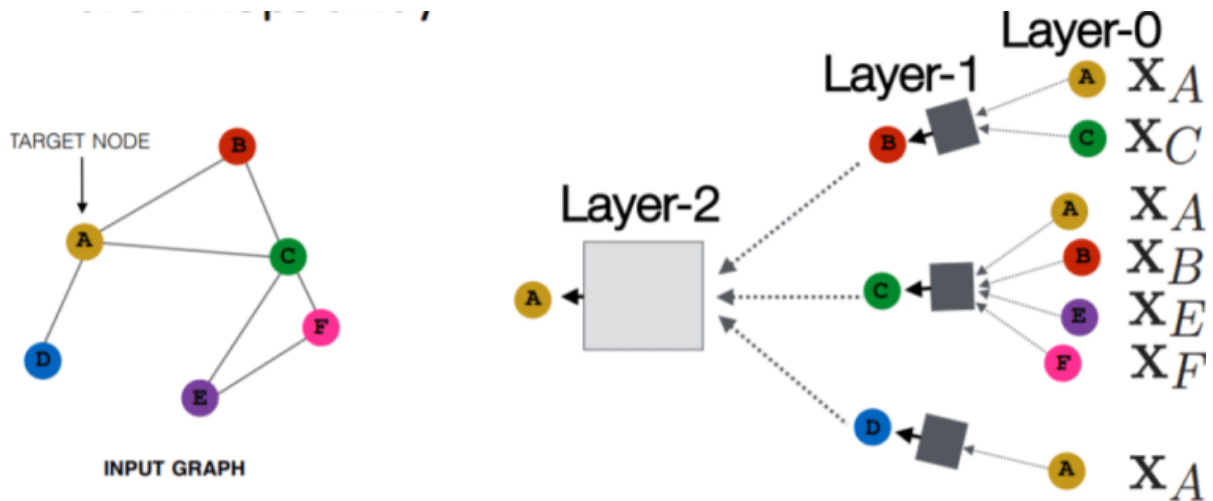


Neural networks

[Source](#)

Neural Networks are presented in grey boxes. They require aggregations to be order-invariant, like sum, average, maximum, because they are permutation-invariant functions. This property enables the aggregations to be performed.

Let's move on to the **forward propagation rule** in GNNs. It determines how the information from the input will go to the output side of the neural network.



Deep Model: Many Layers | [Source](#)

Every node has a feature vector.

For example, (X_A) is a feature vector of node A .

The inputs are those feature vectors, and the box will take the two feature vectors (X_A and X_C), aggregate them, and then pass on to the next layer.

Notice that, for example, the input at node C are the features of node C , but the representation of node C in layer 1 will be a hidden, latent representation of the node, and in layer 2 it'll be another latent representation.

So in order to perform forward propagation in this computational graph, we need 3 steps:

1. Initialize the activation units:

$$h_v^0 = X_v \text{ (feature vector)}$$

2. Every layer in the network:

$$h_v^k = \sigma(W_k \sum \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1}) \text{ where } k = 1, \dots, k-1$$

We can notice that there are two parts for this equation:

- The first part is basically averaging all the neighbors of node v .

$$W_k \sum \frac{h_u^{k-1}}{|N(v)|}$$

- The second part is the previous layer embedding of node v multiplied with a bias B_k , which is a trainable weight matrix and it's basically a self-loop activation for node v .

$$B_k h_v^{k-1}$$

- σ : the non-linearity activation that is performed on the two parts.

3. The last equation (at the final layer):

$$z_v = h_v^K$$

It's the embedding after K layers of neighborhood aggregation.

Now, to train the model we need to define a loss function on the embeddings.

We can feed the embeddings into any loss function and run stochastic gradient descent to train the weight parameters.

Training can be unsupervised or supervised:

- **Unsupervised training:**
Use only the graph structure: similar nodes have similar embeddings. Unsupervised loss function can be a loss based on node proximity in the graph, or random walks.
- **Supervised training:**
Train model for a supervised task like node classification, normal or anomalous node.

To recap, in this section we described a basic idea of generating node embeddings by aggregating neighborhood information.

Next, I'll discuss Graph Convolutional Networks (GCNs).

Graph Convolutional Networks

GCNs were first introduced in "Spectral Networks and Deep Locally Connected Networks on Graphs" (Bruna et al, 2014), as a method for applying neural networks to graph-structured data.

The simplest GCN has only three different operators:

- Graph convolution
- Linear layer
- Nonlinear activation

The operations are usually done in this order. Together, they make up one network layer. We can combine one or more layers to form a complete GCN.

In Python, we can easily build a GCN using PyTorch:

```
import torch
from torch import nn
```

```

class GCN(nn.Module):
    def __init__(self, *sizes):
        super().__init__()
        self.layers = nn.ModuleList([
            nn.Linear(x, y) for x, y in zip(sizes[:-1], sizes[1:])
        ])
    def forward(self, vertices, edges):
        # ----- Build the adjacency matrix -----
        # Start with self-connections
        adj = torch.eye(len(vertices))
        # edges contain connected vertices: [vertex_0, vertex_1]
        adj[edges[:, 0], edges[:, 1]] = 1
        adj[edges[:, 1], edges[:, 0]] = 1

        # ----- Forward data pass -----
        for layer in self.layers:
            vertices = torch.sigmoid(layer(adj @ vertices))

        return vertices

```

GraphSAGE idea

GraphSAGE (Hamilton et al, NIPS 2017) is a representation learning technique for dynamic graphs.

It can predict the embedding of a new node, without needing a re-training procedure.

To do this, GraphSAGE uses inductive learning. It learns aggregator functions which can induce new node embedding, based on the features and neighborhood of the node.

■ Simple neighborhood aggregation:

$$\mathbf{h}_v^k = \sigma \left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1} \right)$$

■ GraphSAGE:

Concatenate neighbor embedding
and self embedding

$$\mathbf{h}_v^k = \sigma \left(\left[\mathbf{W}_k \cdot \text{AGG} \left(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\} \right), \mathbf{B}_k \mathbf{h}_v^{k-1} \right] \right)$$

Generalized aggregation

[Source](#)


We can notice two big differences. Rather than summing two things together and losing track of them, we use a general aggregation function which keeps them separate by concatenating them.

Before, we were using the Mean aggregation function – we simply took the message from the neighbors and added them up, and then normalized that by the number of neighbors. Now, we can also make a pooling type approach, or we can also use a deep neural network like an LSTM.

Mean: Take a weighted average of neighbors

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$

Pool: Transform neighbor vectors and apply symmetric vector function


 Element-wise mean/max

$$\text{AGG} = \gamma(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\})$$

LSTM: Apply LSTM to reshuffled of neighbors

$$\text{AGG} = \text{LSTM}([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))])$$

Neighbor Aggregation: Variants | [Source](#)

A summary overview of GNNs Applications

Application

Deep Learning

Description

Text classification

Graph convolutional network/ graph attention network

A classic application of GNNs in NLP is Text Classification. GNNs utilize the inter-relations of documents or words to infer document labels. GCN and GAT models are applied to solve this task. They convert text to graph-of-words, and then use graph convolution operations to convolve the word graph. They show through experiments that the graph-of-words representation of texts has the advantage of capturing non-consecutive and long-distance semantics

Neural machine translation

Graph convolutional network/ gated graph neural network

The neural machine translation (NMT) is considered a sequence-to-sequence task. One of GNN's common applications is to incorporate semantic information into the NMT task. To do this, we utilize the Syntactic GCN on syntax-aware NMT tasks. We can also use the GGNN in NMT. It converts the syntactic dependency graph into a new structure by turning the edges into additional nodes and thus edges labels can be represented as embeddings

Relation extraction

Graph LSTM/ graph convolutional network

Relation Extraction is the task of extracting semantic relations from the text, which usually occur between two or more entities. Traditional systems treat this task as a pipeline of two separated tasks, i.e., named entity recognition (NER) and relation extraction, but new studies show that end-to-end modeling of entity and relation is important for high performance since relations interact closely with entity information

Image classification

Graph convolutional network/ gated graph neural network

Image classification is a basic computer vision task. Most of the models provide attractive results when given a huge training set of labeled classes. The focus now is towards getting these models to perform well on zero-shot and few-shot learning tasks. For that, GNN appears quite appealing. Knowledge graphs can provide the necessary information to guide the ZSL (Zero-shot learning) task

Object detection

Interaction detection

Region classification

Semantic segmentation

Graph attention network

Graph neural network

Graph CNN

Graph LSTM/ gated graph neural network/ graph CNN/ graph neural network

There are other applications of computer vision tasks like object detection, interaction detection, and region classification. In object detection, GNNs are used to calculate RoI features; in interaction detection, GNN is message-passing tools between humans and

objects; in region classification, GNNs perform reasoning on graphs that connect regions and classes

Physics

Graph neural network/ graph networks

Modeling real-world physical systems is one of the most basic aspects of understanding human intelligence. By representing objects as nodes and relations as edges, we can perform GNN-based reasoning about objects, relations, and physics in an effective way. Interaction networks can be trained to reason about the interactions of objects in a complex physical system. It can make predictions and inferences about various system properties in domains such as collision dynamics

Molecular fingerprints

Graph convolutional network

Molecular fingerprints are feature vectors that represent molecules. ML models predict the properties of a new molecule by learning from example molecules that use fixed-length fingerprints as inputs. GNNs can replace the traditional means that give a fixed encoding of the molecule to allow the generation of differentiable fingerprints adapted to the task for which they are required

Protein interface prediction

Graph convolutional network

This is a challenging problem with important applications in drug discovery. The proposed GCN-based method respectively learns ligand and receptor protein residue representation and merges them for pairwise classification. At a molecular level, the edges can be the bonds between atoms in a molecule or interactions between amino-acid residues in a protein. On a large scale, graphs can represent interactions between more complex structures such as proteins, mRNA, or metabolites

Combinatorial optimization

Graph convolutional network/ graph neural network/ graph attention network

Combinatorial optimization (CO) is a topic that consists of finding an optimal object from a finite set of objects. It is the base of many important applications in finance, logistics, energy, science, and hardware design. Most CO problems are formulated with graphs. In a recent work by DeepMind and Google, graph nets are used for two key subtasks involved in the MILP solver: joint variable assignment and bounding the objective value. Their neural network approach is faster than existing solvers on big datasets

Graph generation

Graph convolutional network/ graph neural network/ LSTM /RNN/ relational-GCN

Generative models for real-world graphs have drawn significant attention for their important applications including modeling social interactions, discovering new chemical structures, and constructing knowledge graphs. The GNN based model learns node embeddings for each graph independently and matches them using attention mechanisms. This method offers good performance compared to standard relaxation-based techniques

Conclusion

Over the past few years, graph neural networks have become powerful and practical tools for any problem that can be modeled by graphs.

In this article, we did a comprehensive overview of graph neural networks and introduced a wide range of GNN applications.

If you stayed with me until the end – thank you for reading!

<https://www.seldon.io/graph-neural-networks>

Graph Neural Networks and its Applications

The power of [machine learning](#) is being leveraged to solve increasingly complex problems across a range of different areas. Models are required to recognise and understand more abstract concepts and objects, and in many cases make non-linear decisions. Although powerful in their own right, the more traditional [types of machine learning models](#) lack the ability to accurately map and process some of the most complex problems.

Increasingly, [artificial neural networks](#) are recognised as providing the architecture for the next step in machine learning. These networks are designed to mirror the functionality of the human brain and nervous system. A web of connected nodes act as artificial neurons, and [deep learning](#) techniques are used to create models which can make non-linear decisions. However, the majority of models are designed to process euclidean input data, with a grid-like structure over one or two dimensional domains. This includes common representations of file types like images or text. But there is non-euclidean data that might exist over three dimensions, such as maps of networks or molecules. Valuable information will be lost if this data is converted to euclidean data structure.

Graphs are a common type of non-euclidean data structure, an abstract way of modelling data which consists of nodes and interconnecting edges. Graph Neural Networks are artificial neural networks developed to process graph structured input data. Graph Neural Networks can be seen as a subdomain of a wider field of geometric deep learning. This guide will explore the concept of Graph Neural Networks, including what they are, how they're built, the challenges the field faces, and the different applications of Graph Neural Networks.

What is a Graph Neural Network?

Graph Neural Networks are a type of artificial neural network which are designed to process graph structured data. They form part of a broader field of artificial neural networks, which are networks designed to reflect the functions of human or animal brains. There are many different types of [artificial neural network](#), but they are generally unified by the aim of artificially replicating the behaviour of neurons and synapses. Networks are usually grouped into layers of nodes or artificial neurons, connected with other nodes. These layers exist between an input and an output layer, through which the data travels.

Artificial neural networks are seen as a powerful way of creating machine learning models, which can understand complex concepts and patterns beyond traditional model architecture. Graph Neural Networks are unique because they process graph structured data, in contrast to many machine learning models that process euclidean data with a grid-like structure. Most of the most common types of data that are processed by machine learning models will be classed as euclidean data. This is data that exists in one or two-dimensional domains, for example text, audio files, and images.

In contrast, graph structured data can exist in a three dimensional domain, and is a key example of non-euclidean data. Graph data structures can be used to more accurately represent complex concepts. This could include molecules or networks where a third dimensional domain is important. Valuable information may be lost when this complex data is transformed into a euclidean structure. Most things can be represented by a graph structure, so Graph Neural Networks have a huge potential array of applications.

There are a range of different Graph Neural Networks, which can be understood as an artificial neural network that is capable of processing graph structured data. The choice of Graph Neural Network will hinge on the task at hand, as well as the type and complexity of graph that requires processing.

Four popular examples of Graph Neural Networks includes:

- Recurrent graph neural networks (RecGNN)
- Convolutional graph neural networks (ConvGNN)
- Graph autoencoders (GAE)
- Spatial-temporal graph neural networks (STGNN)

What is graph structured data?

It's important to grasp the basics of graph structured data to fully understand a Graph Neural Network. Graphs are abstract, non-euclidean structures consisting of a set of points connected by links or edges. Graphs have a defined amount of points or nodes, with the edges or links connecting pairings of nodes. Graphs therefore provide information on the relationships between different nodes as well as the overall structure of features.

The different edges or lines can be labelled or hold a value. Graph structured data can be ingested into models built on Graph Neural Networks, which in turn can perform tasks such as clustering, [regression](#) or classification. There are a range of different types of graphs, each with different properties.

The main types of graphs are:

- Directed graphs. The lines or edges have a defined direction
- Undirected graphs. The lines or edges have no specific direction.
- Weighted graphs. Value, labels or weighting is added to each line or edge.
- Cyclic graphs. A graph which includes a circle of connected nodes.

A huge range of things can be modelled as graph structured data. For example, train stations can be the nodes and the lines that connect them to the edges. A more abstract example would be within [reinforcement learning](#), which is a process in which a model learns through a trial and error feedback loop. The model will perform actions to transition between states in an environment. This can be modelled as a graph in which states are nodes and actions are edges.

Examples of graph structured data includes:

- Molecular structures, in which bonds are links and atoms nodes.
- Social networks, in which users are nodes and social connections are edges.
- Artificial neural networks, in which artificial neurons are nodes and the connections are edges.
- Country maps, in which cities and towns are nodes and roads are edges.

Applications of Graph Neural Networks in machine learning

Machine learning models often face issues when processing graph data, because of the sheer size of many graph structures and the web-like structure of the graph. A graph modelled on big data such as social networks can have billions of interlinking nodes, making it incredibly difficult to process. In addition, the different types of graphs will have different structures in space, making the approach to processing nodes an abstract problem. They're a web of interconnected nodes, so a clear start and end won't exist.

For traditional machine learning approaches, graphs must be transformed into vectors before being processed. This preprocessing can cause relevant information on the relationship between nodes to be lost. Graph Neural Networks are therefore used in machine learning to process graphs directly. Graph Neural Networks can then make predictions about the node or edges in graphs.

Models built on Graph Neural Networks will have three main focuses:

- Tasks focusing on nodes. Examples could include clustering or classifying nodes, or performing regression to predict a continuous value. Classification is the labelling of nodes into different categories, and clustering would be the segmenting of nodes into similar groupings.
- Tasks focusing on edges or lines. These represent the relationships between nodes, so generally this type of task will include classification on the type of edge or predicting the connection between nodes.
- Tasks focusing on the graph-level. This would include predictions and classification of the graph as a whole.

Because most things can be modelled using graph structured data, Graph Neural Networks have a huge range of applications. Graph-level tasks can include clustering and matching different graphs. This could mean clustering different diseases or viruses, or matching molecular structures. Node-level tasks such as node classification have a huge array of applications too. Text, voice and image files can all be classified using Graph Neural Network model architecture. Recommendation systems and text context extraction are both examples of edge-level tasks, which play a key role in approaches to translating sequences and sentences.

Graph Neural Networks have a range of applications, for example:

- Generating user recommendations based on relationships between different products previously purchased
- Recommending new connections in social media networks.
- Predicting shared interests of users within ecommerce or social media networks.
- Predicting changes in ecosystems or interconnected diseases, or the mutation of viruses.
- Classification of objects, or the labelling of unlabelled data through different nodes assigned a different object label.
- Processing modelled scientific systems and structures, with applications in physics, chemistry and biology.
- Processing of knowledge graphs which map relationships between events, locations, concepts or entities.
- Processing user interactions with other users, items and systems.
- Labelling of sequences and the extraction and translation of text.

How are Graph Neural Networks built?

There are many different examples of Graph Neural Network models, but the most common approaches will use different combinations of similar modular parts. The final [model pipeline](#) will be reflective of the network's application, the type and scale of graph structured data it will process, and the type of task the model will perform.

The general process for building a Graph Neural Network will be to define the complexity and scale of the graph data and the type of task the model will complete. The architecture of the Graph Neural Network can then be designed and built, usually from a combination of modules which deal with unique parts of graph structured data. For example a module which is used to sample large, complex graphs to make processing possible.

The first stage is to clearly define the structure of the graph data that needs to be processed. This could be a specific structure as with a molecular model, or a graph that needs to be generated from non-structured data. The type of graph should then be defined, to understand its complexity and scale. For example, establishing whether the graph edges are directional or not will highlight the graph's level of complexity. The size and scale of the graph needs to be determined. Some of the more complex graphs processed by Graph Neural Networks such as social media platform connections can have billions of nodes. These huge graphs aren't easy to access, store, or process, so Graph Neural Networks will need to include data sampling processes.

Next the task the Graph Neural Network model will be performing should be confirmed, so that the correct loss function can be created. Examples of tasks could be clustering of nodes

from chart structured data, or the prediction of edges and connections between nodes. Once this baseline information is determined, the model's architecture can be designed and built.

A common pipeline design includes combinations of different modules which deal with the processing of unique properties of graph data. These modules exist as layers in the Graph Neural Network dataflow. For examples where the graph is huge as with big data like in social media, a sampling module may be required to make processing possible. Likewise, a pooling module might be included to reduce the complexity of graphs.

The challenges of Graph Neural Networks

Graph Neural Networks can be leveraged to create powerful models which can achieve complex tasks beyond traditional machine learning techniques. But Graph Neural Networks face a range of problems and challenges shared across the machine learning field, as well as unique challenges in the graph domain.

A major challenge is in the pretraining and training phase of creating a Graph Neural Network. Whereas a traditional machine learning technique may need hundreds of thousands of data points to be trained, Graph Neural Networks may need millions or billions of training data points. Graph Neural Network models generally need labelled training data, which is labelled by humans so is an extremely resource-intensive process. Techniques in transfer learning are usually leveraged to lower the intensity of training required.

Graph Neural Networks are black box models, which means it's difficult to adequately explain the decision-making process of the model. The challenge is one of model explainability, which is important once a model is deployed to a real-world environment. Decisions made by models may not be fully trusted if they cannot be explained to a human. As Graph Neural Network models have strong applications in medicine and science research, explainability is a focus and challenge.

Finally, the complexity of the graph structured data is an intrinsic challenge for Graph Neural Networks. As models designed for the graph domain, the complexity of graph structures are a constant consideration and challenge. There are many different types of graph structures which range in scale and complexity. More powerful Graph Neural Networks are required as more complex graph structures emerge, such as big data from social media platforms. There's more data than ever before, and the complexity of graph structured data models are increasing exponentially. Dynamic graph structures add another layer of challenge that is a focus of research too.

Machine learning deployment for every organisation

Seldon moves machine learning from POC to production to scale, reducing time-to-value so models can get to work up to 85% quicker. In this rapidly changing environment, Seldon can give you the edge you need to supercharge your performance.

With Seldon Deploy, your business can efficiently manage and monitor machine learning, minimise risk, and understand how machine learning models impact decisions and business processes. Meaning you know your team has done its due diligence in creating a more equitable system while boosting performance.

<https://www.analyticsvidhya.com/blog/2022/03/what-are-graph-neural-networks-and-how-do-they-work/>

<https://revolutionized.com/graph-neural-network/>

Heterogeneous GNN

<https://pytorch-geometric.readthedocs.io/en/latest/notes/heterogeneous.html>