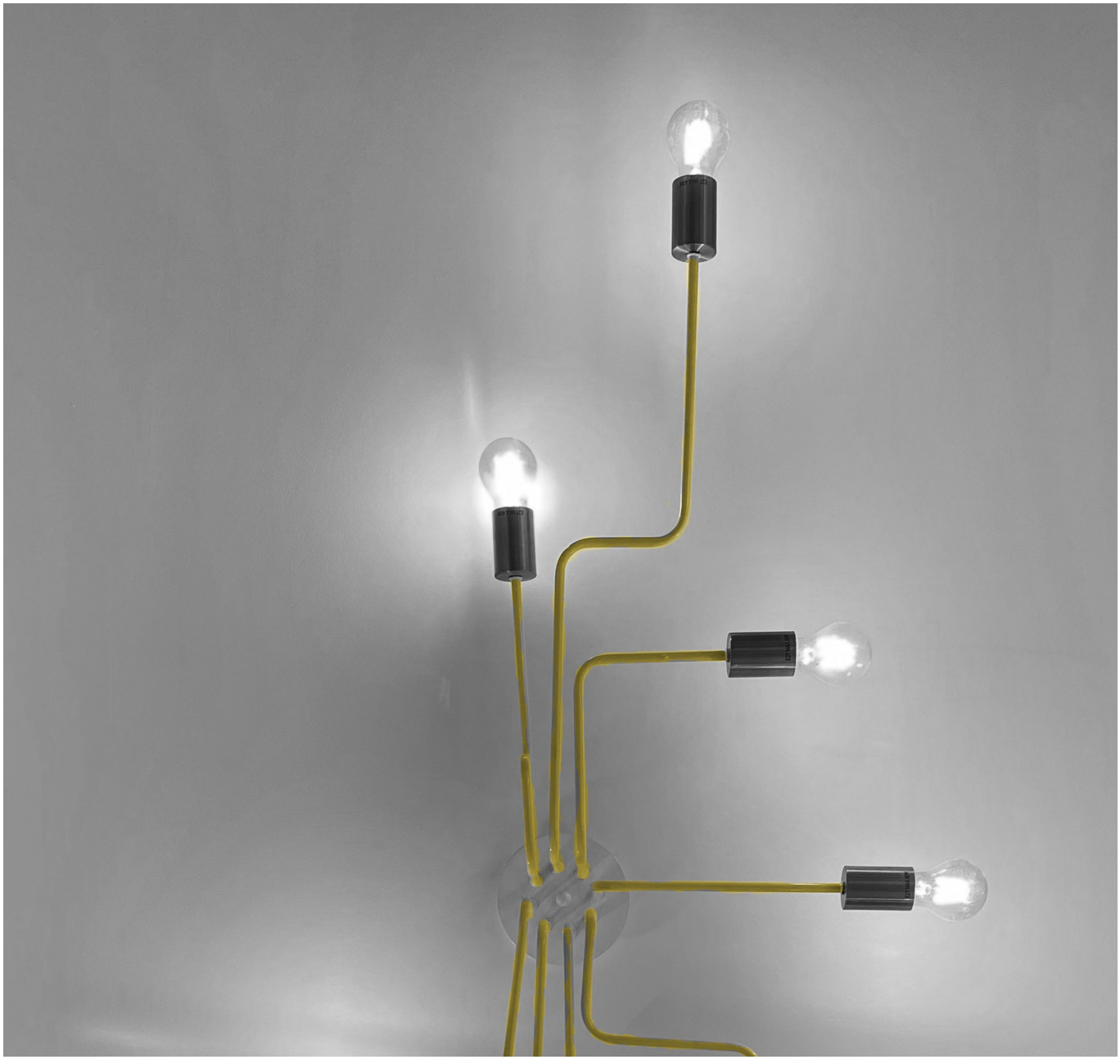
Introduction to Graph Neural Networks

Graph neural networks — their need, real-world applications, and basic architecture with the NetworkX library

Nikita Sharma Follow

Oct 28, 2020 · 9 min read



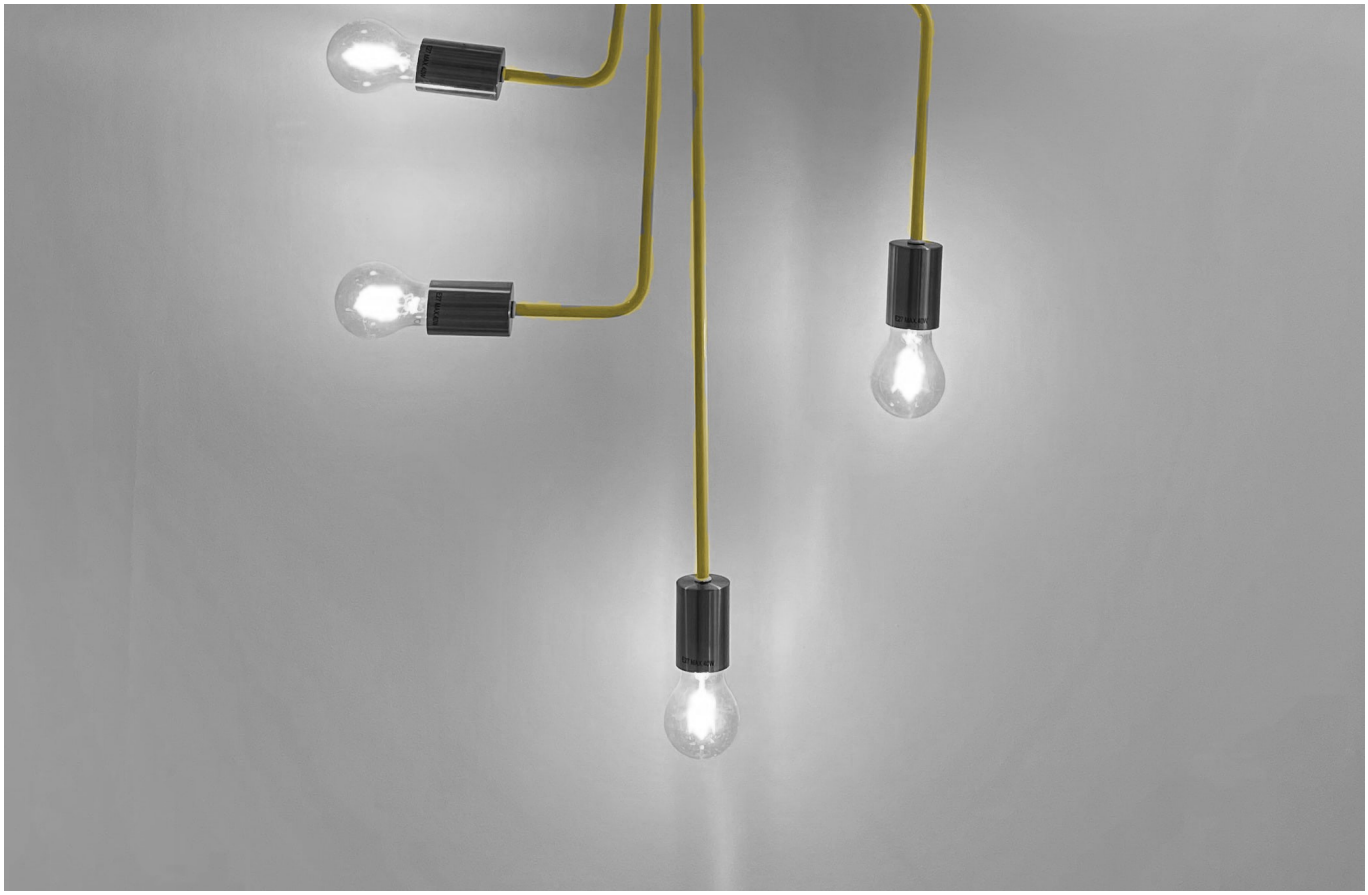


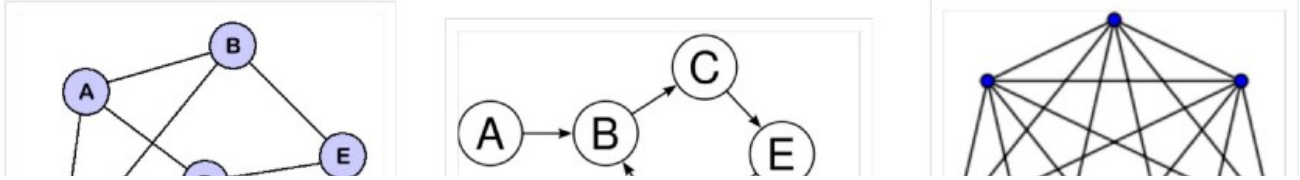
Photo by Fabio Bracht on Unsplash

In this post, we are going to investigate a relatively newer field in deep learning which involves graphs — a very important and widely used data structure. This post encompasses the basics of graphs, the amalgamation of graphs and deep learning, and a basic idea about graph neural networks and their applications. We will also briefly discuss on how to build graphs with a Python library called NetworkX

So, let's dive right in!

What are graphs?

In the world of computer science, graphs are a type of data structure having two components: Nodes (or vertices) and edges, which connect two nodes. Thus, a graph can be defined as a collection of loosely inter-connected nodes via edges.



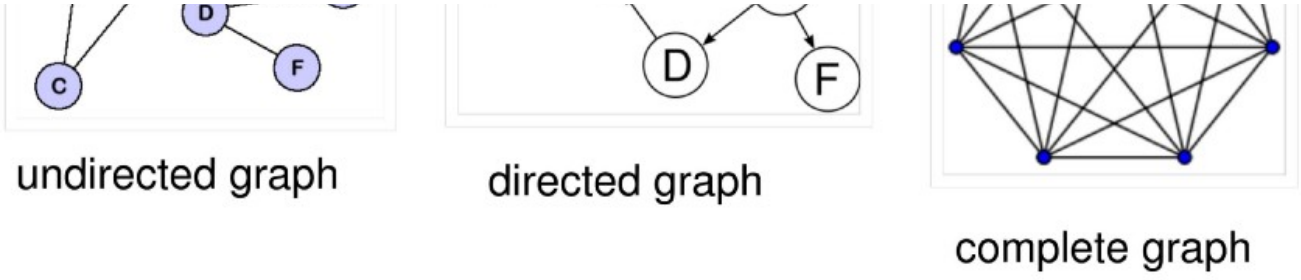


Image Source: https://medium.com/data-structures-and-algorithms/graph-dd2b72c32f1f

The nodes of a graph can be homogenous with all nodes having a similar structure, or heterogenous nodes having different types of structure. The edges define the relationship one node has with another. Edges can be bidirectional (from one node u to another v and vice versa), or unidirectional (from one node u to another node v). Edges can also be weighted — having a weight assigned to the edge that might depict the edge's cost or importance.

An example: Let us suppose a graph to be considered as a network of cities — the cities under observation being nodes and the roads connecting them being edges. Now, there can be various types of relevant problems that can be solved with graphs, such as finding out the shortest distance between cities (where roads can also be weighted as per the condition of the roads or traffic), or finding the cities which are well-connected to each other, etc.

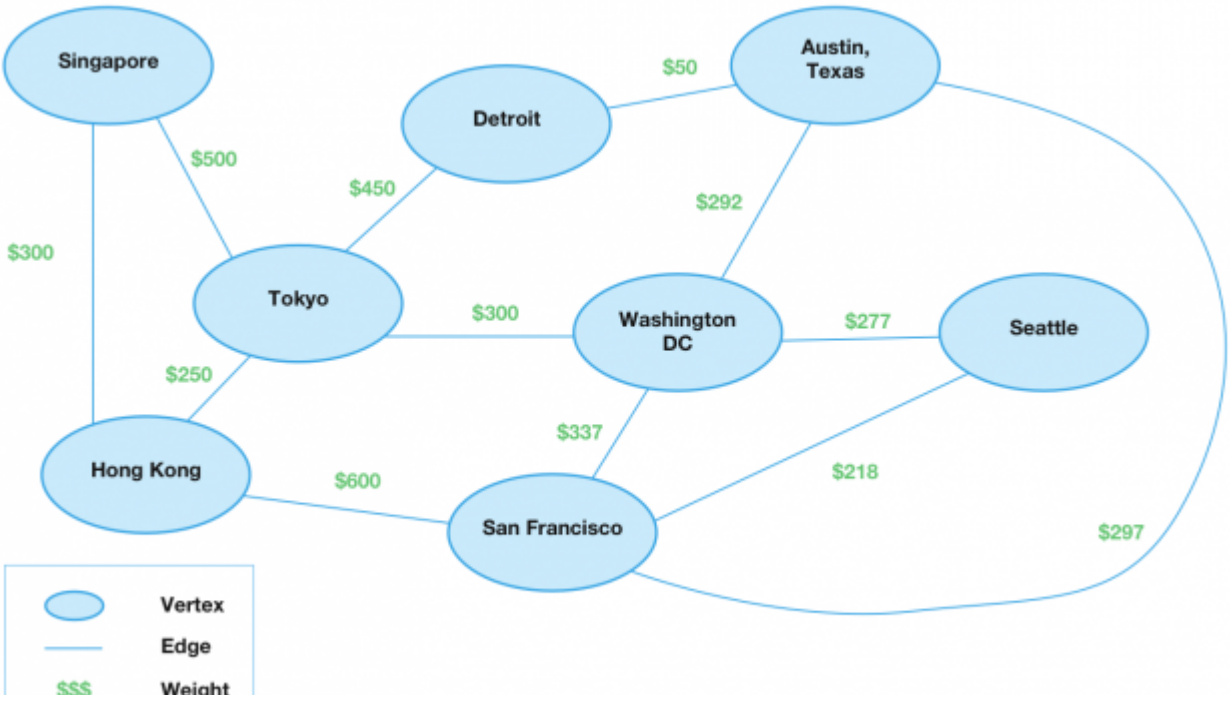
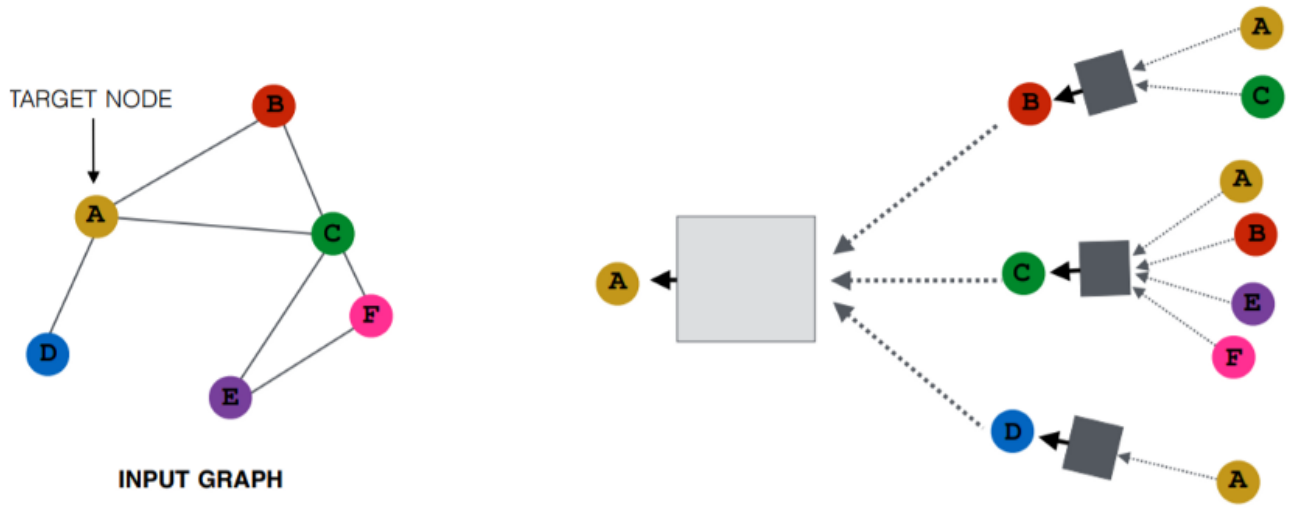


Image Source: https://www.raywenderlich.com/773-swift-algorithm-club-graphs-with-adjacency-list

What are Graph Neural Networks (GNN)?

Graphs have tremendous expressive powers and are therefore gaining a lot of attention in the field of machine learning. Every node has an embedding associated with it that defines the node in the data space. Graph neural networks refer to the neural network architectures that operate on a graph. The aim of a GNN is for each node in the graph to learn an embedding containing information about its neighborhood (nodes directly connected to the target node via edges). This embedding can then be used for different problems like node labelling, node prediction, edge prediction, etc.



Each node and its neighborhood

Thus, after having embeddings associated with each node, we can convert edges by adding feed forward neural network layers and combine graphs and neural networks.

Need for Graph Neural Networks

The need for graph neural networks arose from the fact that a lot of data available to us is in an unstructured format. Unstructured data is data that has not been processed or does not have a pre-defined format which makes it difficult to analyze. Examples of such data are audio, emails, and social media postings. To make sense of this data and to derive inferences from it, we need a structure that defines a relationship between these unstructured data points. The existing machine learning architectures and algorithms

do not seem to perform well with these kinds of data. The primary advantages of graph neural networks are:

1. The graph data structure has proven tremendously successful in the field of computer science while working with unstructured data.
2. Graphs are helpful in defining concepts which are abstract, like relationships between entities. Since each node in the graph is defined by its connections and neighbors, graph neural networks can capture the relationships between nodes in an efficient manner.

Thus, developing GNNs for handling data like social network data, which is highly unstructured, is an exciting amalgamation of graphs and machine learning which holds a lot of potential.

Real-Life Applications of Graph Neural Network

Being introduced recently in 2018, the GNNs still have a lot of real-life applications because their architecture resonates with the irregularity in data collected from various sources. Currently, GNNs have been the hot topic for:

Social Network Analysis — Similar posts prediction, tags prediction, and recommending content to users.

Natural Sciences — GNNs have also gained popularity in dealing with molecular interactions like protein-protein interactions.

Recommender Systems — A heterogenous graph can be used to capture relationships between users and items to recommend relevant items to a buyer.

How do Graph Neural Networks work?

After we have the basic structure of the graph neural network (nodes with their embeddings and edges with feed forward layers), we can move forward to understanding how GNNs actually work.

The basic idea is to learn neighborhood embeddings by aggregating information from a node's neighbors via edges using neural networks.

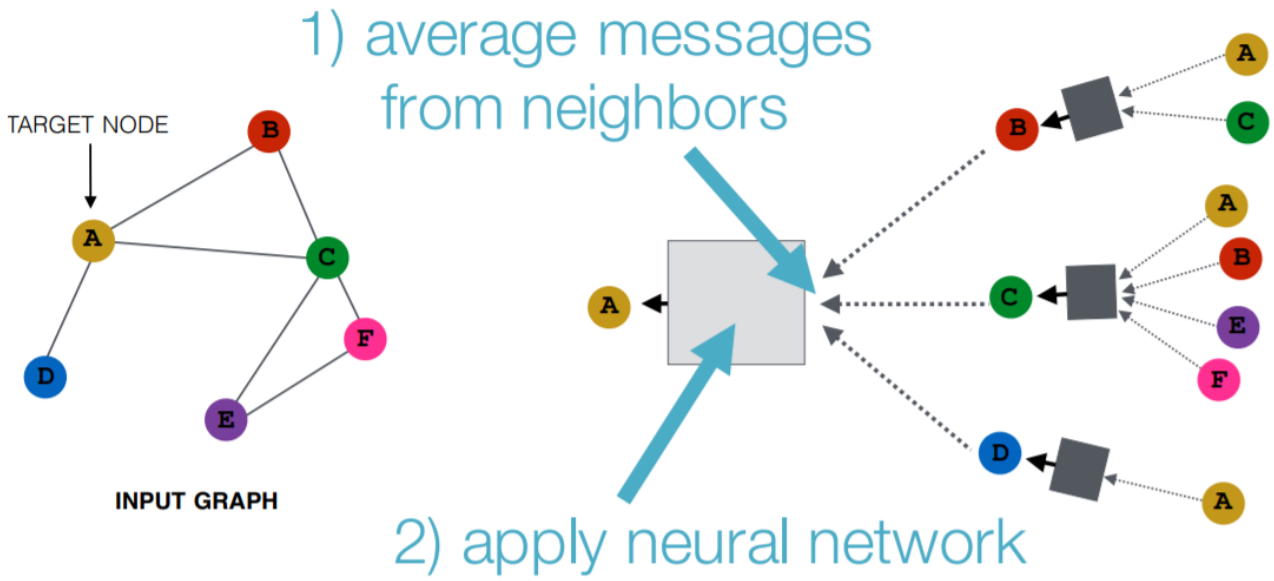


Image source: http://snap.stanford.edu/proj/embeddings-www/files/nrltutorial-part2-gnns.pdf

Neighborhood Aggregation or Message Passing

Message passing refers to passing and receiving information between nodes about its neighborhood. Consider a target node having its initial embeddings: It receives information from its neighbors passed via edge neural networks. Data from these edges are aggregated (many techniques are used, like max pooling, averaging, etc.,) and passed to the activation unit of a node to get a new set of embeddings for the node. Every node in the initial setup has features x_v. The embeddings for each node after message passing can be defined as:

$$\mathbf{h}_v = f(\mathbf{x}_v, \mathbf{x}_{c[1]}, \mathbf{h}_{c[1]}, \mathbf{x}_{c[1]})$$

taken from the research paper: https://arxiv.org/pdf/1812.08434.pdf

Where *x_ne[v]* denotes the features of the neighbors of *v, x_co[v]* is the edge features connected to *v, h_ne[v]* is the embedding of the neighbors of v.

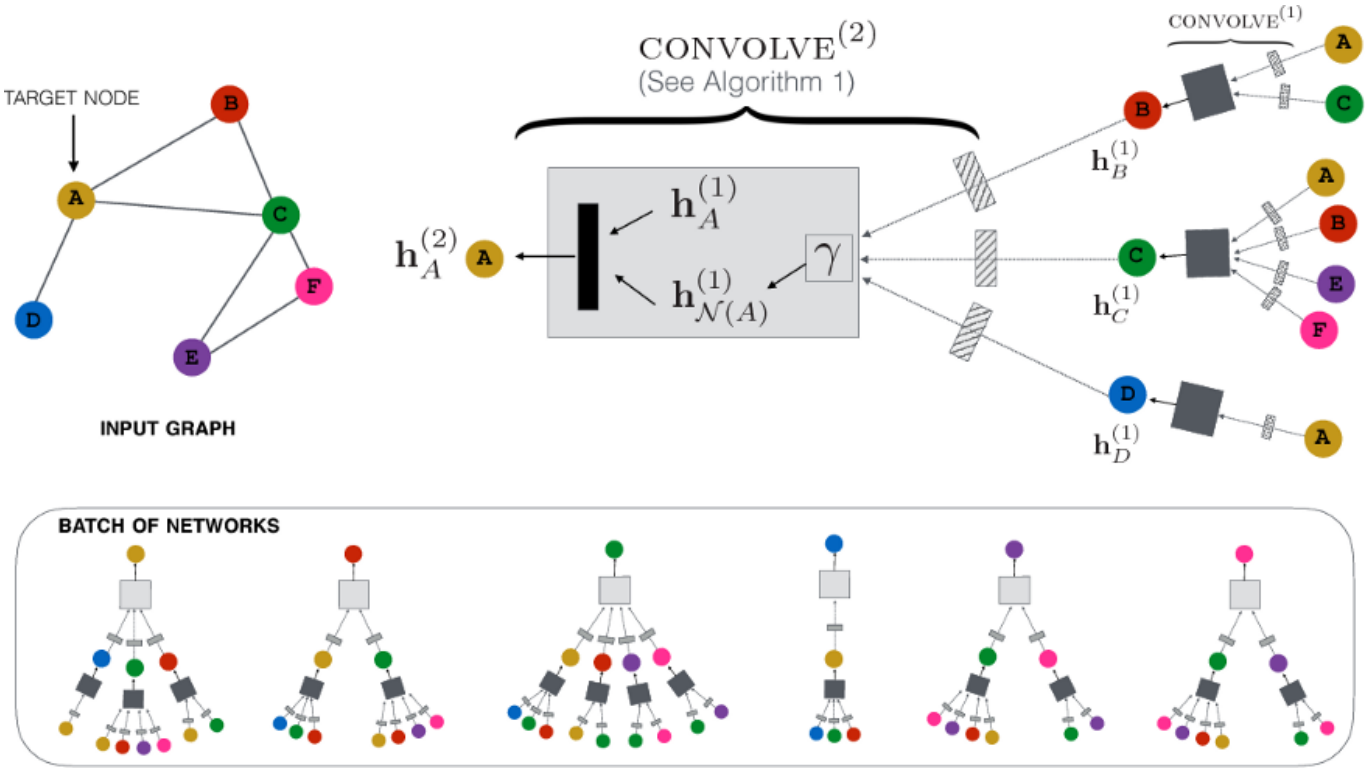


Image Source: http://snap.stanford.edu/proj/embeddings-www/files/nrltutorial-part2-gnns.pdf

In the above figure, $h_a^{(1)}$ is the initial embedding of the node, $hN_a^{(1)}$ is the aggregated embeddings of its neighbors. Combining these and passing to the node's activation unit or filter will provide the new embedding for node A, which will also contain information about its neighbors. In this manner, each node gets a new set of embeddings for itself which determines its position in the graph. With various iterations or K layers of message passing, a node learns more and more about its neighborhood and its distant neighbors as well.

Eventually, each node has a rough idea about the complete graph (or a part of it, depending on the number of iterations and node-node distance/path or layers considered).

An example: Consider a graph with social media posts as nodes. Now, if these nodes have embeddings and are labelled with tags like romance, science, comedy, etc., we get a new post and we need to provide it with a tag. Using the existing network and embeddings, neighborhood aggregation will help us predict the labels and embeddings for the unseen node.

Advantage: Rather than running the entire algorithm again, we can just use embeddings of the neighbors to determine the locality of the new post. Therefore, GNN-based recommendation can be more efficient and scalable than other traditional machine learning recommendation algorithms out there for dealing large datasets.

Types of Graph Neural Networks

Graphs are used with various existing neural network architectures to yield promising results for various machine learning problems. The two most dominant networks are discussed briefly below.

Graph Convolutional Networks (GCNs)

Convolutional neural networks(CNNs) have been vastly used for image classification and segmentation problems. Convolutional operation refers to applying a spatial filter to the input image and getting a feature map as a result.

You can read more about CNNs here.

GCNs refer to applying a spatially moving filter over the nodes of the graph which contains embeddings or data relevant to each node to get a feature representation of each node. Stacking a number of convolutional layers like a regular CNN can also be done to incorporate information from larger neighborhoods.

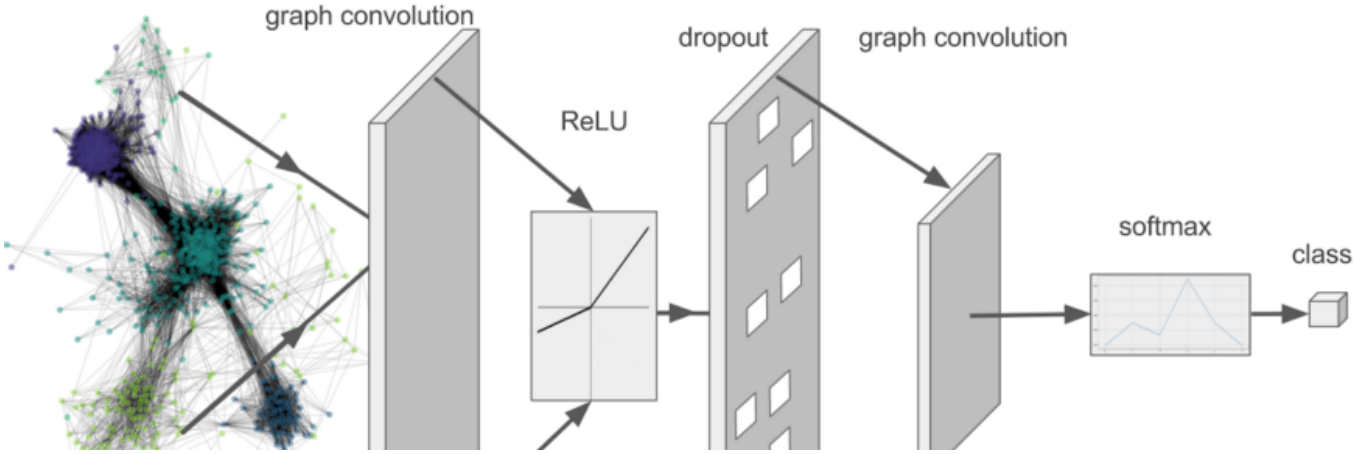


Image source: https://www.experoinc.com/post/node-classification-by-graph-convolutional-network

Graph Auto-Encoder Networks

Auto-encoders are neural networks which consist of two networks combined via a bottleneck layer: An encoder, which downsamples the input by passing it through convolutional filters to provide the compact feature representation of the image, and a decoder which takes the representation provided by the encoder as input and tries to re-construct the input according to the same.

You can read more about auto-encoders here.

Graph auto-encoders try to learn a compact representation of the graph and then re-construct the graph using the decoder. They can be used to learn graph embeddings and hence can be used for predicting embeddings for un-seen nodes and to classify newer nodes into existing categories within the graph.

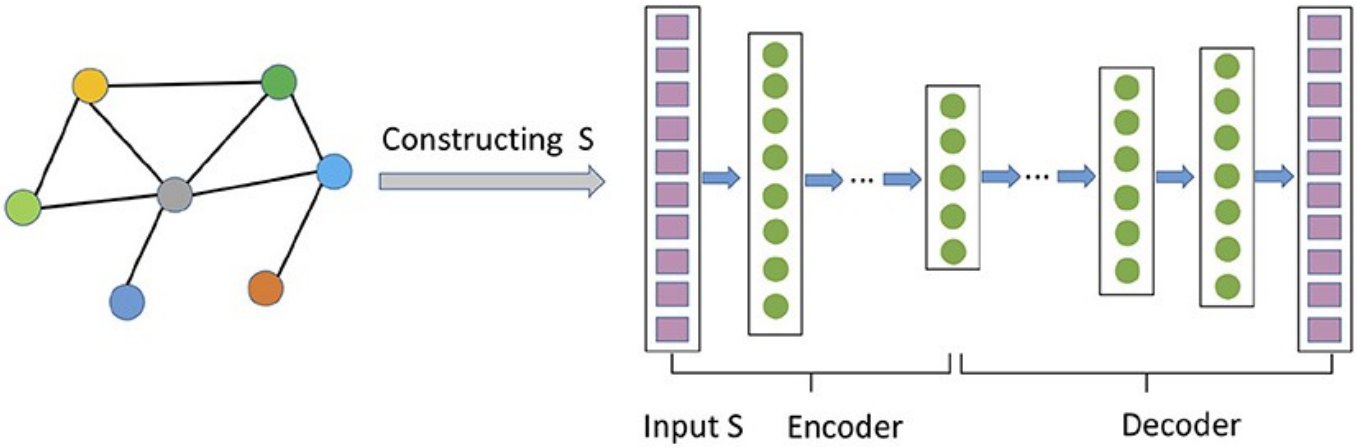


Image source: https://www.frontiersin.org/articles/10.3389/fdata.2019.00002/full

Other kinds of graph neural networks like spatial and temporal graph neural networks, generative graph neural networks, recurrent graph neural networks, etc., have also been developed.

Constructing Graphs with NetworkX

NetworkX, as mentioned in its documentation, is a “Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.” Let’s learn how to build graphs with NetworkX:

Install NetworkX:

```
pip install networkx
```

Creating a graph:

```
import networkx as nx
G = nx.Graph()
#defines an empty graph
```

Adding nodes to the graph:

```
#adds node 1
G.add_node(1)
#adds nodes from any iterable like list
G.add_nodes_from([2, 3])
```

Adding edges to the graph:

```
#adding an edge from first node to another
G.add_edge(1, 2)
#adding edges from a list
G.add_edges_from([(1, 2), (1, 3)])
#adding a weighted edge
G.add_edge(1, 2, weight=4.5)
```

In this manner, we can construct a graph that we want to work on. Here is a simple example to find the shortest path between two nodes:

```
import networkx as nx
G = nx.Graph()
G.add_edge('A', 'B', weight=5)
G.add_edge('B', 'D', weight=3)
G.add_edge('A', 'C', weight=4)
G.add_edge('C', 'D', weight=5)
nx.shortest_path(G, 'A', 'D', weight='weight')
['A', 'B', 'D']
```

Libraries that Support Graph Neural Networks

For the purpose of creation and analysis of graph structure, the following Python libraries can be used:

NetworkX - NetworkX documentation

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of...

networkx.github.io

python-igraph

igraph is on the Python Package Index with pre-compiled wheels for most Python distributions and platforms, so in most...

igraph.org

For the purpose of creating graph neural networks, the following libraries have been gaining a lot of attention and support:

dmlc/dgl

Documentation (Latest | Stable) | DGL at a glance | Model Tutorials | Discussion Forum | Slack Channel DGL is an...

github.com

deepmind/graph_nets

Graph Nets is DeepMind's library for building graph networks in Tensorflow and Sonnet. Contact graph-nets@google.com...
github.com

Conclusion

In this post, we went through a journey of understanding graphs and graph neural networks. Graph neural networks are an intuitive solution to making sense of unstructured data and are useful for many real-world applications. We also learned how to build graphs from scratch with a library called NetworkX. Building an efficient graph is influential to the output of the network.

Hope you like this article and please let me know what you think.

Until next time!

*Editor's Note: **Heartbeat** is a contributor-driven online publication and community dedicated to exploring the emerging intersection of mobile app development and machine learning. We're committed to supporting and inspiring developers and engineers from all walks of life.*

*Editorially independent, Heartbeat is sponsored and published by **Fritz AI**, the machine learning platform that helps developers teach devices to see, hear, sense, and think. We pay our contributors, and we don't sell ads.*

*If you'd like to contribute, head on over to our **call for contributors**. You can also sign up to receive our weekly newsletters (**Deep Learning Weekly** and the **Fritz AI Newsletter**), join us on **Slack**, and follow Fritz AI on **Twitter** for all the latest in mobile machine learning.*

Machine Learning Graph Graph Neural Networks Heartbeat Neural Networks