Everything you need to know about Graph Theory for Deep Learning

Graph Learning and Geometric Deep Learning — Part 0

Flawnson Tong Follow

Apr 23, 2019 · 12 min read

> ***The best way to predict the future is to create it — Abraham Lincoln***

Machine learning is all the rage today, and once the science catches up with the hype, it will likely become a normality in our lives. One of the ways we are reaching for the next step is with a new form of deep learning; Geometric Deep Learning. Read about the inspiration and ideas here. The focus of this series is on how we can use Deep Learning on on graphs

The two prerequisites needed to understand Graph Learning is in the name itself; **Graph Theory and Deep Learning**. This is all you need to know to understand the nature of, and build a high-level intuition for these two ideas.

Progress in A.I means more cute robots (Courtesy of Techgyd)

Graph Theory — crash course

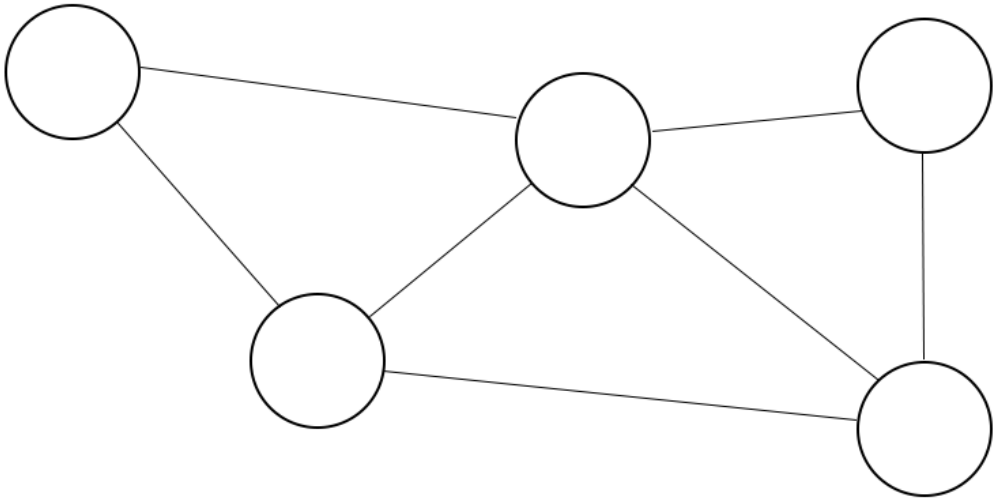
What is a graph?

A **graph**, in the context of graph theory, is a structured datatype that has **nodes** (entities that hold information) and **edges** (connections between nodes that can also hold information). A graph is a way of structuring data, but can be a datapoint itself. Graphs are a type of **Non-Euclidean data**, which means they exist in 3D, unlike other datatypes like images, text, and audio. Graphs can have certain properties, which limit the possible actions and analysis that can be performed on them. These properties can be defined.

Graph Definitions

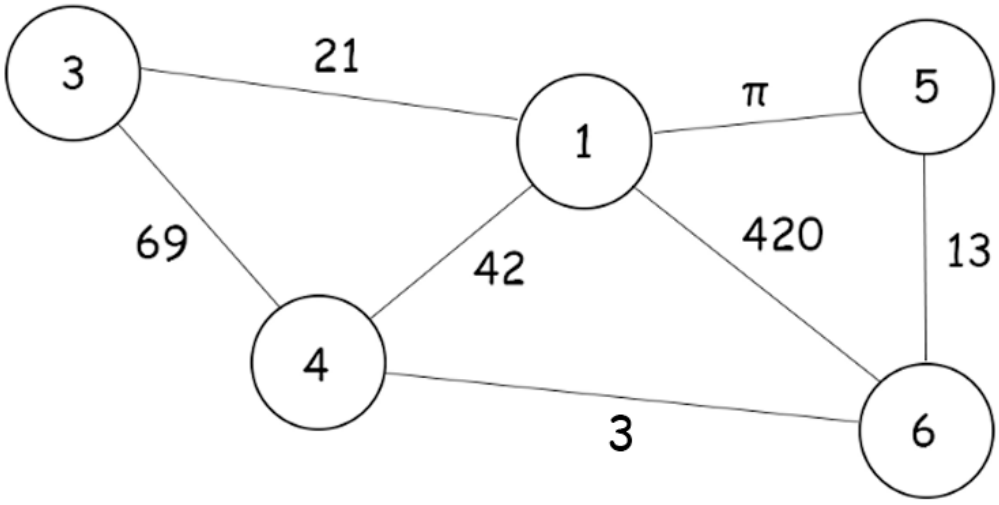
First let's cover some definitions, brought to you by the easiest Photoshop job of my life.

In computer science, we talk alot about a data structure known as **graphs:**



He's cute, let's call him Graham

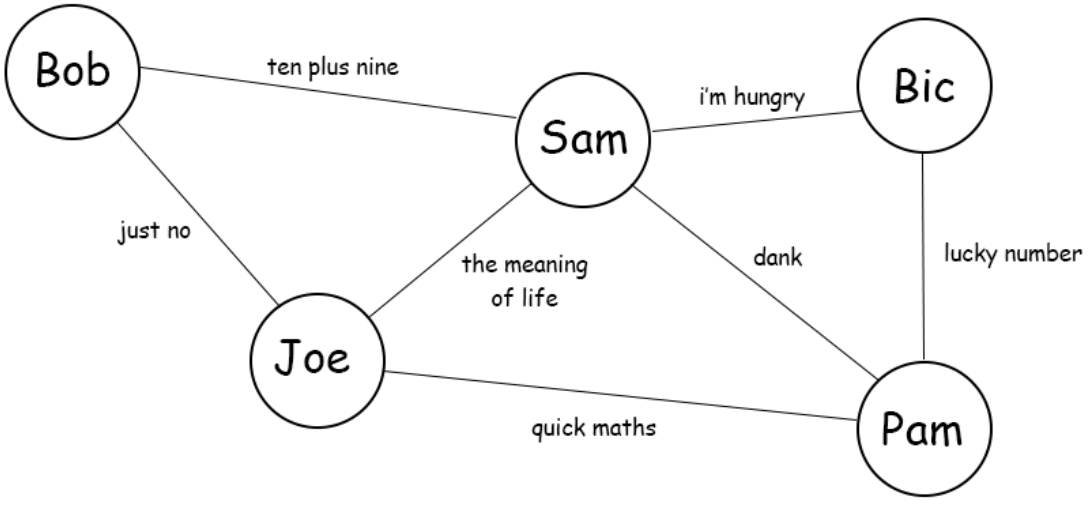
Graphs can have **labels on their edges and/or nodes**, let's give Graham some edge and node labels.



Graham looks rather dashing in his new attire

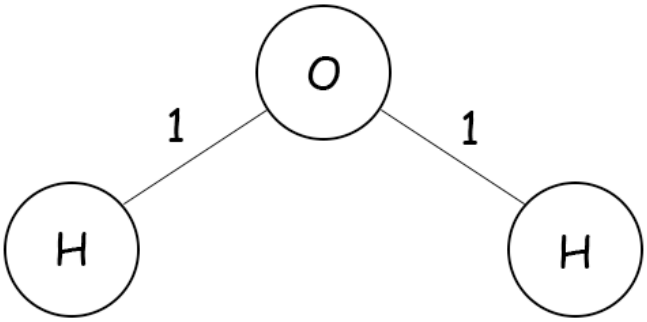
Labels can also be considered **weights**, but that's up to the graph's designer.

Labels don't have to be numerical, they can be textual.



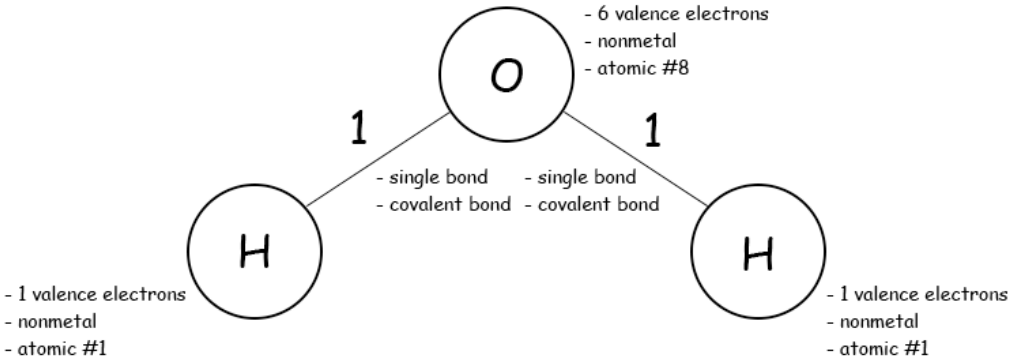
Graham also likes memes

Labels don't have to be unique; it's entirely possible and sometimes useful to give multiple nodes the same label. Take for example, a hydrogen molecule:



Notice the mix of numerical and textual datatypes

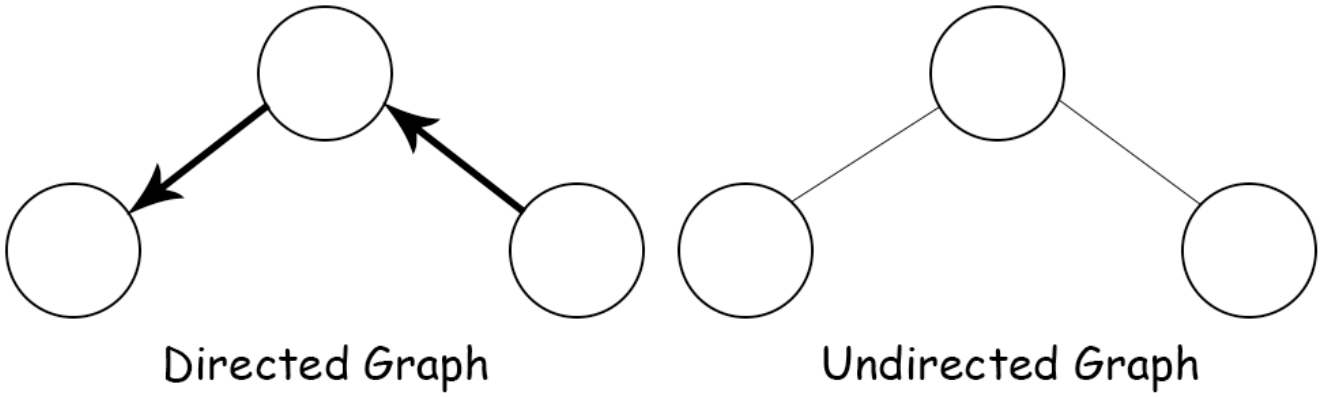
Graphs can have **features** (a.k.a attributes)**.**



Take care not to mix up features and labels. An easy way to think about it is using an analogy to names, characters, and people:

> a node is a person, a node's label is a person's name, and the node's features are the person's characteristics.

Graphs can be **directed or undirected:**



Note that directed graphs can have undirected edges too

A node in the graph can even have an edge that points/connects to itself. This is known as a **self-loop.**

Graphs can be either:

- **Heterogeneous** — composed of different types of nodes
- **Homogeneous** — composed of the same type of nodes

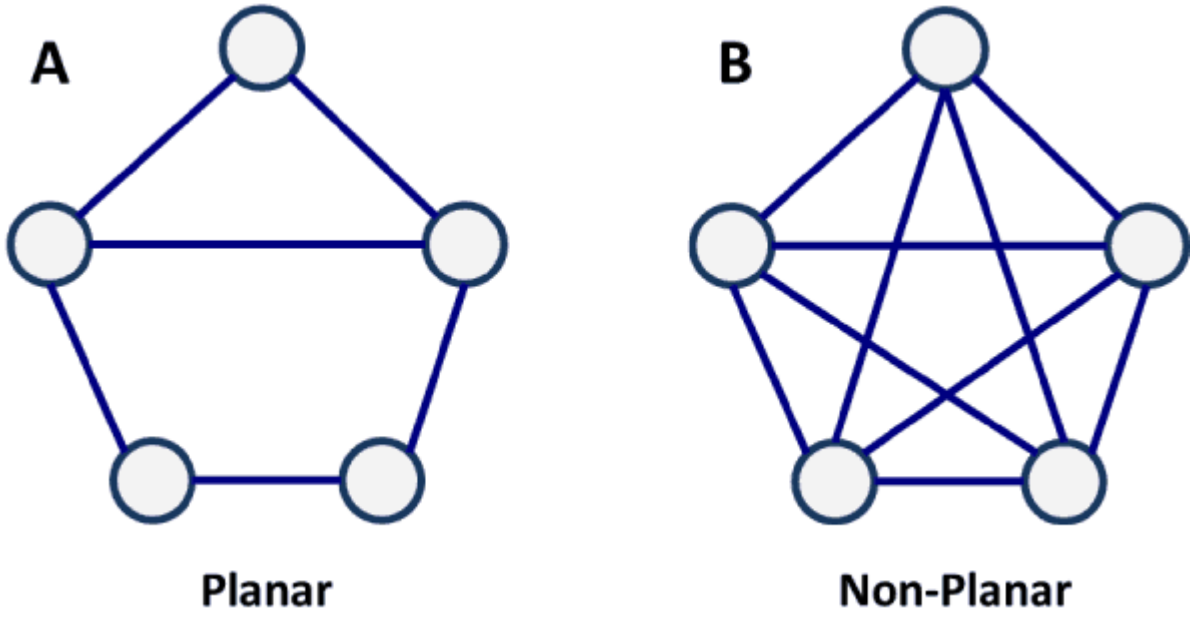
and are either:

- **Static** — nodes and edges do not change, nothing is added or taken away
- **Dynamic** — nodes and edges change, added, deleted, moved, etc.

Roughly speaking, graphs can be vaguely described as either

- **Dense** — composed of many nodes and edges
- **Sparse** — composed of fewer nodes and edges

Graphs can be made to look neater by turning them into their **planar** form, which basically means rearranging nodes such that edges don't intersect



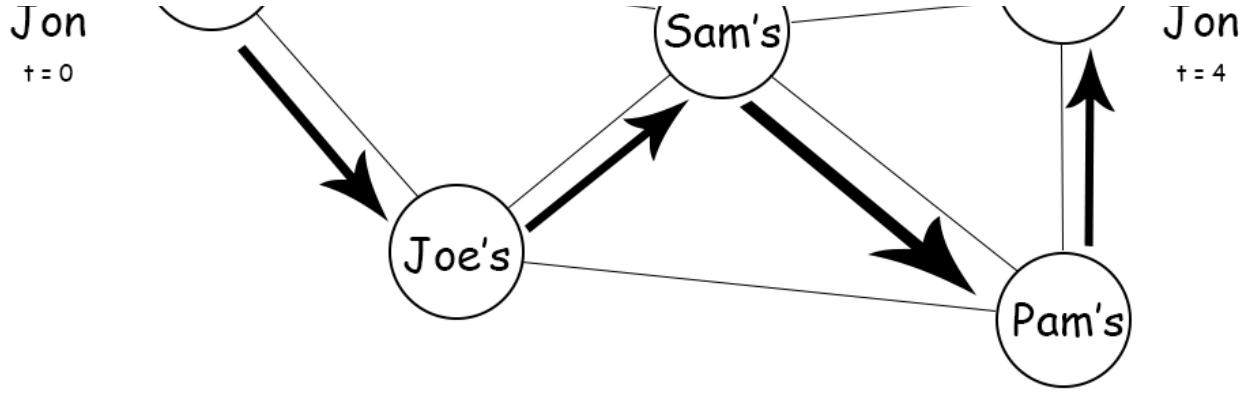
(Courtesy of The Geography of Transport Systems)

These concepts and terminology will come in handy as we explore the many different methods currently being employed in the various GNN architectures. Some of these basic methods are described in:

Graph Analysis

There are a host of different Graph structures available for a ML model to learn from (Wheel, Cycle, Star, Grid, Lollipop, Dense, Sparse, etc.)

You can traverse a graph

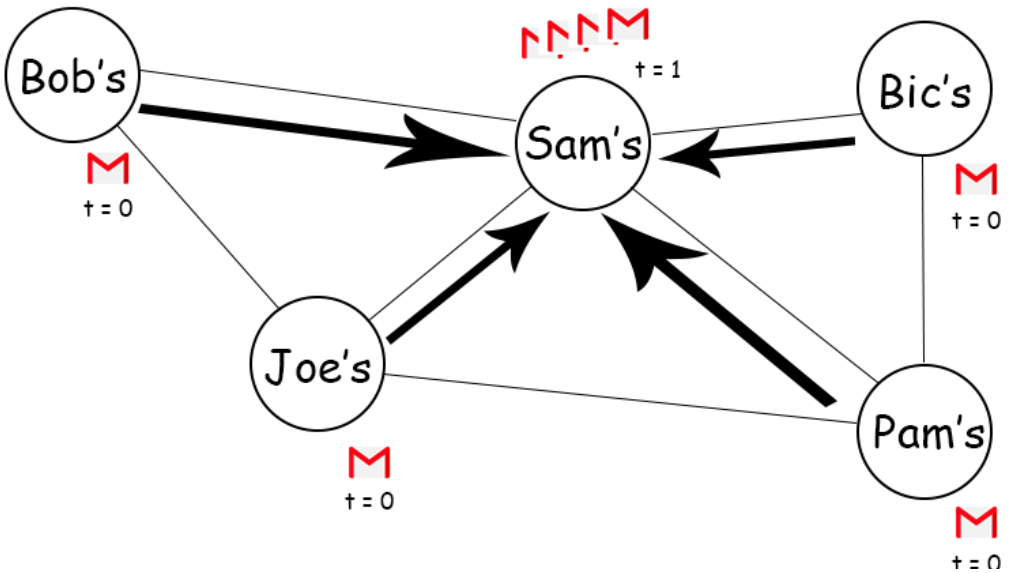


Jon went from Bob's to Bic's in 4 timesteps; he better hope it doesn't snow!

In this case, we're traversing an **undirected** graph. Obviously, if the graph was **directed**, one would simply follow the direction of the edges. There are a couple different types of traversals, so be careful of the wording. These are a couple most common graph traversal terms and what they mean:

- **Walk:** A graph traversal — a **closed walk** is when the destination node is the same as the source node
- **Trail:** A walk with no repeated edges — a **circuit** is a closed trail
- **Path:** A walk with no repeated nodes — a **cycle** is a closed path

Building on the concept of traversals, one can also **send messages across a graph.**



Sam? More like S-p-am...

All of Sam's neighbors are sending him a message, where *t* stands for the timestep. Sam can choose to open his mailbox and update his own information. The concept of propagating information throughout a network is super important for models with **attention mechanisms**. In graphs, message passing is one way we **generalize convolutions.** More on that later.

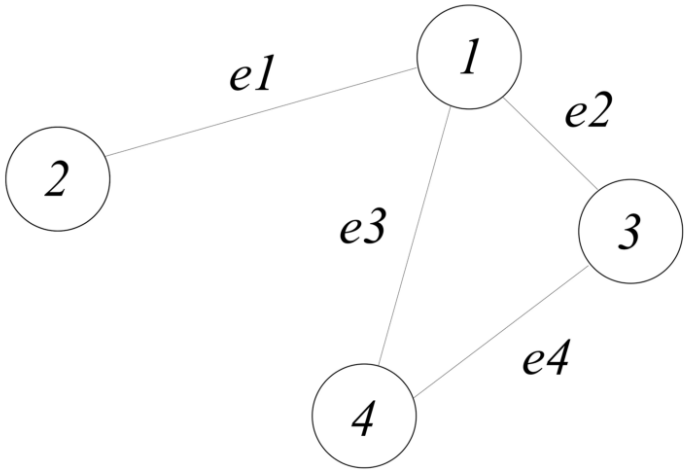
E-graphs — graphs on computers

Having learned all this, you now have a basic understanding of graph theory! Any other concepts important to GNNs will be explained as they come but in the meantime, there is still one last topic concerning graphs that we need to cover. **We must learn how to express graphs computationally.**

There are a couple ways one can turn a graph into a format that a computer can digest; all of them are different types of matrices.

Incidence Matrix (*I*):

The Incidence Matrix, commonly denoted with a capital ***I*** in research papers, Made up of 1s, 0s, and -1s, the incidence matrix can be made by following a simple pattern:



	e_1	e_2	e_3	e_4
1	1	1	1	0
2	1	0	0	0
3	0	1	0	1
4	0	0	1	1

$$= \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

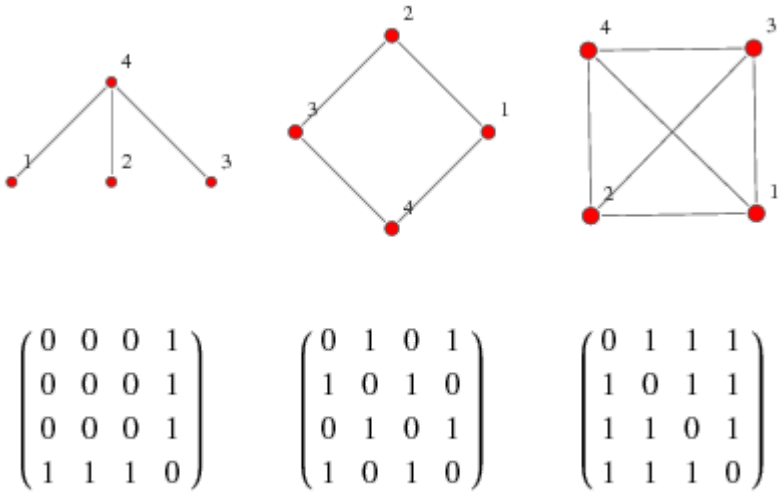
From Graph to Incidence Matrix

(Weighted) Adjacency Matrix (*A*):

The Adjacency Matrix of a graph is be made of 1s and 0s **unless** it is otherwise weighted or labelled. In any case ***A*** can be built by following this rule:

$$a_{ij} = \begin{cases} 1, \text{ if } (v_i, v_j) \in E \\ 0, \text{ otherwise} \end{cases}$$

The Adjacency Matrix of a undirected graph is therefore symmetrical along its diagonal, from the top left object to the bottom right:



Adjacency Matrices (Courtesy of Wolfram Mathworld)

Adjacency matrices of directed graphs only cover one side of the diagonal line, since directed graphs have edges that go in only one direction.

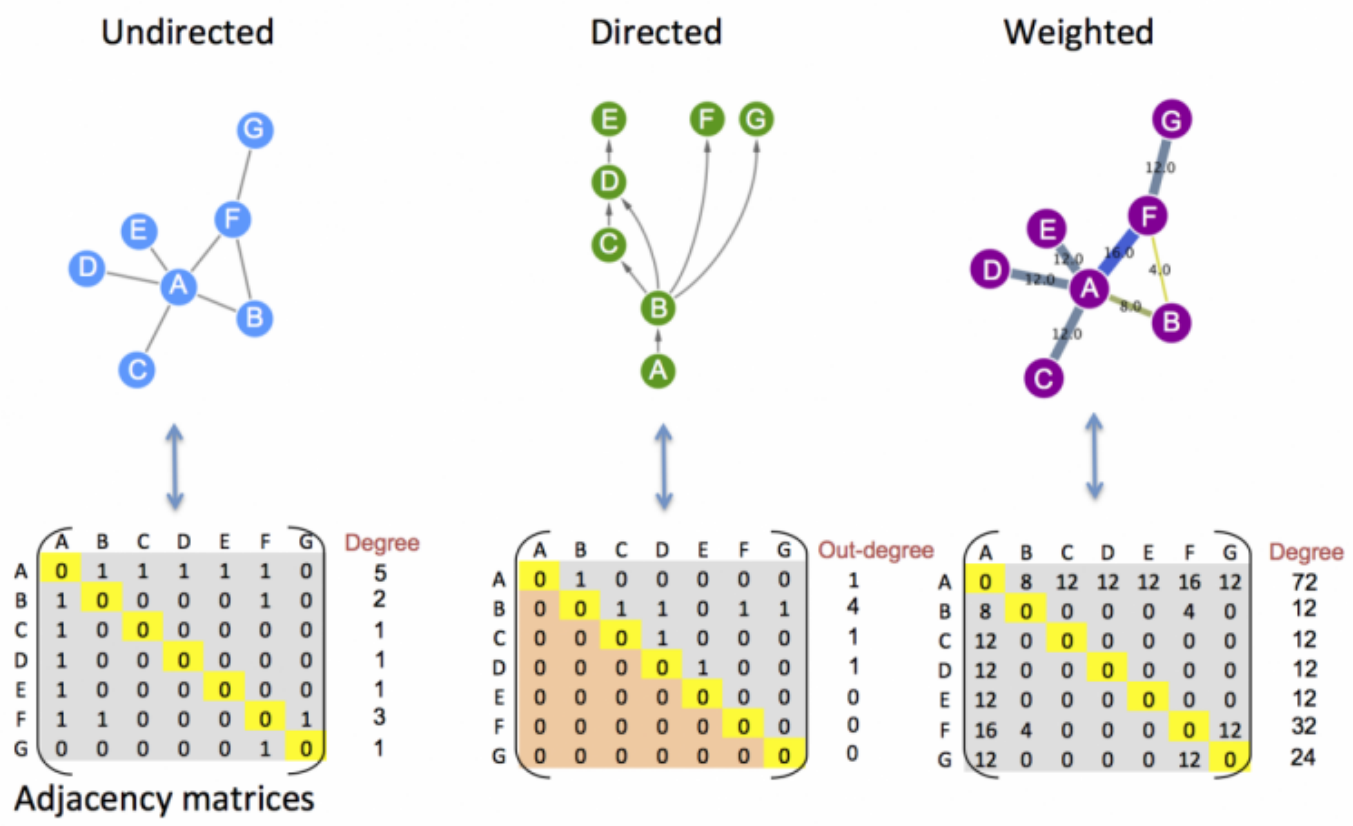
An adjacency matrix can be "weighted", which basically means each edge has an associated value attached to it, so instead of 1s, the value is put in the respective matrix coordinate. These weights can represent anything you want. In the case of molecules for example, they may represent the type of bond between two nodes (atoms). In a social

network like LinkedIn, they can represent 1st, 2nd, or 3rd, order connections between two nodes (people).

This concept of weights for edges is an attribute that makes GNNs so powerful; they allow us to take into account both **structural (dependent) and singular (independent) information.** For real world applications, this means we can consider external as well as internal information.

Degree Matrix (*D*):

The Degree Matrix of a graph can be found by using the concept of degrees covered earlier. ***D*** is essentially a diagonal matrix, where **each value of the diagonal is the degree of its corresponding node.**



The different types of graphs and Matrices (Courtesy of the EU Bioinformatics Institute)

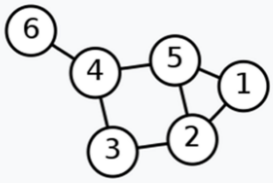
Notice that the degree is simply the sum of each row of the adjacency matrix. These degrees are then placed on the diagonal of the matrix (the line of symmetry for the adjacency matrix). This leads nicely to the final matrix:

Laplacian Matrix (L):

The Laplacian Matrix of a graph is the result of subtracting the Adjacency Matrix from the Degree Matrix:

$$L = D - A$$

Each value in the Degree Matrix is subtracted by its respective value in the Adjacency Matrix as such:

Labeled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

The graph matrix trinity (Courtesy of Wikipedia)

There exist other graph matrix representations like the **Incidence Matrix,** but the vast majority of GNN applications on graph type data utilize one, two, or all three of these matrices. This is because they, and the laplacian matrix in particular, provide substantial information about the **entities** (an element with attributes) and **relations** (a connection between entities).

The only thing missing is a **rule** (a function that maps entities to other entities via relations). This is where neural networks come in handy.

> ***If you need a bit more insight regarding graphs and their representations, I highly recommend a look at this in depth medium article.***

Deep Learning — crash course

Now let's do a quick run down of the other half of "Graph **Neural Networks**". Neural networks are the architecture we talk about when someone says "**Deep Learning**". The neural network architecture is built upon the concept of **perceptrons**, which are inspired by the neuron interactions in human brains.

Artificial Neural Networks (or just NN for short) and its extended family, including Convolutional Neural Networks, Recurrent Neural Networks, and of course, Graph Neural Networks, are all types of Deep Learning algorithms.

> Deep Learning is a type of machine learning algorithm, which in turn is a subset of artificial intelligence.

It all starts with the humble linear equation.

$$y = mx + b$$

If we structure this equation as a perceptron, we see:

Where the **output (y) is the sum (E) of the bias (b) and the input (x) times the weight (m).**

Neural networks usually have an **activation function**, which basically decides if a given neuron output (the *y*) should be considered as "activated", and keeps the output value of a perceptron within a reasonable and compute-able range. (sigmoid for 0–1, tanh for -1–1, ReLU for 0 OR 1, etc.). This is why we attach an activation function at the end of the perceptron.

When we put a bunch of perceptrons together, we get something that resembles the beginnings of a **neural network**! These perceptrons pass numerical values from one layer to another, with each pass bringing that numerical value closer to the target/label that the network is trained against.

When you put a bunch of perceptrons together, you get:

A vanilla NN (courtesy of Digital Trends)

To train a neural network, we need to first calculate how much we need to adjust the model's weights. We do this with a **loss function**, which calculates the **error**.

Where e is the error, Y is the expected output and $\hat{Y}$ is the actual output. At a high level, error is calculated as actual output (the NN's prediction) minus expected output (the target). The goal is to **minimize error**. Error is minimized by adjusting each of the layer's weights using a process known as **back propagation.**

Essentially back propagation distributes the adjustments throughout the network starting from the output layer to the input layer. The amount that is adjusted is determined by the **optimization function** which receives the error as input. The optimization function can be visualized as a ball rolling down a hill, with the ball's location being the error. Hence when the ball rolls to the bottom of the hill, the error is at its minimum.

Additionally there are some **hyperparameters** that must be defined, one of the most important of which is the **learning rate**. Learning rate adjusts the rate of which the optimization function is applied. Learning rate is like the gravity setting; the higher the gravity (higher the learning rate) the faster the ball rolls down the hill, and the same is true in reverse.

The propagation of a network (Courtesy of 3Blue1Brown)

Neural networks have many different macro and micro customization that make each model unique, with varying levels of performance, but all of them are based off of this **vanilla** model. Later we will see how this is true especially for Graph Learning. Operations like convolutions and recurrences will be introduced as needed.

Deep Learning is Graph Theory

To tie up everything we've investigated and put our knowledge to the test, we shall address the elephant in the room. If you've been paying attention, you might have noticed a subtle but glaring fact:

> Artificial Neural Networks are actually just graphs!

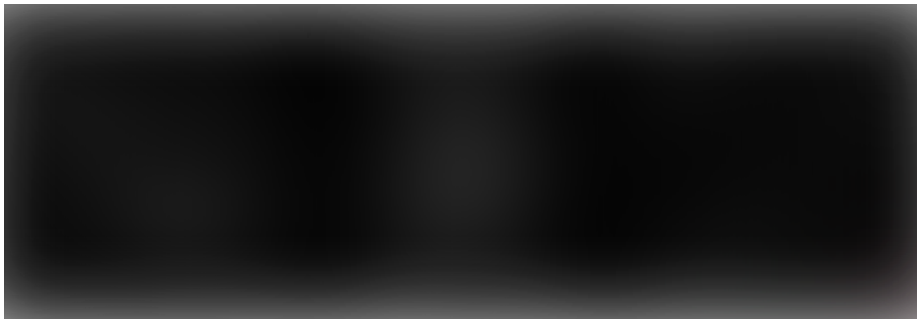
NNs are a special graph, but they have the same structure and therefore the same terminology, concepts, and rules.

Recall that the structure of a perceptron is essentially:

> *Picture of perceptron (modified)*

We can think of the input value (x), the bias value (b), and the sum operation (E) as 3 nodes in a graph. We can think of the weight (m) as an edge connecting input value (x) to sum operation (E).

The specific type of graph that NNs most resemble are **multipartite graphs.** A multipartite graph is a graph that can be separated into different sets of nodes. **The nodes in each set can share edges between sets, but not within each set.**



Isomorphic bipartite graphs (courtesy of Wolfram MathWorld)

Some neural networks even have fully connected nodes, conditional nodes, and other crazy architectures that give NNs their trademark versatility and power; here are some of the most popular architectures:

Neural Network Zoo (Courtesy of Asimov Institute)

Each color corresponds to a different type of node, which can be arranged in a multitude of different ways. Propagating data forward or backwards through the network is analogous to **message passing in graphs**. The **edge or node features** in a graph are similar to the weights in a neural network. Notice some nodes even have the **self-loops** we mentioned earlier (native to RNNs — recurrent neural networks).

Neural networks aren't the only machine learning models to have a graph-like structure.

- K-means
- K-nearest neighbors
- Decision tress

- Random forests
- Markov chains

are all structured like graphs themselves, or output data in a graph structure.

The implication therefore, is that Graph Learning models, can be used to learn from these machine learning algorithms themselves. There is a potential application in **hyperparameter optimization**. This is exactly what the authors of this amazing paper did.

The possibilities are only just beginning to surface as we learn more about generalizing deep learning on geometric data.

. . .

In Essence

We covered alot but to recap, we took a dive into 3 concepts

1. Graph Theory
2. Deep Learning

With the prerequisites in mind, one can fully understand and appreciate Graph Learning. At a high level, Graph Learning further explores and exploits the relationship between Deep Learning and Graph Theory using a family of neural networks that are designed to work on Non-Euclidean data.

Key Takeaways

There are many key takeaways, but the highlights are:

- All graphs have **properties that define the possible actions and limitations** for which it can be used or analyzed.
- **Graphs are represented computationally using various matrices**. Each matrix provides a different amount or type of information.

- **Deep learning is a subset of machine learning** that roughly mimics the way a human minds works using neurons.
- **Deep learning learns over iterations** by passing information forward through a network and propagating neuron adjustments backwards.
- Neural Networks (and other machine learning algorithms) have close ties with graph theory; **some are graphs themselves, or output them.**

Now you have all the prerequisites needed to dive into the wonderful world of Graph Learning. A good place to start would be to look into the varieties of Graph Neural Networks that have been developed thus far.

. . .

Need to see more content like this?

*Follow me on **LinkedIn, Facebook, Instagram**, and of course, **Medium** for more content.*

*All my content is on **my website** and all my projects are on **GitHub***

*I'm always looking to meet new people, collaborate, or learn something new so feel free to reach out to **flawnsontong1@gmail.com***

Upwards and onwards, always and only 🚀

Machine Learning Artificial Intelligence Entrepreneurship Technology Startup

About Help Legal