Search Medium

Write

G

Get unlimited access to all of Medium for less than $1/week. Become a member

DEMYSTIFYING GRAPH NEURAL NETWORKS — PART 1

Graph Machine Learning: An Overview

Key concepts for getting started

Zach Blumenfeld · Follow

Published in Towards Data Science · 9 min read · Apr 4

432 3

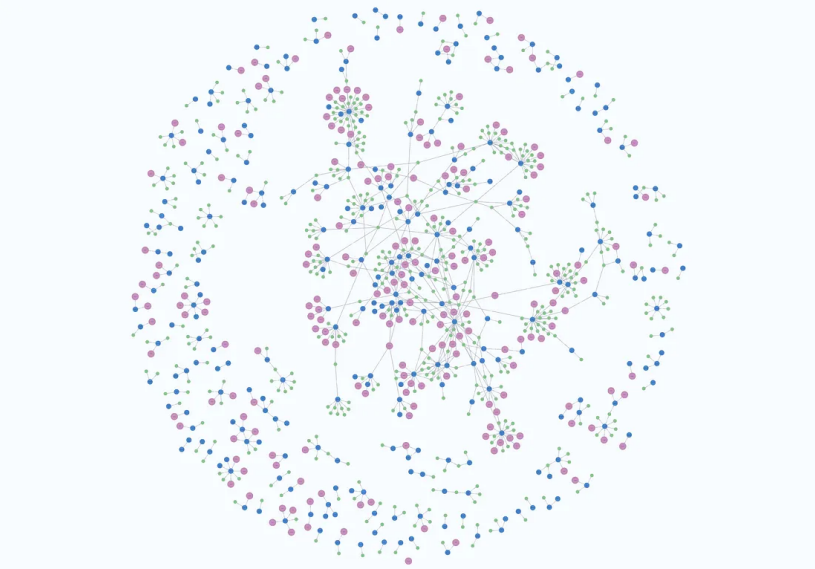


Image by Author

Graph Neural Networks (GNNs) are gaining attention in data science and machine learning but still remain poorly understood outside expert circles. To grasp this exciting approach, we must start with the broader field of Graph Machine Learning (GML). Many online resources talk about GNNs and GML as if they are interchangeable concepts or as if GNNs are a panacea approach that makes other GML approaches obsolete. This is simply not the case. One of GML's primary purposes is to compress large sparse graph data structures to enable feasible prediction and inference. GNNs are one way to accomplish this, perhaps the most advanced way, but not the only way. Understanding this will help create a better foundation for future parts of this series, where we will cover specific types of GNNs and related GML approaches in more detail.

In this post, we will:

- Go over a brief recap on graph data structures
- Cover GML tasks and the types of problems they solve
- Examine the concept of compression and its importance in driving different GML approaches, including GNNs

What are Graphs?

If you're reading this article, you likely already have some background on graph data structures. If not, I recommend reading this resource on property graphs or this resource on graph database concepts. I will give a *very* brief recap here:

A graph consists of nodes connected by relationships. There are a couple of different ways to model graph data. For simplicity, I will use the property graph model, which has three primary components:

1. **Nodes** that represent entities (sometimes called vertices),
2. **Relationships** that represent associations or interactions between nodes (sometimes called edges or links), and
3. **Properties** that represent attributes of nodes or relationships.

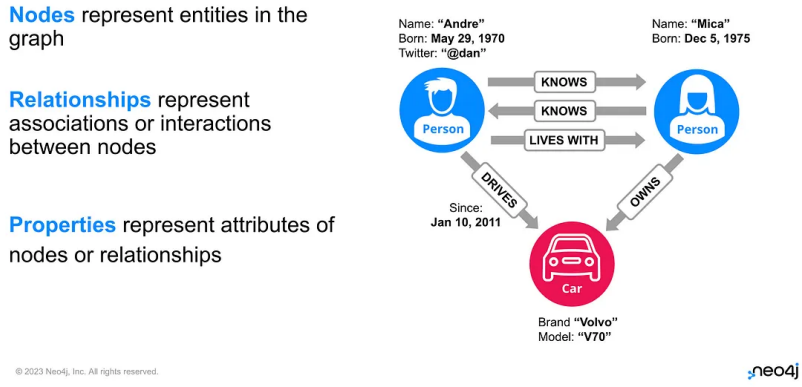


Image by Author

What is Graph Machine Learning (GML)?

At its core, Graph machine learning (GML) is the application of machine learning to graphs specifically for predictive and prescriptive tasks. GML has a variety of use cases across supply chain, fraud detection, recommendations, customer 360, drug discovery, and more.

One of the best ways to understand GML is through the different types of ML tasks it can accomplish. I break this out for supervised and unsupervised learning below.

Supervised GML Tasks

The below diagram outlines three of the most common GML tasks for supervised learning:

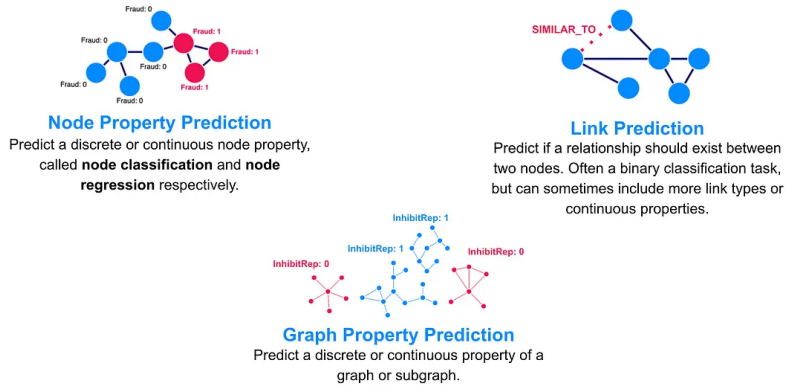


Image by Author

To expand further:

1. **Node property prediction:** Predicting a discrete or continuous node property. One can think of node property prediction as ***predicting an adjective about a thing***, such as whether an account on a financial services platform should be classified as fraud or how to categorize a product on an online retail store.
2. **Link prediction:** Predicting whether or not a relationship should exist between two nodes and potentially some properties about the relationship. Link prediction is helpful for applications like entity resolution, where we want to predict whether two nodes reflect the same underlying entity; recommendation systems where we want to predict what a user will want to purchase or interact with next; and bioinformatics, for predicting things like protein and drug interactions. For each case, we care about ***predicting an association, similarity, or potential action or interaction between entities.***
3. **Graph property prediction:** Predicting a discrete or continuous property of a graph or subgraph. Graph property prediction is useful in domains where you want to ***model each entity as an individual graph for prediction*** rather than modeling entities as nodes within a larger graph representing a complete dataset. Use cases include material sciences, bioinformatics, and drug discovery, where individual graphs can represent molecules or proteins that you want to make predictions about.

Unsupervised GML Tasks

Below are four of the most common GML tasks for unsupervised learning:

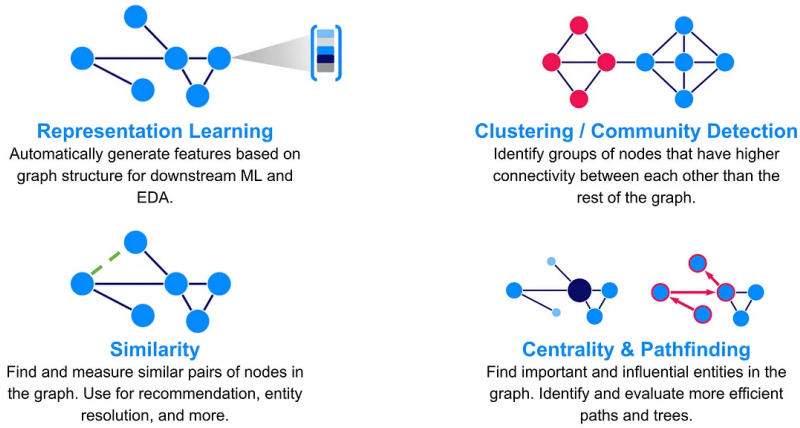


Image by Author

To elaborate on these further:

1. **Representation Learning**: Reducing dimensionality while maintaining important signals is a central theme for GML applications. Graph representation learning does this explicitly by generating low-dimensional features from graph structures, usually to use them for downstream exploratory data analysis (EDA) and ML.
2. **Community Detection (clustering for relationships):** Community detection is a clustering technique for identifying groups of densely interconnected nodes within a graph. Community detection has various practical applications in anomaly detection, fraud and investigative analytics, social network analysis, and biology.
3. **Similarity:** Similarity in GML refers to finding and measuring similar pairs of nodes in a graph. Similarity is applicable to many use cases, including recommendation, entity resolution, and anomaly and fraud detection. Common Similarity techniques include node similarity algorithms, topological link prediction, and K-Nearest-Neibor (KNN).
4. **Centrality & Pathfinding:** I'm grouping these together as they tend to be less associated with ML tasks and more with analytical measures. However, they still technically fit here, so I will cover them for completeness. Centrality finds important or influential nodes in a graph. Centrality is ubiquitous throughout many use cases, including fraud and anomaly detection, recommendation, supply chain, logistics, and infrastructure problems. Pathfinding is used to find the lowest cost paths in a graph or to evaluate the quality and availability of paths. Pathfinding can benefit many use cases dealing with physical systems such as logistics, supply chain, transportation, and infrastructure.

How Compression is Key to GML

I came across this interesting blog post by Matt Ranger which explains this point beautifully: One of the most significant objectives with GML, and to a large extent natural language processing too, is compressing large sparse data structures while maintaining important signals for prediction and inference.

Consider a regular graph represented by an adjacency matrix, a square matrix where each row and column represents a node. If a relationship goes from node A to node B, the cell at the intersection of row A and column B is 1; otherwise, it is 0. Below is an illustration of some small regular graphs and their adjacency matrices.

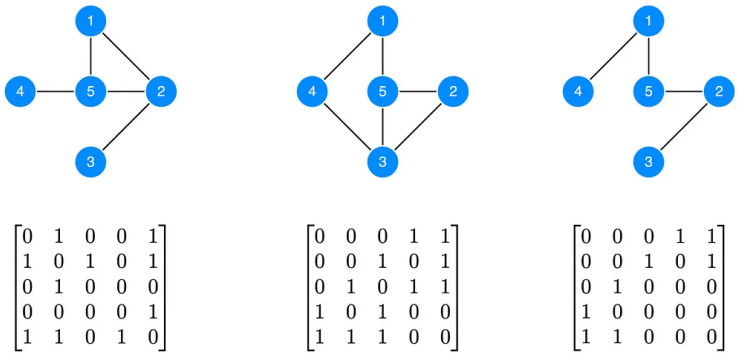
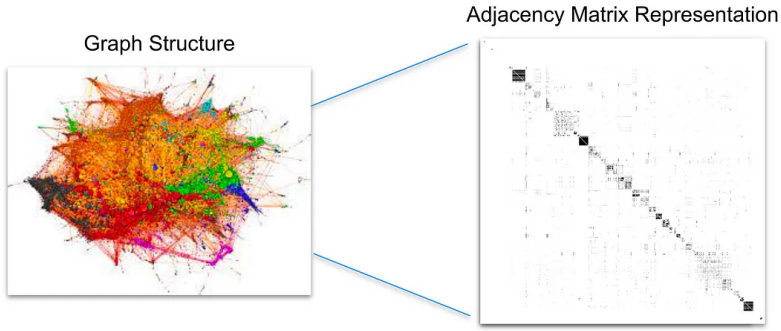


Image by Author

Notice that many of the cells in the above adjacency matrices are 0. If you scale this to large graphs, particularly those found in real-world applications, the proportion of zeros increases, and the adjacency matrix becomes mostly zeros.



Illustrative example created using Last.fm recommendation graph visual from Large Graph Visualization Tools and Approaches and matrix image from Beck, Fabian, et al. Identifying modularization patterns by visual comparison of multiple hierarchies

This happens because as these graphs grow, the average degree centrality grows much slower or not at all. In social networks, this is evidenced by concepts like the Dunbar Number, a cognitive limit to the number of people with whom one can maintain stable social relationships. You can intuit this for other types of graphs too, such as graphs of financial transactions or graphs of user purchases for recommendation systems. As these graphs grow, the number of potential unique transactions or purchases a single individual can participate in grows much faster than their capacity to do so. I.e. If there are six products on a website, one user buying half of them is feasible, but if there are hundreds of thousands, then not so much. As a result, you end up with very large and sparse data structures.

If you could use these sparse data structures directly for machine learning, you wouldn't need GNNs or any GML — you would just plug them as features into conventional ML models. However, this isn't possible. It wouldn't scale, and even beyond that, it would cause mathematical issues around convergence and estimation that would render ML models ill-specified and infeasible. As a result, a fundamental key to GML is compressing these data structures; arguably, it is the entire point of GML.

How to Accomplish Compression? — Graph Machine Learning Approaches

At the highest level, three GML approaches exist for accomplishing this compression.

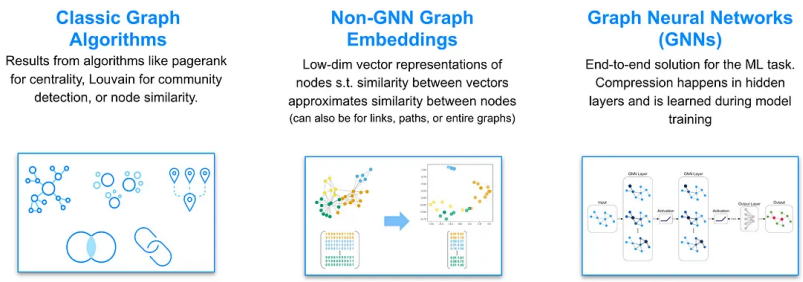


Image by Author

Classic Graph Algorithms

Classic graph algorithms include things like PageRank, Louvain, and Dijkstra's Shortest Path. They can be used independently for unsupervised community detection, similarity, centrality, or pathfinding. The results of classic algorithms can also be used as features for conventional downstream models, such as linear and logistic regressions, random forests, or neural networks to perform GML tasks.

Classic graph algorithms tend to be simple, easy to get started with, and relatively interpretable and explainable. However, they can require more manual work and subject matter expertise (SME) than other GML approaches. This makes classic graph algorithms good first choices in experimentation and development to help understand what works best on your graph. They can also do well in production for simpler problems, but more complex use cases may require graduating to another GML approach.

Non-GNN Graph Embeddings

Graph embeddings are a form of representation learning. Some graph embedding techniques leverage GNN architectures while others do not. The latter group, non-GNN, is the focus of this approach. These embedding techniques instead rely on matrix factorization/decomposition, random projections, random walks, or hashing function architectures. Some examples include Node2vec (random walk based), FastRP (random projection and matrix operations), and HashGNN (hashing function architecture).

Graph embedding involves generating numeric or binary feature vectors to represent nodes, relationships, paths, or entire graphs. The foremost of those, node embedding, is among the most fundamental and commonly used. The basic idea is to generate a vector for each node such that the similarity between vectors (e.g. dot product) approximates the similarity between nodes in the graph. Below is an illustrative example of the small Zachary's karate club network. Note how the adjacency matrix is compressed to 2-d embedding vectors for each node and how those vectors cluster together to reflect the graph community structure. Most real-world embeddings will have more than two dimensions (128 to 256 or higher) to

represent larger real-world graphs with millions or billions of nodes, but the basic intuition is the same.

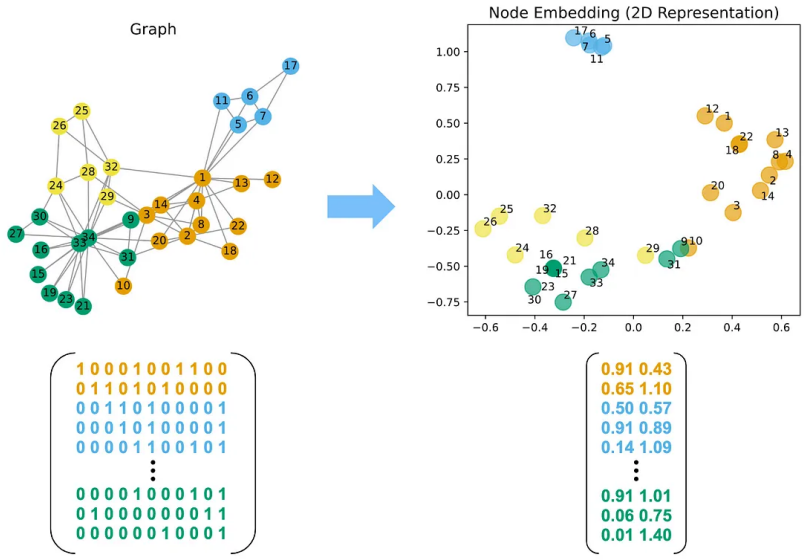


Image by Author

The same logic as above applies to relationship, path, and entire graph embeddings: similarity in the embedding vectors should approximate similarity in the graph structure. This accomplishes the compression while maintaining important signals, making the embeddings useful for various downstream ML tasks.

Non-GNN embeddings can benefit from reduced manual workload and required SME compared to classic graph algorithms. While non-GNN embeddings often require hyperparameter tuning to get right, they tend to be easier to automate and generalize across different graphs. Additionally, some non-GNN embeddings like FastRP and HashGNN can scale incredibly well to large graphs on commodity hardware since they don't require model training. This can be a massive benefit over GNN-based approaches.

However, non-GNN embeddings come with some trade-offs too. They are less interpretable and explainable than classic graph algorithms due to the more generalized mathematical operations involved. They are also generally transductive, though recent improvements in Neo4j Graph Data Science allow some of them to effectively behave inductively in certain applications. We will cover transductive and inductive settings in more depth later in this series; it has to do with the ability to predict on new unseen data and is an essential point of consideration for GML.

Graph Neural Networks (GNNs)

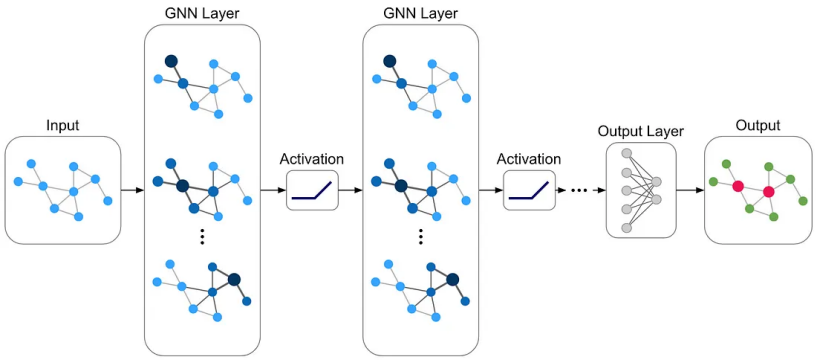


Image by Author

A GNN is a neural network model that takes graph data as input, transforms it into intermediate embeddings, and feeds the embeddings to a final layer aligned to a prediction task. This prediction task can be supervised (node property prediction, link prediction, graph property prediction) or unsupervised (clustering, similarity, or simply a final output embedding for representation learning). So unlike classic algorithms and non-GNN embeddings, which pass results as features to downstream ML models, particularly for supervised tasks, GNNs are fully end-to-end graph native solutions.

GNNs have a variety of benefits related to being complete end-to-end solutions. Notably, intermediate embeddings are learned during training and, in theory, automatically infer the most important information from the graph. Most recent GNNs are also inductive due to having a trained model.

GNNs also come with some weaknesses. This includes high complexity, scaling difficulties, and low interpretability and explainability. GNNs can also have limitations around depth due to over-smoothing and other mathematical principles.

I'll discuss GNNs more in my next blog, *GNNs: What They Are and Why They Matter*. In the meantime, **if you want to get started with graph machine learning, please take a look at Neo4j Graph Data Science**. Data scientists and engineers can find technical documentation for getting started here.

Wrapping Things Up

Biggest takeaways from this post:

- Graph Machine Learning (GML) is a broad field with many use case applications and comprising multiple different supervised and unsupervised ML tasks
- One of the primary purposes of GML is compressing large sparse graph structures while maintaining important signals for prediction and inference.
- GNNs are one of multiple GML approaches that accomplish this compression.

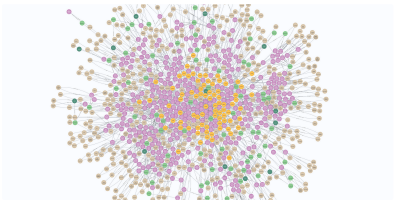
Graph Machine Learning Graph Neural Networks Data Science Machine Learning

Written by Zach Blumenfeld

227 Followers · Writer for Towards Data Science

Follow

More from Zach Blumenfeld and Towards Data Science



Zach Blumenfeld in Towards Data Science

Exploring Practical Recommendation Engines In Neo4j

Leveraging Graph Data Science & Machine Learning

16 min read · Dec 17, 2021

62 3

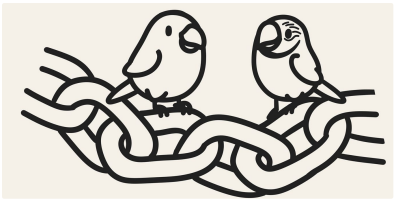
Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company's Docs into a Searchable Database with...

And how you can do the same with your docs

15 min read · Apr 25

2.2K 28



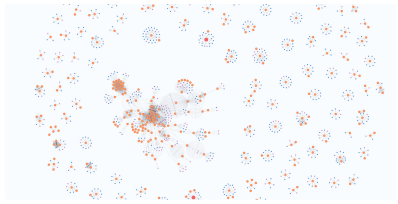
Leonie Monigatti in Towards Data Science

Getting Started with LangChain: A Beginner's Guide to Building LLM...

A LangChain tutorial to build anything with large language models in Python

· 12 min read · Apr 25

1.2K 13



Zach Blumenfeld in Neo4j Developer Blog

Exploring Fraud Detection With Neo4j & Graph Data Science—...

A Multi-Part Series with Hands on Examples Using Peer-to-Peer (P2P) Data

3 min read · Mar 1, 2022

12

See all from Zach Blumenfeld

See all from Towards Data Science

Recommended from Medium

Leonie Monigatti in Towards Data Science

Getting Started with LangChain: A Beginner's Guide to Building LLM...

A LangChain tutorial to build anything with large language models in Python

· 12 min read · Apr 25

1.2K 13

Dennis Ganzaroli in MLearning.ai

The Best kept Secret in Data Science is KNIME

Discover KNIME, the best kept secret in data science. This powerful and versatile open...

· 14 min read · Feb 2

406 4

Lists

What is ChatGPT?
9 stories · 26 saves

Stories to Help You Level-Up at Work
19 stories · 20 saves

Staff Picks
302 stories · 62 saves

Erdogan Taskesen in Towards Data Science

From Clusters To Insights; The Next Step

Learn how to quantitatively detect which features drive the formation of the clusters

· 9 min read · 5 days ago

315

Matt Chapman in Towards Data Science

How I Stay Up to Date With the Latest AI Trends as a Full-Time...

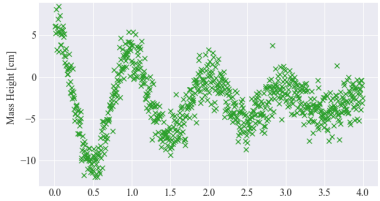
No, I don't just ask ChatGPT to tell me

· 8 min read · May 1

869 18

Arthur Mello in Level Up Coding

Exploratory Data Analysis: The Ultimate Workflow



Nick Hemenway in Python in Plain English

My Favorite Way to Smooth Noisy Data With Python

Explore the true potential of your data with Python

· 16 min read · Apr 20

678 5

Nearly all real-world data is noisy. What do I mean by noisy? Consider the following simpl...

· 7 min read · Dec 4, 2022

174 2

See more recommendations