

Graph Neural Networks (GNNs)

Nguyễn Quân Bá Hồng*

Ngày 6 tháng 8 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:

URL: https://nqbh.github.io/advanced_STEM/.

Latest version:

- .
PDF: URL: [.pdf](#).
T_EX: URL: [.tex](#).
- .
PDF: URL: [.pdf](#).
T_EX: URL: [.tex](#).

Mục lục

| | | |
|----------|--|-----------|
| 1 | Introduction to Graph Neural Networks | 1 |
| 1.1 | KEITA BROADWATER, NAMID STILLMANN. Graph Neural Networks in Action | 1 |
| 1.2 | MAXIME LABONNE. Hands-On Graph Neural Networks Using Python: Practical Techniques & Architectures for Building Powerful Graph & DL Apps with PyTorch. 2023 | 33 |
| 2 | Miscellaneous | 60 |

1 Introduction to Graph Neural Networks

1.1 KEITA BROADWATER, NAMID STILLMANN. Graph Neural Networks in Action

- Fig: Mental model of GNN project. The steps involved for a GNN project are similar to many conventional ML pipelines, but we need to use graph-specific tools to create them. Start with raw data, which is then transformed into a graph data model & that can be stored in a graph database or used in a graph processing system. From the graph processing system (& some graph database), we can do exploratory data analysis & visualization. Finally, for graph ML, we preprocess data into a format that can be submitted for training & then train our graph ML model, in our examples, these will be GNNs.

$\boxed{\text{GNNs} \subset \text{Graph ML models}}$.

– Hình: Mô hình tinh thần của dự án GNN. Các bước cần thực hiện cho 1 dự án GNN tương tự như nhiều quy trình ML thông thường, nhưng chúng ta cần sử dụng các công cụ dành riêng cho đồ thị để tạo ra chúng. Bắt đầu với dữ liệu thô, sau đó được chuyển đổi thành mô hình dữ liệu đồ thị & có thể được lưu trữ trong cơ sở dữ liệu đồ thị hoặc sử dụng trong hệ thống xử lý đồ thị. Từ hệ thống xử lý đồ thị (& 1 số cơ sở dữ liệu đồ thị), chúng ta có thể thực hiện phân tích dữ liệu thăm dò & trực quan hóa. Cuối cùng, đối với ML đồ thị, chúng ta xử lý trước dữ liệu thành 1 định dạng có thể được gửi để huấn luyện & sau đó huấn luyện mô hình ML đồ thị của chúng ta, trong các ví dụ của chúng ta, đây sẽ là các GNN. $\boxed{\text{GNN} \subset \text{Mô hình ML đồ thị}}$.

- Foreword. Our world is highly rich in structure, comprising objects, their relations, & hierarchies. Sentences can be represented as sequences of words, maps can be broken down into streets & intersections, www connects websites via hyperlinks, & chemical compounds can be described by a set of atoms & their interactions. Despite prevalence of graph structures in our world, both traditional & even modern ML methods struggle to properly handle such rich structural information: ML conventionally expects fixed-sized vectors as inputs & is thus only applicable to simpler structures e.g. sequences or grids. Consequently, graph ML has long relied on labor-intensive & error-prone handcrafted feature engineering techniques. Graph neural networks (GNNs) finally revolutionize this paradigm by breaking up with regularity restriction of conventional DL techniques. They unlock ability to learn representations from raw graph data with exceptional performance & allow us to view DL as a much broader technique that can seamlessly generalize to complex & rich topological structures.

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: <https://nqbh.github.io/>. GitHub: <https://github.com/NQBH>.

– Thế giới của chúng ta vô cùng phong phú về cấu trúc, bao gồm các đối tượng, mối quan hệ của chúng, & hệ thống phân cấp. Cấu trúc có thể được biểu diễn dưới dạng chuỗi từ, bản đồ có thể được chia nhỏ thành các con phố & giao lộ, www kết nối các trang web thông qua siêu liên kết, & hợp chất hóa học có thể được mô tả bằng 1 tập hợp các nguyên tử & tương tác của chúng. Mặc dù cấu trúc đồ thị rất phổ biến trong thế giới của chúng ta, cả các phương pháp ML truyền thống & thậm chí hiện đại đều gặp khó khăn trong việc xử lý đúng cách thông tin cấu trúc phong phú như vậy: ML thường mong đợi các vectơ có kích thước cố định làm đầu vào & do đó chỉ áp dụng cho các cấu trúc đơn giản hơn, ví dụ như chuỗi hoặc lưới. Do đó, ML đồ thị từ lâu đã dựa vào các kỹ thuật thiết kế đặc trưng thủ công tốn nhiều công sức & dễ xảy ra lỗi. Mạng nơ-ron đồ thị (GNN) cuối cùng đã cách mạng hóa mô hình này bằng cách phá vỡ sự hạn chế về quy tắc của các kỹ thuật DL thông thường. Chúng mở khóa khả năng học các biểu diễn từ dữ liệu đồ thị thô với hiệu suất vượt trội & cho phép chúng ta xem DL như 1 kỹ thuật rộng hơn nhiều, có thể khái quát hóa liên mạch thành các cấu trúc tô pô phức tạp & phong phú.

When MATTHIAS FEY – creator of PyTorch Geometric & founding engineer Kumo.AI – begin to dive into field of graph ML, DL on graphs was still in its early stages. Over time, dozens to hundreds of different methods were developed, contributing incremental insights & refreshing ideas. Tools like our own PyTorch Geometric library have expanded significantly, offering cutting-edge graph-based building blocks, models, examples, & scalability solutions. Reflecting on this growth, it is clear how overwhelming it can be for newcomers to navigate essentials & best practices that have emerged over time, as valuable information is scattered across theoretical research papers or buried in implementations in GitHub repositories.

– Khi MATTHIAS FEY – người sáng lập PyTorch Geometric & kỹ sư sáng lập Kumo.AI – bắt đầu dấn thân vào lĩnh vực học máy đồ thị, học máy trên đồ thị vẫn còn ở giai đoạn sơ khai. Theo thời gian, hàng chục đến hàng trăm phương pháp khác nhau đã được phát triển, đóng góp những hiểu biết sâu sắc & những ý tưởng mới mẻ. Các công cụ như thư viện PyTorch Geometric của chúng tôi đã mở rộng đáng kể, cung cấp các khối xây dựng, mô hình, ví dụ, & giải pháp khả năng mở rộng dựa trên đồ thị tiên tiến. Nhìn lại sự phát triển này, rõ ràng là những người mới bắt đầu có thể gặp khó khăn như thế nào khi tìm hiểu những điều cốt lõi & các phương pháp hay nhất đã xuất hiện theo thời gian, khi thông tin giá trị nằm rải rác trong các bài báo nghiên cứu lý thuyết hoặc bị chôn vùi trong các triển khai trên kho lưu trữ GitHub.

Now power of GNNs has been widely understood, this timely book provides a well-structured & easy-to-follow overview of field, providing answers to many pain points of graph ML practitioners. Hands-on approach, with practical code examples embedded directly within each chap, invaluable demystifies complexities, making concepts tangible & actionable. Despite success of GNNs across all kinds of domains in research, adoption in real-world applications remains limited to companies that have enough resources to acquire necessary knowledge for applying GNNs in practice. Confident: this book will serve as an invaluable resource to empower practitioners to over that gap & unlock full potentials of GNNs.

– Giờ đây, sức mạnh của GNN đã được hiểu rộng rãi, cuốn sách kịp thời này cung cấp 1 cái nhìn tổng quan được cấu trúc tốt & dễ hiểu về lĩnh vực này, giải đáp nhiều vấn đề khó khăn của các chuyên gia ML đồ thị. Phương pháp tiếp cận thực hành, với các ví dụ mã thực tế được nhúng trực tiếp trong mỗi chương, giúp làm sáng tỏ những điều phức tạp, biến các khái niệm thành hiện thực & khả thi. Mặc dù GNN đã thành công trong nhiều lĩnh vực nghiên cứu, việc áp dụng vào các ứng dụng thực tế vẫn chỉ giới hạn ở các công ty có đủ nguồn lực để có được kiến thức cần thiết cho việc áp dụng GNN vào thực tế. Tự tin: cuốn sách này sẽ là 1 nguồn tài nguyên vô giá giúp các chuyên gia vượt qua khoảng cách đó & khai phá toàn bộ tiềm năng của GNN.

- Preface. My journey into world of graphs began unexpectedly, during an interview at LinkedIn. As session wrapped up, shown a visualization of network – a mesmerizing structure that told stories without a single word. Organizations I had been part of appeared clustered, like constellations against a dark canvas. What surprised me most was that this structure was not built using metadata LinkedIn held about my connection; rather, it emerged organically from relationships between nodes & edges.

– Hành trình khám phá thế giới đồ thị của tôi bắt đầu 1 cách bất ngờ, trong 1 buổi phỏng vấn tại LinkedIn. Khi buổi phỏng vấn kết thúc, 1 hình ảnh trực quan về mạng lưới được trình chiếu - 1 cấu trúc mê hoặc kể những câu chuyện mà không cần 1 lời nào. Các tổ chức mà tôi từng là thành viên hiện ra như những chòm sao trên nền vải tối. Điều khiến tôi ngạc nhiên nhất là cấu trúc này không được xây dựng bằng siêu dữ liệu mà LinkedIn nắm giữ về kết nối của tôi; thay vào đó, nó xuất hiện 1 cách tự nhiên từ các mối quan hệ giữa các nút & cạnh.

Years later, driven by curiosity, I recreated that visualization. I marveled once again at how underlying connections along could map out an intricate picture of my professional life. This deepened my appreciation for power inherent in graphs – a fascination that only grew when I joined Cloudera & encountered graph neural networks (GNNs). Their potential for solving complex problems was captivating, but diving into them was like trying to navigate an uncharted forest without a map. There were no comprehensive resources tailored for nonacademics; progress was slow, often cobbled together from fragments & trial & error.

– Nhiều năm sau, nhờ sự tò mò, tôi đã tái hiện lại hình ảnh đó. Tôi lại 1 lần nữa kinh ngạc trước cách các kết nối cơ bản có thể vẽ nên 1 bức tranh phức tạp về cuộc sống nghề nghiệp của mình. Điều này càng làm tôi trân trọng hơn sức mạnh tiềm ẩn của đồ thị – 1 niềm đam mê chỉ lớn dần khi tôi gia nhập Cloudera & gặp gỡ mạng nơ-ron đồ thị (GNN). Tiềm năng giải quyết các vấn đề phức tạp của chúng thật hấp dẫn, nhưng việc đào sâu vào chúng cũng giống như cố gắng khám phá 1 khu rừng chưa được khám phá mà không có bản đồ. Không có tài nguyên toàn diện nào được thiết kế riêng cho những người không chuyên; tiến độ rất chậm, thường được chắp vá từ những mảnh & thử & sai.

This book is guide I wish I had during those early days. It aims to provide a clear & accessible path for practitioners, enthusiasts, & anyone looking to understand & apply GNNs without wading through endless academic papers or fragmented online searches. Hop: it serves as a 1-stop resource to learn fundamentals & paves way for deeper exploration. Whether you

are here out of professional necessity, sheer curiosity, or same kind of amazement that 1st drew me in, invite to embark on this journey, bring potential of GNNs to life.

– Cuốn sách này chính là cầm nang mà tôi ước mình đã có trong những ngày đầu ấy. Nó hướng đến việc cung cấp 1 lộ trình rõ ràng & dễ tiếp cận cho các chuyên gia, người đam mê, & bất kỳ ai muốn hiểu & áp dụng GNN mà không cần phải lội qua vô số bài báo học thuật hay tìm kiếm trực tuyến rời rạc. Hoped: nó là 1 nguồn tài nguyên tổng hợp để học các kiến thức cơ bản & mở đường cho những khám phá sâu hơn. Cho dù bạn đến đây vì nhu cầu công việc, sự tò mò đơn thuần, hay cùng 1 sự ngạc nhiên như đã thu hút tôi lần đầu, tham gia vào hành trình này, khai phá tiềm năng của GNN.

- **About this book.** GNNs in Action is a book designed for people to jump quickly into this new field & start building applications. At same time, try to strike a balance by including just enough critical theory to make this book as standalone as possible. Also fill in implementation details that may not be obvious or are left unexplained in currently available online tutorials & documents. In particular, information about new & emerging topics is very likely to be fragmented. This fragmentation adds friction when implementing & testing new technologies.

– GNNs in Action là 1 cuốn sách được thiết kế để mọi người có thể nhanh chóng bước vào lĩnh vực mới này & bắt đầu xây dựng ứng dụng. Đồng thời, cố gắng cân bằng bằng cách đưa vào vừa đủ lý thuyết quan trọng để cuốn sách này trở nên độc lập nhất có thể. Đồng thời, bổ sung những chi tiết triển khai có thể chưa rõ ràng hoặc chưa được giải thích trong các hướng dẫn trực tuyến hiện có & tài liệu. Đặc biệt, thông tin về các chủ đề mới & đang nổi lên rất có thể sẽ bị phân mảnh. Sự phân mảnh này gây khó khăn khi triển khai & thử nghiệm các công nghệ mới.

With GNNs in Action, offer a book that can reduce that friction by filling in gaps & answering key questions whose answers are likely scattered over internet or not covered at all. Done so in a way that emphasizes approachability rather than high rigor.

– Với GNNs in Action, cung cấp 1 cuốn sách có thể giảm thiểu sự khó khăn đó bằng cách lấp đầy những khoảng trống & trả lời những câu hỏi quan trọng mà câu trả lời có thể nằm rải rác trên internet hoặc chưa được đề cập đến. Hãy làm điều này theo cách nhấn mạnh tính dễ tiếp cận hơn là tính nghiêm ngặt cao.

- **Who should read this book.** This book is designed for ML engineers & data scientists familiar with neural networks but new to graph learning. If have experience in OOP, find concepts particularly accessible & applicable.

– Cuốn sách này được thiết kế dành cho các kỹ sư ML & nhà khoa học dữ liệu đã quen thuộc với mạng nơ-ron nhưng chưa quen với học đồ thị. Nếu có kinh nghiệm về OOP, tìm các khái niệm đặc biệt dễ hiểu & áp dụng.

- **How this book is organized: A road map.** In Part 1 of this book, provide a motivation for exploring GNNs, as well as cover fundamental concepts of graphs & graph-based ML. In Chap. 1, introduce concepts of graphs & graph ML, providing guidelines for their use & applications. Chap. 2 covers graph representations up to & including node embeddings. This will be 1st programmatic exposure to GNNs, which are used to create such embeddings.

– Trong Phần 1 của cuốn sách này, chúng tôi sẽ cung cấp động lực để khám phá GNN, cũng như đề cập đến các khái niệm cơ bản về đồ thị & Học máy dựa trên đồ thị. Trong Chương 1, chúng tôi sẽ giới thiệu các khái niệm về đồ thị & Học máy dựa trên đồ thị, đồng thời cung cấp hướng dẫn sử dụng & ứng dụng của chúng. Chương 2 sẽ đề cập đến các biểu diễn đồ thị, bao gồm cả nhúng nút. Đây sẽ là lần đầu tiên chúng tôi tiếp xúc với GNN, được sử dụng để tạo ra các nhúng như vậy.

In part 2, core of book, introduce major types of GNNs, including graph convolutional networks (GCNs) & GraphSAGE in Chap. 3, graph attention networks (GATs) in Chap. 4, & graph autoencoders (GAEs) in Chap. 5. These methods are bread & butter for most GNN applications & also cover a range of other DL concepts e.g. convolution, attention, & autoencoders.

– Trong phần 2, cốt lõi của cuốn sách, giới thiệu các loại GNN chính, bao gồm mạng tích chập đồ thị (GCN) & GraphSAGE trong Chương 3, mạng chú ý đồ thị (GAT) trong Chương 4, & bộ mã hóa tự động đồ thị (GAE) trong Chương 5. Các phương pháp này là nền tảng cho hầu hết các ứng dụng GNN & cũng bao gồm 1 loạt các khái niệm DL khác, e.g., tích chập, chú ý, & bộ mã hóa tự động.

In part 3, look at more advanced topics. Describe GNNs for dynamic graphs (spatio-temporal GNNs) in Chap. 6 & give methods to train GNNs at scale in Chap. 7. Finally, end with some consideration for project & system planning for graph learning projects in Chap. 8.

– Trong phần 3, xem xét các chủ đề nâng cao hơn. Mô tả GNN cho đồ thị động (GNN không gian-thời gian) trong Chương 6 & đưa ra các phương pháp huấn luyện GNN ở quy mô lớn trong Chương 7. Cuối cùng, kết thúc bằng 1 số cân nhắc về dự án & lập kế hoạch hệ thống cho các dự án học đồ thị trong Chương 8.

- **About code.** Python is coding language of choice throughout this book. There are now several GNN libraries in Python ecosystem, including PyTorch Geometric (PyG), Deep Graph Library (DGL), GraphScope, & Jraph. Focus on PyG, which is 1 of most popular & easy-to-use frameworks, written on top of PyTorch. Want this book to be approachable by an audience with a wide set of hardware constraints, so with exception of some individual sects & Chap. 7 on scalability, distributed systems & GPU systems aren't required, although they can be used for some of coded examples.

– Python là ngôn ngữ lập trình được lựa chọn trong suốt cuốn sách này. Hiện nay, hệ sinh thái Python đã có 1 số thư viện GNN, bao gồm PyTorch Geometric (PyG), Thư viện Đồ thị Sâu (DGL), GraphScope, Jraph. Tập trung vào PyG, 1 trong những framework phổ biến nhất, dễ sử dụng, được viết trên nền tảng PyTorch. Tôi muốn cuốn sách này dễ tiếp cận với những độc giả có nhiều hạn chế về phần cứng, vì vậy, ngoại trừ 1 số điểm riêng biệt trong Chương 7 về khả năng mở rộng, các hệ thống phân tán & GPU không bắt buộc, mặc dù chúng có thể được sử dụng cho 1 số ví dụ được mã hóa.

Book provides a survey of most relevant implementations of GNNs, including graph convolutional networks (GCNs), graph autoencoders (GAEs), graph attention networks (GATs), & graph long short-term memory (LSTM). Aim: cover GNN tasks mentioned earlier. In addition, touch on different types of graphs, including knowledge graphs.

– Sách cung cấp 1 bản tổng quan về các triển khai GNN phổ biến nhất, bao gồm mạng tích chập đồ thị (GCN), bộ mã hóa tự động đồ thị (GAE), mạng chú ý đồ thị (GAT), bộ nhớ dài hạn đồ thị (LSTM). Mục tiêu: bao quát các nhiệm vụ GNN đã đề cập trước đó. Ngoài ra, đề cập đến các loại đồ thị khác nhau, bao gồm đồ thị tri thức.

This book contains many examples of source code both in numbered listings & in line with normal text. In both case, source code is formatted in a **fixed-width font like this** to separate it from ordinary text. Sometimes code is also **in bold** to highlight code

PART 1: 1ST STEPS. Graphs are 1 of most versatile & powerful ways to represent complex, interconnected data. This 1st part introduces fundamental concepts of graph theory, explaining what graphs are, why they matter as a data type, & how their structure captures relationships that traditional data formats miss. Explore building blocks of graphs & different graph types.

– Đồ thị là 1 trong những phương pháp linh hoạt & mạnh mẽ nhất để biểu diễn dữ liệu phức tạp, có liên kết với nhau. Phần 1 này giới thiệu các khái niệm cơ bản của lý thuyết đồ thị, giải thích đồ thị là gì, tại sao chúng quan trọng như 1 kiểu dữ liệu, & cách cấu trúc của chúng nắm bắt các mối quan hệ mà các định dạng dữ liệu truyền thống bỏ sót. Khám phá các khối xây dựng của đồ thị & các kiểu đồ thị khác nhau.

Explore fundamental concepts about GNNs, beginning with what they are & how they differ from traditional neural networks. With this foundation, study graph embeddings, uncovering how to represent graphs in a way that makes them useful for ML. These concepts set stage for mastering GNNs & their transformative capabilities in later chaps. By end of this book, have a solid understanding of basics, preparing you to dive deeper into mechanics of GNNs.

– Khám phá các khái niệm cơ bản về GNN, bắt đầu với bản chất của chúng & sự khác biệt so với mạng nơ-ron truyền thống. Với nền tảng này, nghiên cứu nhúng đồ thị, khám phá cách biểu diễn đồ thị sao cho hữu ích cho ML. Những khái niệm này đặt nền tảng cho việc nắm vững GNN & khả năng biến đổi của chúng trong các chương sau. Khi đọc xong cuốn sách này, bạn sẽ có được kiến thức cơ bản vững chắc, sẵn sàng cho việc tìm hiểu sâu hơn về cơ chế hoạt động của GNN.

- 1. Discovering graph neural networks. Covers:

- Defining graphs & GNNs
- Understanding why people are excited about GNNs
- Recognizing when to use GNNs
- Taking a big picture look at solving a problem with a GNN

For data practitioners, fields of ML & DS initially excite us because of potential to draw nonintuitive & useful insights from data. In particular, insights from ML & DL promise to enhance our understanding of world. For working engineer, these tools promise to deliver business value in unprecedented ways.

– Đối với các chuyên gia dữ liệu, lĩnh vực Học máy & Phân tích dữ liệu (ML & DS) ban đầu khiến chúng ta hào hứng vì tiềm năng rút ra những hiểu biết phi trực quan & hữu ích từ dữ liệu. Đặc biệt, những hiểu biết từ Học máy & Phân tích dữ liệu (ML & DL) hứa hẹn sẽ nâng cao hiểu biết của chúng ta về thế giới. Đối với các kỹ sư đang làm việc, những công cụ này hứa hẹn mang lại giá trị kinh doanh theo những cách chưa từng có.

Experience deviates from this ideal. Real-world data is usually messy, dirty, & biased. Furthermore, statistical methods & learning systems come with their own set of limitations. An essential role of practitioners: comprehend these limitations & bridge gap between real data & a feasible solution. E.g., may want to predict fraudulent activity in a bank, but 1st need to make sure that our training data has been correctly labeled. Even more importantly, need to check that our models won't incorrectly assign fraudulent activity to normal behaviors, possibly due to some hidden confounders in data.

– Kinh nghiệm thực tế thường khác xa lý tưởng này. Dữ liệu thực tế thường lộn xộn, bẩn thỉu, & thiên vị. Hơn nữa, các phương pháp thống kê & hệ thống học tập cũng có những hạn chế riêng. 1 vai trò thiết yếu của người thực hành: hiểu rõ những hạn chế này & thu hẹp khoảng cách giữa dữ liệu thực tế & 1 giải pháp khả thi. Ví dụ: có thể muốn dự đoán hoạt động gian lận trong 1 ngân hàng, nhưng trước tiên cần đảm bảo rằng dữ liệu đào tạo của chúng ta đã được dán nhãn chính xác. Quan trọng hơn nữa, cần kiểm tra xem các mô hình của chúng ta có gán sai hoạt động gian lận cho các hành vi bình thường hay không, có thể do 1 số yếu tố gây nhiễu tiềm ẩn trong dữ liệu.

For graph data, until recently, bridging this gap has been particularly challenging. Graphs are a data structure that is rich with information & especially adept at capturing intricacies of data where relationships play a crucial role. Graphs are omnipresent, with relationship data appearing in different forms e.g. atoms in molecules (nature), social networks (society), & even models connection of web pages on internet (technology). Important to note: term *relational* here does not refer to *relational databases*, but rather to data where relationships are of significance.

– Đối với dữ liệu đồ thị, cho đến gần đây, việc thu hẹp khoảng cách này đặc biệt khó khăn. Đồ thị là 1 cấu trúc dữ liệu giàu thông tin & đặc biệt khéo léo trong việc nắm bắt những dữ liệu phức tạp, nơi các mối quan hệ đóng vai trò then chốt. Đồ thị hiện diện ở khắp mọi nơi, với dữ liệu mối quan hệ xuất hiện dưới nhiều dạng khác nhau, ví dụ: nguyên tử trong phân tử (tự nhiên), mạng xã hội (xã hội), & thậm chí cả mô hình kết nối các trang web trên internet (công nghệ). Điều quan trọng cần

lưu ý: thuật ngữ *relational* ở đây không đề cập đến *relational databases*, mà là dữ liệu trong đó các mối quan hệ có ý nghĩa quan trọng.

Previously, if you wanted to incorporate relational features from a graph into a DL model, it had to be done in an indirect way, with different models used to process, analyze, & then use graph data. These separate models often couldn't be easily scaled & had trouble taking into account all node & edge properties of graph data. To make best use of this rich & ubiquitous data type for ML, needed a specialized ML technique specifically designed for distinct qualities of graphs & relational data. This is gap that GNNs fill.

– Trước đây, nếu muốn tích hợp các đặc điểm quan hệ từ đồ thị vào mô hình DL, việc này phải được thực hiện gián tiếp, sử dụng các mô hình khác nhau để xử lý, phân tích, & sau đó sử dụng dữ liệu đồ thị. Các mô hình riêng biệt này thường không dễ dàng mở rộng & gặp khó khăn trong việc tính đến tất cả các thuộc tính nút & cạnh của dữ liệu đồ thị. Để tận dụng tối đa kiểu dữ liệu phong phú & phổ biến này cho ML, cần có 1 kỹ thuật ML chuyên biệt được thiết kế riêng cho các đặc tính riêng biệt của đồ thị & dữ liệu quan hệ. Đây chính là khoảng trống mà GNN lấp đầy.

DP field often contains a lot of hype around new technologies & methods. However, GNNs are widely recognized as a genuine leap forward for graph-based learning [2]. This does not mean: GNNs are a silver bullet. Careful comparisons should be done between predictive results derived from GNNs & other ML & DL methods.

– Lĩnh vực DP thường chứa đựng nhiều thông tin cường điệu về các công nghệ & phương pháp mới. Tuy nhiên, GNN được công nhận rộng rãi là 1 bước tiến thực sự cho học tập dựa trên đồ thị [2]. Điều này không có nghĩa là: GNN là giải pháp hoàn hảo. Cần so sánh cẩn thận giữa các kết quả dự đoán thu được từ GNN & các phương pháp ML & DL khác.

Key thing to remember: if your DS problem involves data that can be structured as a graph – i.e., data is connected or relational – then GNNs could offer a valuable approach, even if you weren't aware that sth was missing in your approach. GNNs can be designed to handle very large data, to scale, to adapt to graphs of different sizes & shapes. This can make working with relationship-centric data easier & more efficient, as well as yield richer results.

– Điều quan trọng cần nhớ: nếu bài toán DS của bạn liên quan đến dữ liệu có thể được cấu trúc dưới dạng đồ thị - tức là dữ liệu được kết nối hoặc quan hệ - thì GNN có thể cung cấp 1 phương pháp tiếp cận hữu ích, ngay cả khi bạn không nhận ra rằng phương pháp của mình còn thiếu điều gì đó. GNN có thể được thiết kế để xử lý dữ liệu rất lớn, có khả năng mở rộng, thích ứng với các đồ thị có kích thước & hình dạng khác nhau. Điều này có thể giúp việc xử lý dữ liệu tập trung vào mối quan hệ dễ dàng & hiệu quả hơn, cũng như mang lại kết quả phong phú hơn.

Standout advantages of GNNs are why data scientists & engineers are increasingly recognizing importance of mastering them. GNNs have the ability to unveil unique insights from relational data – from identifying new drug candidates to optimizing ETA prediction accuracy in your Google Maps app – acting as a catalyst for discovery & innovation, & empowering professionals to push boundaries of conventional data analysis. Their diverse applicability spans various fields, offering professionals a versatile tool that is as relevant in e-commerce (e.g., recommendation engines) as it is in bioinformatics (e.g., drug toxicity prediction). Proficiency in GNNs equips data professionals with a multifaceted tool for enhanced, accurate, & innovative data analysis of graphs.

– Những lợi thế nổi bật của GNN là lý do tại sao các nhà khoa học dữ liệu & kỹ sư ngày càng nhận thức được tầm quan trọng của việc thành thạo chúng. GNN có khả năng khám phá những hiểu biết độc đáo từ dữ liệu quan hệ – từ việc xác định các ứng cử viên thuốc mới đến tối ưu hóa độ chính xác dự đoán ETA trong ứng dụng Google Maps – đóng vai trò là chất xúc tác cho khám phá & đổi mới, & trao quyền cho các chuyên gia vượt qua các giới hạn của phân tích dữ liệu thông thường. Khả năng ứng dụng đa dạng của chúng trải rộng trên nhiều lĩnh vực, mang đến cho các chuyên gia 1 công cụ đa năng, vừa phù hợp trong thương mại điện tử (ví dụ: công cụ đề xuất) vừa phù hợp trong tin sinh học (ví dụ: dự đoán độc tính của thuốc). Thành thạo GNN trang bị cho các chuyên gia dữ liệu 1 công cụ đa năng để phân tích dữ liệu đồ thị chính xác, & sáng tạo.

For all these reasons, GNNs are now popular choice for recommender engines, analyzing social networks, detecting fraud, understanding how biomolecules behave, & many other practical examples.

– Vì tất cả những lý do này, GNN hiện là lựa chọn phổ biến cho các công cụ đề xuất, phân tích mạng xã hội, phát hiện gian lận, hiểu cách các phân tử sinh học hoạt động, & nhiều ví dụ thực tế khác.

◦ 1.1. Goals of this book. GNNs in Action is aimed at practitioners who want to begin to deploy GNNs to solve real problems. This could be a ML engineer not familiar with graph data structures, a data scientist who hasn't yet tried GNNs, or even a software engineer who may be unfamiliar with either. Throughout this book, cover topics from basics of graphs all way to more complex GNN models. Build up architecture of a GNN, step-by-step. This includes overall architecture of a GNN & critical aspect of message passing. Then go on to add different features & extensions to these basic aspects, e.g. introducing convolution & sampling, attention mechanisms, a generative model, & operating on dynamic graphs. When building our GNNs, work with Python & use some standard libraries. GNNs libraries are either standalone or use TensorFlow or PyTorch as a backend. In this text, focus will be on PyTorch Geometric (PyG). Other popular libraries include Deep Graph Library (DGL, a standalone library) & Spektral (which uses Keras & TensorFlow as a backend). There is also Jraph for JAX users.

– GNNs in Action hướng đến những người thực hành muốn bắt đầu triển khai GNN để giải quyết các vấn đề thực tế. Họ có thể là 1 kỹ sư ML chưa quen thuộc với cấu trúc dữ liệu đồ thị, 1 nhà khoa học dữ liệu chưa từng thử GNN, hoặc thậm chí là 1 kỹ sư phần mềm chưa quen thuộc với cả hai. Xuyên suốt cuốn sách này, chúng tôi sẽ đề cập đến các chủ đề từ kiến thức cơ bản về đồ thị cho đến các mô hình GNN phức tạp hơn. Xây dựng kiến trúc của GNN, từng bước một. Điều này bao gồm kiến trúc tổng thể của GNN & khía cạnh quan trọng của việc truyền thông điệp. Sau đó, tiếp tục thêm các tính năng khác nhau & phần mở rộng cho các khía cạnh cơ bản này, ví dụ: giới thiệu tích chập & lấy mẫu, cơ chế chú ý, mô hình sinh,

& hoạt động trên đồ thị động. Khi xây dựng GNN, làm việc với Python & sử dụng 1 số thư viện chuẩn. Thư viện GNN có thể độc lập hoặc sử dụng TensorFlow hoặc PyTorch làm nền tảng. Trong văn bản này, trọng tâm sẽ là PyTorch Geometric (PyG). Các thư viện phổ biến khác bao gồm Thư viện Deep Graph (DGL, 1 thư viện độc lập) & SPektral (sử dụng Keras & TensorFlow làm nền tảng). Ngoài ra còn có Jraph dành cho người dùng JAX.

Our aim throughout this book is to enable you to:

1. access suitability of a GNN solution for your problem.
2. understand when traditional neural networks won't perform as well as a GNN for graph structured data & when GNNs may not be the best tool for tabular data.
3. design & implement a GNN architecture to solve problems specific to you.
4. make clear limitations of GNNs.

This book is weighted toward implementation using programming. Also devote some time on essential theory & concepts, so that techniques covered can be sufficiently understood. These are covered in an “Under Hood” sect at end of most chaps to separate technical reasons from actual implementation. There are many different models & packages that build on key concepts introduced in this book. So, this book should not be seen as a comprehensive review of all GNNs methods & models, which could run to several thousands of pages, but rather starting point for curious & eager-to-learn practitioner.

– Mục tiêu của chúng tôi trong suốt cuốn sách này là giúp bạn:

1. đánh giá tính phù hợp của giải pháp GNN cho vấn đề của bạn.
2. hiểu khi nào mạng nơ-ron truyền thống không hoạt động tốt bằng GNN đối với dữ liệu có cấu trúc đồ thị & khi nào GNN có thể không phải là công cụ tốt nhất cho dữ liệu dạng bảng.
3. thiết kế & triển khai kiến trúc GNN để giải quyết các vấn đề cụ thể của bạn.
4. làm rõ những hạn chế của GNN.

Cuốn sách này thiên về việc triển khai bằng lập trình. Đồng thời dành thời gian cho các lý thuyết & khái niệm thiết yếu, để các kỹ thuật được đề cập có thể được hiểu đầy đủ. Những điều này được trình bày trong phần “Under Hood” ở cuối hầu hết các chương để phân biệt lý do kỹ thuật với việc triển khai thực tế. Có rất nhiều mô hình & gói khác nhau được xây dựng dựa trên các khái niệm chính được giới thiệu trong cuốn sách này. Vì vậy, cuốn sách này không nên được coi là 1 bài tổng quan toàn diện về tất cả các phương pháp & mô hình GNN, có thể dài tới hàng nghìn trang, mà nên là điểm khởi đầu cho những người thực hành & ham học hỏi.

Book is divided into 3 parts. Part 1 covers basics of GNNs, especially ways in which they differ from other neural networks, e.g. *message passing & embeddings*, which have specific meaning for GNNs. Part 2, heart of book, goes over models themselves, where we cover a handful of key model types. Then, in part 3, go into more detail with some of harder models & concepts, including how to scale graphs & deal with temporal data.

– Sách được chia thành 3 phần. Phần 1 trình bày những kiến thức cơ bản về GNN, đặc biệt là những điểm khác biệt giữa chúng với các mạng nơ-ron khác, ví dụ như truyền thông điệp & nhúng, vốn có ý nghĩa riêng đối với GNN. Phần 2, trọng tâm của sách, sẽ đề cập đến bản thân các mô hình, trong đó chúng ta sẽ tìm hiểu 1 số loại mô hình chính. Sau đó, trong phần 3, chúng ta sẽ đi sâu hơn vào 1 số mô hình & khái niệm khó hơn, bao gồm cách chia tỷ lệ đồ thị & xử lý dữ liệu thời gian.

GNNs in Action is designed for people to jump quickly into this new field & start building applications. Aim for this book: reduce friction of implementing new technologies by filling in gaps & answering key development questions whose answers may not be easy to find or may not be covered elsewhere at all. Each method is introduced through an example application so you can understand how GNNs are applied in practice.

– GNNs in Action được thiết kế để mọi người nhanh chóng tiếp cận lĩnh vực mới này & bắt đầu xây dựng ứng dụng. Mục tiêu của cuốn sách này: giảm thiểu sự cản trở khi triển khai các công nghệ mới bằng cách lấp đầy những khoảng trống & trả lời những câu hỏi phát triển quan trọng mà câu trả lời có thể không dễ tìm hoặc chưa được đề cập ở bất kỳ nơi nào khác. Mỗi phương pháp được giới thiệu thông qua 1 ứng dụng ví dụ để bạn có thể hiểu cách GNN được áp dụng trong thực tế.

* 1.1.1. **Catching up on graph fundamentals.** Do need to understand basics of graphs before you can understand GNNs. Goal for this book is to teach GNNs to DL practitioners & builders for traditional neural networks who may not know much about graphs. At same time, also recognize: readers of this book may vary enormously in their knowledge of graphs. How to address these differences & make sure everyone has what they need to make the most of this book? In this chap, provide an introduction to fundamental graph concepts that are most essential to understanding GNNs.

– Bạn cần nắm vững những kiến thức cơ bản về đồ thị trước khi có thể hiểu về GNN. Mục tiêu của cuốn sách này là hướng dẫn GNN cho những người thực hành DL & những người xây dựng mạng nơ-ron truyền thống, những người có thể chưa biết nhiều về đồ thị. Đồng thời, cũng cần lưu ý: kiến thức về đồ thị của độc giả có thể rất khác nhau. Làm thế nào để giải quyết những khác biệt này & đảm bảo mọi người đều có những kiến thức cần thiết để tận dụng tối đa cuốn sách này? Trong chương này, chúng tôi sẽ giới thiệu các khái niệm cơ bản về đồ thị, những khái niệm thiết yếu nhất để hiểu về GNN.

After refresher on key concepts in graphs & graph learning, look into some case studies in several fields where GNNs are being successfully applied. Then, break down those specific cases to see what makes a good case for using a GNN, as well as how to know if you have a GNN problem on your hands. At end of chap, introduce mechanics of GNNs, barebone skeleton that the rest of book will add to.

– Sau khi ôn lại các khái niệm chính về đồ thị & học đồ thị, xem xét 1 số nghiên cứu điển hình trong 1 số lĩnh vực mà GNN đang được ứng dụng thành công. Sau đó, phân tích các trường hợp cụ thể đó để xem đâu là lý do tốt để sử dụng

GNN, cũng như cách nhận biết liệu bạn có đang gặp vấn đề về GNN hay không. Cuối chương, giới thiệu cơ chế hoạt động của GNN, bộ khung xương cốt mà phần còn lại của cuốn sách sẽ bổ sung.

- 1.2. Graph-based learning. This section defines graphs, graph-based learning, & some fundamentals of GNNs, including basic structure of a graph & a taxonomy of different types of graphs. Then, review graph-based learning, putting GNNs in context with other learning methods. Finally, explain value of graphs, ending with an example of data derived from Titanic dataset.
 - Học tập dựa trên đồ thị. Phần này định nghĩa đồ thị, học tập dựa trên đồ thị, & 1 số kiến thức cơ bản về mạng nơ-ron nhân tạo (GNN), bao gồm cấu trúc cơ bản của đồ thị & phân loại các loại đồ thị khác nhau. Sau đó, xem xét lại học tập dựa trên đồ thị, đặt GNN vào bối cảnh của các phương pháp học tập khác. Cuối cùng, giải thích giá trị của đồ thị, kết thúc bằng 1 ví dụ về dữ liệu được lấy từ tập dữ liệu Titanic.

* 1.2.1. What are graphs? Graphs are data structures with elements, expressed as *nodes or vertices*, & relationships between elements, expressed as *edges or links*. All nodes in graph will have additional *feature data*. This is node-specific data, relating to things e.g. names or ages of individuals in a social network. Links are key to power of relational data, as they allow us to learn more about system, give new tools for analyzing data, & predict new properties from it. This is in contrast to tabular data e.g. a database table, dataframe, or spreadsheet, where data is fixed in rows & columns.

– Đồ thị là cấu trúc dữ liệu với các phần tử, được biểu diễn dưới dạng *nút hoặc đỉnh*, & mối quan hệ giữa các phần tử, được biểu diễn dưới dạng *cạnh hoặc liên kết*. Tất cả các nút trong đồ thị sẽ có thêm *dữ liệu đặc trưng*. Đây là dữ liệu cụ thể của từng nút, liên quan đến các thông tin như tên hoặc tuổi của các cá nhân trong mạng xã hội. Liên kết là chìa khóa cho sức mạnh của dữ liệu quan hệ, vì chúng cho phép chúng ta tìm hiểu thêm về hệ thống, cung cấp các công cụ mới để phân tích dữ liệu & dự đoán các thuộc tính mới từ dữ liệu đó. Điều này trái ngược với dữ liệu dạng bảng, ví dụ: bảng cơ sở dữ liệu, khung dữ liệu hoặc bảng tính, trong đó dữ liệu được cố định theo hàng & cột.

To describe & learn from edges between nodes, we need a way to write them down. This can be explicitly, quickly, can see describing things in this way becomes unwieldy & might be repeating redundant information. Luckily, there are many mathematical formalisms for describing relations in graphs. 1 of most common: describe *adjacency matrix*. Notice: adjacency matrix is symmetric across diagonal & all values are 1s or 0s. Adjacency matrix of a graph is an important concept that makes it easy to observe all connections of a graph in a single table. Here assumed: there is no directionability in our graphs, i.e., if 0 is connected to 1, then 1 is also connected to 0. This is known as an *undirected graph*. Undirected graphs can be easily inferred from an adjacency matrix because, in this case, matrix is symmetric across diagonal, upper-right triangle is reflected onto bottom-left.

– Để mô tả & học từ các cạnh giữa các nút, chúng ta cần 1 cách để viết chúng ra. Điều này có thể rõ ràng, nhanh chóng, có thể thấy việc mô tả mọi thứ theo cách này trở nên cồng kềnh & có thể lặp lại thông tin dư thừa. May mắn thay, có nhiều công thức toán học để mô tả các mối quan hệ trong đồ thị. 1 trong những công thức phổ biến nhất: mô tả *adjacency matrix*. Lưu ý: ma trận kề là đối xứng qua đường chéo & tất cả các giá trị là 1 hoặc 0. Ma trận kề của đồ thị là 1 khái niệm quan trọng giúp dễ dàng quan sát tất cả các kết nối của đồ thị trong 1 bảng duy nhất. Ở đây giả sử: không có tính định hướng trong đồ thị của chúng ta, tức là nếu 0 được kết nối với 1, thì 1 cũng được kết nối với 0. Đây được gọi là *undirected graph*. Đồ thị vô hướng có thể dễ dàng suy ra từ ma trận kề vì, trong trường hợp này, ma trận đối xứng qua đường chéo, tam giác trên cùng bên phải được phản chiếu xuống dưới cùng bên trái.

Also assume: all relations between nodes are identical. If we wanted relation of nodes B-E to mean more than relation of nodes B-A, then we could increase weight of this edge. This translates to increasing value in adjacency matrix, making entry for B-A edge equal to 10 instead of 1, e.g.

– Cũng giả sử: tất cả các mối quan hệ giữa các nút đều giống hệt nhau. Nếu chúng ta muốn mối quan hệ giữa các nút B-E có ý nghĩa hơn mối quan hệ giữa các nút B-A, thì chúng ta có thể tăng trọng số của cạnh này. Điều này tương đương với việc tăng giá trị trong ma trận kề, ví dụ, nhập giá trị cho cạnh B-A bằng 10 thay vì 1.

Graphs where all relations are of equal importance are known as *unweighted graphs* & can also be easily observed from adjacency matrix because all graph entries are either 1s or 0s. Graphs where edges have multiple values are known as *weighted*.

– Đồ thị mà tất cả các mối quan hệ đều có tầm quan trọng như nhau được gọi là *unweighted graphs* & cũng có thể dễ dàng quan sát được từ ma trận kề vì tất cả các mục đồ thị đều là 1 hoặc 0. Đồ thị mà các cạnh có nhiều giá trị được gọi là *weighted*.

If any of nodes in graph do not have an edge that connects to itself, then nodes will also have 0s at their own value in adjacency matrix (0s along diagonal), i.e., a graph does not have self-loops. A *self-loop* occurs when a node has an edge that connects to that same node. To add a self-loop, we just make value for that node nonzero at its position in diagonal.

– Nếu bất kỳ nút nào trong đồ thị không có cạnh nối với chính nó, thì các nút cũng sẽ có giá trị 0 tại chính nó trong ma trận kề (các giá trị 0 dọc theo đường chéo), tức là đồ thị không có vòng lặp tự thân. 1 vòng lặp tự thân xảy ra khi 1 nút có cạnh nối với chính nút đó. Để thêm 1 vòng lặp tự thân, chúng ta chỉ cần gán giá trị khác không cho nút đó tại vị trí của nó trên đường chéo.

In practice, an adjacency matrix is only 1 of many ways to describe relations in a graph. Others include adjacency lists, edge lists, or an incidence matrix. Understanding these types of data structures well is vital to graph-based learning. If you are unfamiliar with these terms, or need a refresher, recommend looking through appendix A, which has additional details & explanations.

– Trên thực tế, ma trận kề chỉ là 1 trong nhiều cách để mô tả các mối quan hệ trong đồ thị. Các cách khác bao gồm danh sách kề, danh sách cạnh, hoặc ma trận liên quan. Việc hiểu rõ các loại cấu trúc dữ liệu này rất quan trọng đối với việc học tập dựa trên đồ thị. Nếu bạn chưa quen với các thuật ngữ này hoặc cần ôn tập lại, xem Phụ lục A, trong đó có thêm chi tiết & giải thích.

* 1.2.2. Different types of graphs. Understanding many different types of graphs can help us work out what methods to use to analyze & transform graph, & what ML methods to apply. In following, give a very quick overview of some of most common properties for graphs to have.

– Hiểu biết về nhiều loại đồ thị khác nhau có thể giúp chúng ta tìm ra phương pháp nào cần sử dụng để phân tích & biến đổi đồ thị, & áp dụng phương pháp ML nào. Sau đây, chúng tôi sẽ giới thiệu sơ lược về 1 số thuộc tính phổ biến nhất của đồ thị.

• **Homogeneous & Heterogeneous Graphs.** Most basic graphs are *homogeneous graphs*, which are made up of 1 type of node & 1 type of edge. Consider a homogeneous graph that describes a recruitment network. In this type of graph, nodes would represent job candidates, & edges would represent relationships between candidates.

– Đồ thị Đồng nhất & Đồ thị Không Đồng nhất. Hầu hết các đồ thị cơ bản là *đồ thị đồng nhất*, được tạo thành từ 1 loại nút & 1 loại cạnh. Xem xét 1 đồ thị đồng nhất mô tả 1 mạng lưới tuyển dụng. Trong loại đồ thị này, các nút sẽ đại diện cho các ứng viên xin việc, & cạnh sẽ đại diện cho mối quan hệ giữa các ứng viên.

If want to expand power of our graph to describe our recruitment network, could give it more types of nodes & edges, making it a *heterogeneous graphs*. With this expansion, some nodes may be candidates & others may be companies. Edges could now consist of relationships between candidates & current or past employment of job candidates at companies. See Fig. 1.2: A homogeneous graph & a heterogeneous graph. Here, shade of a node or edge represents its type or class. For homogeneous graph, all nodes are of same type, & all edges are of same type. For heterogeneous graph, nodes & edges have multiple types for a comparison of a homogeneous graph (all nodes or edges have same shade) with a heterogeneous graph (nodes & edges have a variety of shades).

– Nếu muốn mở rộng sức mạnh của đồ thị để mô tả mạng lưới tuyển dụng, có thể cung cấp cho nó nhiều loại nút & cạnh hơn, biến nó thành *đồ thị không đồng nhất*. Với sự mở rộng này, 1 số nút có thể là ứng viên & những nút khác có thể là công ty. Các cạnh bây giờ có thể bao gồm các mối quan hệ giữa ứng viên & việc làm hiện tại hoặc trước đây của ứng viên tại các công ty. Xem Hình 1.2: Đồ thị đồng nhất & đồ thị không đồng nhất. Ở đây, sắc thái của 1 nút hoặc cạnh biểu thị loại hoặc lớp của nó. Đối với đồ thị đồng nhất, tất cả các nút đều cùng loại, & tất cả các cạnh đều cùng loại. Đối với đồ thị không đồng nhất, các nút & cạnh có nhiều loại để so sánh đồ thị đồng nhất (tất cả các nút hoặc cạnh có cùng sắc thái) với đồ thị không đồng nhất (các nút & cạnh có nhiều sắc thái khác nhau).

• **Bipartite graphs.** Similar to heterogeneous graphs, *bipartite graphs* also can be separated or partitioned into different subsets. However, bipartite graphs (Fig. 1.3: A bipartite graph. There are 2 types of nodes (2 shades of circles). In a bipartite graph, nodes cannot be connected to nodes of same type. This is also an example of a heterogeneous graph.) have a very specific network structure s.t. nodes in each subset connect to nodes outside of their subset & not inside. Later, discuss recommendation system & Pinterest graph. This graph is bipartite because 1 set of nodes (pins) connects another set of nodes (boards) but not to nodes within their set (pins).

– Đồ thị hai phần. Tương tự như đồ thị không đồng nhất, *Đồ thị 2 phần* cũng có thể được tách hoặc phân vùng thành các tập con khác nhau. Tuy nhiên, đồ thị hai phần (Hình 1.3: Đồ thị hai phần. Có 2 loại nút (2 sắc thái của hình tròn). Trong đồ thị hai phần, các nút không thể được kết nối với các nút cùng loại. Đây cũng là 1 ví dụ về đồ thị không đồng nhất.) có cấu trúc mạng rất cụ thể, ví dụ: các nút trong mỗi tập con kết nối với các nút bên ngoài tập con của chúng & không kết nối với các nút bên trong. Sau đó, thảo luận về hệ thống đề xuất & đồ thị Pinterest. Đồ thị này là hai phần vì 1 tập hợp các nút (ghim) kết nối với 1 tập hợp các nút khác (bảng) nhưng không kết nối với các nút trong tập hợp của chúng (ghim).

• **Cyclic graphs, acyclic graphs, & directed acyclic graphs.** A graph is *cyclic* if it allows you to start at a node, travel along its edge, & return to starting node without retracing any steps, creating a circular path within graph. In contrast, in an *acyclic* graph, no matter which path you take from any starting node, you cannot return to starting point without backtracking. These graphs, as shown in Fig. 1.4: A cyclic graph, an acyclic graph, & a DAG. In cyclic graph, cycle is shown by arrows (directed edges) connecting nodes A-E-D-C-B-A. Note 2 nodes, F & G are part of graph, but not part of its defining cycle. Acyclic graph is composed of undirected edges, & no cycle is possible. In DAG, all directed edges flow in 1 direction, from A to F., often resemble tree-like structures or paths that do not loop back on themselves.

– Đồ thị tuần hoàn, đồ thị phi chu trình, & đồ thị phi chu trình có hướng. 1 đồ thị là *tuần hoàn* nếu nó cho phép bạn bắt đầu tại 1 nút, di chuyển dọc theo cạnh của nó, & quay lại nút bắt đầu mà không cần quay lại bất kỳ bước nào, tạo ra 1 đường tròn trong đồ thị. Ngược lại, trong đồ thị *phi chu trình*, bất kể bạn đi theo đường nào từ bất kỳ nút bắt đầu nào, bạn không thể quay lại điểm bắt đầu mà không quay lại. Các đồ thị này, như được thể hiện trong Hình 1.4: Đồ thị tuần hoàn, đồ thị phi chu trình, & DAG. Trong đồ thị tuần hoàn, chu trình được biểu diễn bằng các mũi tên (cạnh có hướng) nối các nút A-E-D-C-B-A. Lưu ý 2 nút, F & G là 1 phần của đồ thị, nhưng không phải là 1 phần của chu trình xác định của nó. Đồ thị phi chu trình bao gồm các cạnh không có hướng, & không thể có chu trình. Trong DAG, tất cả các cạnh có hướng đều chảy theo 1 hướng, từ A đến F., thường giống các cấu trúc giống cây hoặc các đường dẫn không vòng lại với chính chúng.

While both cyclic & acyclic graphs can be either undirected or directed, a *directed acyclic graph (DAG)* is a specific type of acyclic graph that is exclusively directed. In a DAG, all edges have a direction, & no cycles are allowed. DAGs represent 1-way relationships where we can't follow arrows & end up back at starting point. This characteristic makes DAGs essential in causal analysis, as they reflect causal structures where causality is assumed to be unidirectional. E.g., A can cause B, but B can't simultaneously cause A. This unidirectional nature aligns perfectly with structure of DAGs, making them ideal for modeling workflow processes, dependency chains, & causal relationships in various fields.

– Trong khi cả đồ thị tuần hoàn & đồ thị phi tuần hoàn đều có thể vô hướng hoặc có hướng, *đồ thị phi tuần hoàn có hướng (DAG)* là 1 loại đồ thị phi tuần hoàn cụ thể chỉ có hướng. Trong DAG, tất cả các cạnh đều có hướng, & không

cho phép chu trình. DAG biểu diễn các mối quan hệ 1 chiều, trong đó chúng ta không thể theo mũi tên & kết thúc ở điểm xuất phát. Đặc điểm này làm cho DAG trở nên thiết yếu trong phân tích nhân quả, vì chúng phản ánh các cấu trúc nhân quả trong đó quan hệ nhân quả được coi là vô hướng. Ví dụ: A có thể gây ra B, nhưng B không thể đồng thời gây ra A. Bản chất đơn hướng này hoàn toàn phù hợp với cấu trúc của DAG, khiến chúng trở nên lý tưởng để mô hình hóa các quy trình công việc, chuỗi phụ thuộc, & các mối quan hệ nhân quả trong nhiều lĩnh vực khác nhau.

• **Knowledge graphs.** A *knowledge graph* is a specialized type of heterogeneous graph that represents data with enriched semantic meaning, capturing not only relationships between different entities but also context & nature of these relationships. Unlike conventional graphs, which primarily emphasize structure & connectivity, a knowledge graph incorporates metadata & follow specific schemas to provide deeper contextual information. This allows for advanced reasoning & querying capabilities, e.g. identifying patterns, uncovering specific types of connections, or inferring new relationships.

– **Đồ thị tri thức.** Đồ thị tri thức là 1 loại đồ thị không đồng nhất chuyên biệt, biểu diễn dữ liệu với ý nghĩa ngữ nghĩa phong phú, không chỉ nắm bắt mối quan hệ giữa các thực thể khác nhau mà còn cả bối cảnh & bản chất của các mối quan hệ này. Không giống như đồ thị thông thường, chủ yếu nhấn mạnh vào cấu trúc & kết nối, đồ thị tri thức kết hợp siêu dữ liệu & tuân theo các lược đồ cụ thể để cung cấp thông tin ngữ cảnh sâu hơn. Điều này cho phép khả năng suy luận & truy vấn nâng cao, ví dụ: xác định các mẫu, khám phá các loại kết nối cụ thể hoặc suy ra các mối quan hệ mới. In example of an academic research network at a university, a knowledge graph might represent various entities e.g. Professors, Students, Papers, & Research Topics, & explicitly define relationships between them. E.g., Professors & Students could be associated with Papers through an Authorship relationship, while Professors might also Supervise Students. Furthermore, graph would reflect hierarchical structures, e.g., Professors & Students being categorized under Departments. Can see this knowledge graph depicted in Fig. 1.5: A knowledge graph representing an academic research network within a university's physics department. Graph illustrates both hierarchical relationships, e.g., professors & students as members of department, & behavioral relationships, e.g., professors supervising students & authoring papers. Entities e.g. Professors, Students, Papers, & Topics are connected through semantically meaningful relationships (Supervises, Wrote, Inspires). Entities also have detailed features (Name, Department, Type) providing further context. Semantic connections & features enable advanced querying & analysis of complex academic interactions.

– Trong ví dụ về mạng lưới nghiên cứu học thuật tại 1 trường đại học, biểu đồ kiến thức có thể biểu diễn nhiều thực thể khác nhau, ví dụ: Giáo sư, Sinh viên, Bài báo, & Chủ đề nghiên cứu, & định nghĩa rõ ràng mối quan hệ giữa chúng. Ví dụ: Giáo sư & Sinh viên có thể được liên kết với Bài báo thông qua mối quan hệ Tác giả, trong khi Giáo sư cũng có thể Giám sát Sinh viên. Hơn nữa, biểu đồ sẽ phản ánh các cấu trúc phân cấp, ví dụ: Giáo sư & Sinh viên được phân loại theo Khoa. Có thể xem biểu đồ kiến thức này được mô tả trong Hình 1.5: Biểu đồ kiến thức biểu diễn mạng lưới nghiên cứu học thuật trong khoa vật lý của 1 trường đại học. Biểu đồ minh họa cả các mối quan hệ phân cấp, ví dụ: giáo sư & sinh viên là thành viên của khoa, & các mối quan hệ hành vi, ví dụ: giáo sư hướng dẫn sinh viên & biên soạn bài báo. Các thực thể, ví dụ: Giáo sư, Sinh viên, Bài báo, & Chủ đề được kết nối thông qua các mối quan hệ có ý nghĩa ngữ nghĩa (Giám sát, Viết, Truyền cảm hứng). Các thực thể cũng có các tính năng chi tiết (Tên, Khoa, Loại) cung cấp thêm ngữ cảnh. Kết nối ngữ nghĩa & các tính năng cho phép truy vấn nâng cao & phân tích các tương tác học thuật phức tạp.

Note 1. Should use multigraphs to further represent knowledge graphs.

A key feature of knowledge graphs is their ability to provide explicit context. Unlike conventional heterogeneous graphs, which display different types of entities & their basic connections without detailed semantic meaning, knowledge graphs go further by defining specific types & meanings of relationships. E.g., while a traditional graph might show that Professors are connected to Departments or that Students are linked to Papers, a knowledge graph would specify that Professors supervise Students or that Students & Professors Wrote Papers. This added layer of meaning enables more powerful querying & analysis, making knowledge graphs particularly valuable in fields e.g. NLP, recommendation systems, & academic research analysis.

– 1 đặc điểm quan trọng của đồ thị tri thức là khả năng cung cấp ngữ cảnh rõ ràng. Không giống như các đồ thị không đồng nhất thông thường, vốn hiển thị các loại thực thể khác nhau & các kết nối cơ bản của chúng mà không có ý nghĩa ngữ nghĩa chi tiết, đồ thị tri thức tiến xa hơn bằng cách xác định các loại & ý nghĩa cụ thể của các mối quan hệ. Ví dụ: trong khi đồ thị truyền thống có thể hiển thị Giáo sư được kết nối với Khoa hoặc Sinh viên được liên kết với Bài báo, đồ thị tri thức sẽ chỉ rõ Giáo sư hướng dẫn Sinh viên hoặc Sinh viên & Giáo sư viết Bài báo. Lớp ý nghĩa bổ sung này cho phép truy vấn & phân tích mạnh mẽ hơn, khiến đồ thị tri thức đặc biệt có giá trị trong các lĩnh vực như NLP, hệ thống đề xuất, & phân tích nghiên cứu học thuật.

• **Hypergraphs.** 1 of more complex & difficult graphs to work with is hypergraph. *Hypergraphs* are those where a single edge can be connected to multiple different nodes. For graphs that are not hypergraphs, edges are used to connected exactly 2 nodes (or a node to itself for self-loops). As shown in Fig. 1.6: 1 undirected hypergraph, illustrated in 2 ways. On left, have a graph whose edges are represented by shaded areas, marked by letters, & whose vertices are dots, marked by numbers. On right, have a graph whose edge lines (marked by letters) connect up to 3 nodes (circles marked by numbers)., edges in a hypergraph can connect between any number of nodes. Complexity of a hypergraph is reflected in its adjacency data. For typical graphs, network connectivity is represented by a 2D adjacency matrix. For hypergraphs, adjacency matrix extends to a higher dimensional tensor, referred to as an *incidence tensor*. This tensor is N -dimensional, where N is maximum number of nodes connected by a single edge. An example of a hypergraph might be a communication platform that allows for group chats as well as single person conversations. In an ordinary graph, edges would only connect 2 people. In a hypergraph, 1 hyperedge could connect multiple people, representing a group chat.

– **Siêu đồ thị.** 1 trong những đồ thị phức tạp hơn & khó làm việc hơn là siêu đồ thị. Siêu đồ thị là những đồ thị mà 1 cạnh đơn có thể được kết nối với nhiều nút khác nhau. Đối với những đồ thị không phải là siêu đồ thị, các cạnh được

sử dụng để kết nối chính xác 2 nút (hoặc 1 nút với chính nó đối với các vòng lặp tự thân). Như thể hiện trong Hình 1.6: 1 siêu đồ thị vô hướng, được minh họa theo 2 cách. Bên trái, có 1 đồ thị mà các cạnh được biểu diễn bằng các vùng tô bóng, được đánh dấu bằng các chữ cái, & có các đỉnh là các dấu chấm, được đánh dấu bằng các số. Bên phải, có 1 đồ thị mà các đường cạnh (được đánh dấu bằng các chữ cái) kết nối tối đa 3 nút (các vòng tròn được đánh dấu bằng các số)., các cạnh trong siêu đồ thị có thể kết nối giữa bất kỳ số lượng nút nào. Độ phức tạp của siêu đồ thị được phản ánh trong dữ liệu kề của nó. Đối với các đồ thị thông thường, kết nối mạng được biểu diễn bằng ma trận kề 2D. Đối với siêu đồ thị, ma trận kề mở rộng đến 1 tenxơ có chiều cao hơn, được gọi là tenxơ *incidence tenxơ*. Tenxơ này là tenxơ N chiều, trong đó N là số lượng nút tối đa được kết nối bởi 1 cạnh duy nhất. 1 ví dụ về siêu đồ thị có thể là 1 nền tảng giao tiếp cho phép trò chuyện nhóm cũng như trò chuyện cá nhân. Trong 1 đồ thị thông thường, các cạnh chỉ kết nối 2 người. Trong 1 siêu đồ thị, 1 siêu cạnh có thể kết nối nhiều người, đại diện cho 1 cuộc trò chuyện nhóm.

- * 1.2.3. Graph-based learning. Graphs are ubiquitous in our everyday life. *Graph-based learning* takes graphs as input data to build models that give insight into questions about this data. Later in this chap, look at different examples of graph data as well as at sort of questions & tasks we can use graph-based learning to answer.

- Học tập dựa trên đồ thị. Đồ thị hiện diện khắp nơi trong cuộc sống hàng ngày của chúng ta. *Học tập dựa trên đồ thị* sử dụng đồ thị làm dữ liệu đầu vào để xây dựng các mô hình cung cấp thông tin chi tiết về các câu hỏi liên quan đến dữ liệu này. Ở phần sau của chương này, xem xét các ví dụ khác nhau về dữ liệu đồ thị cũng như các loại câu hỏi & bài tập mà chúng ta có thể sử dụng học tập dựa trên đồ thị để trả lời.

Graph-based learning uses a variety of ML methods to build *representations* of graphs. These representations are then used for downstream tasks e.g. node or link prediction or graph classification. In Chap. 2, learn about 1 of essential tools in graph-based learning, building embeddings. Briefly, embeddings are *low-dimensional* vector representations. Can build an embedding of different nodes, edges, or entire graphs, & there are a number of different ways to do this e.g. Node2Vec (N2V) or DeepWalk algorithms.

- Học tập dựa trên đồ thị sử dụng nhiều phương pháp học máy khác nhau để xây dựng *biểu diễn* của đồ thị. Các biểu diễn này sau đó được sử dụng cho các tác vụ hạ nguồn, ví dụ như dự đoán nút hoặc liên kết hoặc phân loại đồ thị. Trong Chương 2, tìm hiểu về 1 trong những công cụ thiết yếu trong học tập dựa trên đồ thị, đó là xây dựng nhúng. Nói 1 cách ngắn gọn, nhúng là biểu diễn vectơ *chiều thấp*. Có thể xây dựng nhúng của các nút, cạnh hoặc toàn bộ đồ thị khác nhau, & có 1 số cách khác nhau để thực hiện việc này, ví dụ như thuật toán Node2Vec (N2V) hoặc DeepWalk.

Methods for analysis on graph data have been around for a long time, at least as early as 1950s when *clique methods* used certain features of a graph to identify subsets or communities in graph data [4].

- Các phương pháp phân tích dữ liệu đồ thị đã có từ rất lâu, ít nhất là từ những năm 1950 khi *clique methods* sử dụng 1 số tính năng nhất định của đồ thị để xác định các tập hợp con hoặc cộng đồng trong dữ liệu đồ thị.

1 of most famous graph-based algorithms is PageRank, which was developed by LARRY PAGE & SERGEY BRIN in 1996 & formed basis for Google's search algorithms. Some believe: this algorithm was a key element in company's meteoric rise in following years. This highlights that a successful graph-based learning algorithm can have a huge effect.

- 1 trong những thuật toán dựa trên đồ thị nổi tiếng nhất là PageRank, được phát triển bởi Larry Page & Sergey Brin vào năm 1996 & tạo nền tảng cho các thuật toán tìm kiếm của Google. 1 số người tin rằng: thuật toán này là yếu tố then chốt cho sự phát triển vượt bậc của công ty trong những năm tiếp theo. Điều này nhấn mạnh rằng 1 thuật toán học dựa trên đồ thị thành công có thể mang lại hiệu quả to lớn.

These methods are only a small subset of graph-based learning & analysis techniques. Others include belief propagation [5], graph kernel methods [6], label propagation [7], & isomaps [8]. However, in this book, focus on 1 of newest & most exciting additions to family of graph-based learning techniques: GNNs.

- Những phương pháp này chỉ là 1 tập hợp con nhỏ của các kỹ thuật học tập dựa trên đồ thị & phân tích. Các phương pháp khác bao gồm truyền bá niềm tin [5], phương pháp hạt nhân đồ thị [6], truyền bá nhân [7], & isomaps [8]. Tuy nhiên, trong cuốn sách này, chúng tôi tập trung vào 1 trong những bổ sung mới nhất & thú vị nhất cho nhóm kỹ thuật học tập dựa trên đồ thị: GNN.

- * 1.2.4. What is a GNN? GNNs combine graph-based learning with DL, i.e., neural networks are used to build embeddings & process relational data. An overview of inner workings of a GNN is shown in Fig. 1.7: An overview of how GNNs work. An input graph is passed to a GNN. GNN then uses neural networks to transform graph features e.g. nodes or edges into nonlinear embeddings through a process known as message passing. These embeddings are then tuned to specific unknown properties using training data. After GNN is trained, it can predict unknown features of a graph.

- GNN kết hợp học tập dựa trên đồ thị với DL, tức là mạng nơ-ron được sử dụng để xây dựng các nhúng & xử lý dữ liệu quan hệ. Tổng quan về hoạt động bên trong của GNN được thể hiện trong Hình 1.7: Tổng quan về cách thức hoạt động của GNN. 1 đồ thị đầu vào được truyền đến GNN. Sau đó, GNN sử dụng mạng nơ-ron để chuyển đổi các đặc trưng đồ thị, ví dụ như các nút hoặc cạnh, thành các nhúng phi tuyến tính thông qua 1 quá trình được gọi là truyền thông điệp. Các nhúng này sau đó được điều chỉnh theo các thuộc tính cụ thể chưa biết bằng cách sử dụng dữ liệu huấn luyện. Sau khi GNN được huấn luyện, nó có thể dự đoán các đặc trưng chưa biết của đồ thị.

GNNs allows you to represent & learn from graphs, including their constituent nodes, edges, & features. In particular, many methods of GNNs are built specifically to scale effectively with size & complexity of a graph, i.e., GNNs can operate on huge graphs. In this sense, GNNs provide analogous advantages to relational data as convolutional neural networks have given for image-based data & computer vision.

- GNN cho phép bạn biểu diễn & học từ các đồ thị, bao gồm các nút, cạnh, & đặc trưng cấu thành của chúng. Đặc biệt, nhiều phương pháp GNN được xây dựng chuyên biệt để mở rộng hiệu quả theo kích thước & độ phức tạp của đồ thị, tức là GNN có thể hoạt động trên các đồ thị rất lớn. Theo nghĩa này, GNN mang lại những lợi thế tương tự cho dữ liệu quan

hệ như mạng nơ-ron tích chập đã mang lại cho dữ liệu dựa trên hình ảnh & thị giác máy tính.

Historically, applying traditional ML methods to graph data structures has been challenging because graph data, when represented in grid-like formats & data structures, can lead to massive repetitions of data. To address this, graph-based learning focuses on approaches that are *permutation invariant*, i.e., ML method is uninfluenced by ordering of graph representation. In concrete terms, it means: we can shuffle rows & columns of adjacency matrix without affecting our algorithm's performance. Whenever we are working with data that contains relational data, i.e., has an adjacency matrix, then we want to use a ML method that is permutation invariant to make our method more general & efficient. Although GNNs can be applied to all graph data, GNNs are especially useful because they can deal with huge graph datasets & typically perform better than other ML methods.

– Theo truyền thống, việc áp dụng các phương pháp ML truyền thống vào cấu trúc dữ liệu đồ thị là 1 thách thức vì dữ liệu đồ thị, khi được biểu diễn ở định dạng dạng lưới & cấu trúc dữ liệu, có thể dẫn đến sự lặp lại dữ liệu rất lớn. Để giải quyết vấn đề này, học dựa trên đồ thị tập trung vào các phương pháp *permutation invariant*, tức là phương pháp ML không bị ảnh hưởng bởi thứ tự biểu diễn đồ thị. Nói 1 cách cụ thể, điều này có nghĩa là: chúng ta có thể xáo trộn các hàng & cột của ma trận kề mà không ảnh hưởng đến hiệu suất của thuật toán. Bất cứ khi nào chúng ta làm việc với dữ liệu có chứa dữ liệu quan hệ, tức là có ma trận kề, thì chúng ta muốn sử dụng phương pháp ML không thay đổi hoán vị để làm cho phương pháp của chúng ta tổng quát hơn & hiệu quả hơn. Mặc dù GNN có thể được áp dụng cho tất cả dữ liệu đồ thị, nhưng GNN đặc biệt hữu ích vì chúng có thể xử lý các tập dữ liệu đồ thị khổng lồ & thường hoạt động tốt hơn các phương pháp ML khác.

Permutation invariances are a type of *inductive bias*, or an algorithm's learning bias, & are powerful tools for designing ML algorithms [1]. Need for permutation-invariant approaches is 1 of central reasons that graph-based learning has increased in popularity in recent years.

– Bất biến hoán vị là 1 loại *thiên vị quy nạp*, hay thiên vị học tập của thuật toán, & là những công cụ mạnh mẽ để thiết kế các thuật toán ML [1]. Nhu cầu về các phương pháp bất biến hoán vị là 1 trong những lý do chính khiến học tập dựa trên đồ thị ngày càng phổ biến trong những năm gần đây.

Insights. Being designed for permutation-invariant data comes with some drawbacks along with its advantages. **GNNs are not as**

While this might seem obvious, images & tables are not permutation invariant & therefore not a good fit for GNNs. If we shuffle rows & columns of an image, then we scramble input. Instead, ML algorithms for images seek *translational invariance*, i.e., we can translate (shift) object in an image, & it won't affect performance of algorithm. Other neural networks, e.g. convolutional neural networks (CNNs) typically perform much better on images.

– Việc được thiết kế cho dữ liệu bất biến hoán vị đi kèm với 1 số nhược điểm bên cạnh những ưu điểm của nó.

GNN không phù hợp lắm với các dữ liệu khác, ví dụ như hình ảnh hoặc bảng. Mặc dù điều này có vẻ hiển nhiên, nhưng hình ảnh & bảng không bất biến hoán vị & do đó không phù hợp với GNN. Nếu chúng ta xáo trộn các hàng & cột của 1 hình ảnh, thì chúng ta sẽ xáo trộn dữ liệu đầu vào. Thay vào đó, các thuật toán ML cho hình ảnh tìm kiếm *translational invariance*, tức là chúng ta có thể dịch chuyển (shift) đối tượng trong 1 hình ảnh, & điều này sẽ không ảnh hưởng đến hiệu suất của thuật toán. Các mạng nơ-ron khác, ví dụ như mạng nơ-ron tích chập (CNN) thường hoạt động tốt hơn nhiều trên hình ảnh.

* 1.2.5. Differences between tabular & graph data. Graph data includes all data with some relational content, making it a powerful way to represent complex connections. While graph data might initially seem distinct from traditional tabular data, many datasets that are typically represented in tables can be recreated as graphs with some data engineering & imagination. Take a closer look at Titanic dataset, a classic example in ML, & explore how it can be transformed from a table format to a graph format.

– Sự khác biệt giữa dữ liệu dạng bảng & dữ liệu đồ thị. Dữ liệu đồ thị bao gồm tất cả dữ liệu có nội dung quan hệ, khiến nó trở thành 1 phương pháp mạnh mẽ để biểu diễn các kết nối phức tạp. Mặc dù ban đầu dữ liệu đồ thị có vẻ khác biệt so với dữ liệu dạng bảng truyền thống, nhưng nhiều tập dữ liệu thường được biểu diễn dưới dạng bảng có thể được tái tạo dưới dạng đồ thị với 1 chút kỹ thuật dữ liệu & trí tưởng tượng. Hãy xem xét kỹ hơn tập dữ liệu Titanic, 1 ví dụ kinh điển trong ML, & khám phá cách nó có thể được chuyển đổi từ định dạng bảng sang định dạng đồ thị.

Titanic dataset describes passengers on Titanic, a ship that famously met an untimely end when it collided with an iceberg. Historically, this datasets has been analyzed in tabular format, containing rows for each passenger with columns representing features e.g. age, gender, fare, class, & survival status. However, dataset also contains rich, unexplored relationships that are not immediately visible in a table format Fig. 1.8: Titanic Dataset is usually displayed & analyzed using a table format.

– Bộ dữ liệu Titanic mô tả hành khách trên Titanic, 1 con tàu nổi tiếng đã gặp phải cái chết bất ngờ khi va chạm với 1 tảng băng trôi. Trước đây, bộ dữ liệu này được phân tích theo định dạng bảng, bao gồm các hàng cho mỗi hành khách & các cột biểu diễn các đặc điểm như tuổi, giới tính, giá vé, hạng ghế, & tình trạng sống sót. Tuy nhiên, bộ dữ liệu cũng chứa các mối quan hệ phong phú, chưa được khám phá mà không thể hiển thị ngay lập tức ở định dạng bảng Hình 1.8: Bộ dữ liệu Titanic thường được hiển thị & phân tích bằng định dạng bảng.

• Recasting Titanic dataset as a graph. To transform Titanic dataset into a graph, need to consider how to represent underlying relationships between passengers as nodes & edges:

1. Nodes: In graph, each passenger can be represented as a node. Can also introduce nodes for other entities, e.g. cabins, families, or even groups e.g. “3rd-class passengers”.
2. Edges represent relationships or connections between these nodes, e.g.: Passengers who are family members (siblings, spouses, parents, or children) based on available data; Passengers who share a cabin or were traveling together; Social

or business relationships that might be inferred from shared ticket numbers, last names, or other identifying features.

– Tái cấu trúc tập dữ liệu Titanic dưới dạng đồ thị. Để chuyển đổi tập dữ liệu Titanic thành đồ thị, cần xem xét cách biểu diễn các mối quan hệ cơ bản giữa hành khách dưới dạng các nút & cạnh:

1. Nút: Trong đồ thị, mỗi hành khách có thể được biểu diễn dưới dạng 1 nút. Cũng có thể giới thiệu các nút cho các thực thể khác, ví dụ: cabin, gia đình hoặc thậm chí các nhóm, ví dụ: “hành khách hạng 3”.
2. Cạnh biểu diễn các mối quan hệ hoặc kết nối giữa các nút này, ví dụ: Hành khách là thành viên gia đình (anh chị em ruột, vợ/chồng, cha mẹ hoặc con cái) dựa trên dữ liệu có sẵn; Hành khách ở chung cabin hoặc đi cùng nhau; Các mối quan hệ xã hội hoặc kinh doanh có thể được suy ra từ số vé, họ hoặc các đặc điểm nhận dạng chung khác.

To construct this graph, need to use existing information in table & potentially enrich it with secondary data sources or assumptions (e.g., linking last names to create family groups). This process converts tabular data into a graph-based structure, shown in Fig. 1.9: Titanic dataset, showing family relationships of people on Titanic visualized as a graph. Here, can see that there was a rich social network as well as many passengers with unknown family ties., where each edge & node encapsulates meaningful relational data.

– Để xây dựng biểu đồ này, cần sử dụng thông tin hiện có trong bảng & có thể làm giàu nó bằng các nguồn dữ liệu thứ cấp hoặc giả định (ví dụ: liên kết họ để tạo nhóm gia đình). Quá trình này chuyển đổi dữ liệu dạng bảng thành cấu trúc dạng biểu đồ, được hiển thị trong Hình 1.9: Bộ dữ liệu Titanic, thể hiện mối quan hệ gia đình của những người trên Titanic được trực quan hóa dưới dạng biểu đồ. Ở đây, có thể thấy rằng có 1 mạng lưới xã hội phong phú cũng như nhiều hành khách có mối quan hệ gia đình chưa rõ ràng., trong đó mỗi cạnh & nút đóng gói dữ liệu quan hệ có ý nghĩa.

• How graph data adds depth & meaning. Once dataset is represented as a graph, it provides a much deeper view of social & familial connections between passengers, e.g.,:

1. *Family relationships*: Graph clearly shows how certain passengers were related (e.g., as parents, children, or siblings). This could help us understand survival patterns, as family members might have behaved differently in a crisis than individuals traveling alone.
2. *Social networks*: Beyond families, graph could reveal broader social networks (e.g., friendships or business connections), which could be important factors in analyzing behavior & outcomes.
3. *Community insights*: Graph structure also allows for community detection algorithms to identify clusters of related or connected passengers, which may reveal new insights into survival rates, rescue patterns, or other behaviors.

Graph representations add depth by specifying connections that might not be obvious in a tabular format. E.g., understanding who traveled together, who shared a cabin, or who had social or family ties can provide more context on survival rates & passenger behavior. This is crucial for tasks e.g. node prediction, where we want to predict attributes or outcomes based on relationships represented in graph.

– Cách dữ liệu đồ thị tăng thêm chiều sâu & ý nghĩa. Khi tập dữ liệu được biểu diễn dưới dạng đồ thị, nó cung cấp cái nhìn sâu sắc hơn nhiều về các mối quan hệ xã hội & gia đình giữa các hành khách, ví dụ:

1. *Mối quan hệ gia đình*: Đồ thị cho thấy rõ mối quan hệ của 1 số hành khách nhất định (ví dụ: cha mẹ, con cái hoặc anh chị em ruột). Điều này có thể giúp chúng ta hiểu được các mô hình sinh tồn, vì các thành viên trong gia đình có thể đã hành xử khác nhau trong khủng hoảng so với những cá nhân đi du lịch 1 mình.
2. *Mạng xã hội*: Ngoài gia đình, đồ thị có thể tiết lộ các mạng xã hội rộng hơn (ví dụ: tình bạn hoặc kết nối kinh doanh), đây có thể là những yếu tố quan trọng trong việc phân tích hành vi & kết quả.
3. *Thông tin chi tiết về cộng đồng*: Cấu trúc đồ thị cũng cho phép các thuật toán phát hiện cộng đồng xác định các cụm hành khách có liên quan hoặc kết nối, điều này có thể tiết lộ những hiểu biết mới về tỷ lệ sống sót, mô hình cứu hộ hoặc các hành vi khác.

Biểu diễn đồ thị tăng thêm chiều sâu bằng cách chỉ định các kết nối có thể không rõ ràng ở định dạng bảng. E.g., việc hiểu rõ ai đi cùng nhau, ai ở chung cabin, hoặc ai có mối quan hệ xã hội hoặc gia đình có thể cung cấp thêm bối cảnh về tỷ lệ sống sót & hành vi của hành khách. Điều này rất quan trọng đối với các tác vụ như dự đoán nút, trong đó chúng ta muốn dự đoán các thuộc tính hoặc kết quả dựa trên các mối quan hệ được biểu diễn trên đồ thị.

By creating an adjacency matrix or defining graph edges & nodes based on relationships in dataset, can transition from simple data analysis to more sophisticated graph-based learning methods.

– Bằng cách tạo ma trận kề hoặc xác định các cạnh & nút đồ thị dựa trên các mối quan hệ trong tập dữ liệu, có thể chuyển đổi từ phân tích dữ liệu đơn giản sang các phương pháp học dựa trên đồ thị phức tạp hơn.

- o 1.3. GNN applications: Case studies. GNNs are neural networks designed to work on relational data. They give new ways for relational data to be transformed & manipulated, by being easier to scale & more accurate than previous graph-based learning methods. In following, discuss some exciting applications of GNNs, to see, at a high level, how this class of models are solving real-world problems.

– Ứng dụng GNN: Nghiên cứu điển hình. GNN là mạng nơ-ron được thiết kế để hoạt động trên dữ liệu quan hệ. Chúng cung cấp những cách thức mới để chuyển đổi & thao tác dữ liệu quan hệ, nhờ khả năng mở rộng dễ dàng & chính xác hơn so với các phương pháp học dựa trên đồ thị trước đây. Tiếp theo, thảo luận về 1 số ứng dụng thú vị của GNN, để xem xét ở cấp độ tổng quan, cách lớp mô hình này đang giải quyết các vấn đề thực tế.

- * 1.3.1. Recommendation engines. Enterprise graphs can exceed billions of nodes & many billions of edges. On other hand, many GNNs are benchmarked on datasets that consist of fewer than a million nodes. When applying GNNs to large graphs, adjustments of training & inference algorithms & storage techniques all have to be made. (Can learn more about specifics of scaling GNNs in Chap. 7.)

– Công cụ đề xuất. Đồ thị doanh nghiệp có thể vượt quá hàng tỷ nút & hàng tỷ cạnh. Mặt khác, nhiều GNN được đánh giá chuẩn trên các tập dữ liệu có ít hơn 1 triệu nút. Khi áp dụng GNN cho đồ thị lớn, cần phải điều chỉnh các thuật toán huấn luyện & suy luận & kỹ thuật lưu trữ. (Bạn có thể tìm hiểu thêm về các chi tiết cụ thể về việc mở rộng GNN trong Chương 7.)

1 of most well-known industry examples of GNNs is their use as recommendation engines. E.g., Pinterest is a social media platform for finding & sharing images & ideas. there are 2 major concepts to Pinterest’s users: collections or categories of ideas, called *boards* (like a bulletin board); & objects a user wants to bookmark called *pins*. Pins include images, videos, & websites URLs. A user board focused on dogs might then include pins of pet photos, puppy videos, or dog-related website links. A board’s pins aren’t exclusive to it; a pet drawing that was pinned to Dogs board could also be pinned to a Puppies board, as shown in Fig. 1.10: A bipartite graph that is like Pinterest graph. Nodes in this case are pins & boards.

– 1 trong những ví dụ nổi tiếng nhất về GNN trong ngành là việc sử dụng chúng làm công cụ đề xuất. E.g., Pinterest là 1 nền tảng truyền thông xã hội để tìm kiếm & chia sẻ hình ảnh & ý tưởng. Có 2 khái niệm chính đối với người dùng Pinterest: bộ sưu tập hoặc danh mục ý tưởng, được gọi là *boards* (giống như bảng tin); & đối tượng mà người dùng muốn đánh dấu được gọi là *pins*. Ghim bao gồm hình ảnh, video, & URL trang web. 1 bảng người dùng tập trung vào chó có thể bao gồm các ghim ảnh thú cưng, video về chó con hoặc các liên kết trang web liên quan đến chó. Ghim của 1 bảng không chỉ giới hạn ở đó; 1 bức vẽ thú cưng được ghim vào bảng Chó cũng có thể được ghim vào bảng Chó con, như thể hiện trong Hình 1.10: Đồ thị hai phần giống như đồ thị Pinterest. Các nút trong trường hợp này là ghim & boards.

1 way to interpret relationships between pins & boards is as a *bipartite graph*. For Pinterest graph, all pins are connected to boards, but no pin is connected to another pin, & no board is connected to another board. Pins & boards are 2 classes of nodes. Members of these classes can be linked to members of other class, but not to member of same class. Pinterest graph was reported to have 3 billion nodes & 18 billion edges.

– 1 cách để diễn giải mối quan hệ giữa các chân & bảng là sử dụng đồ thị lưỡng đối. Đối với đồ thị Pinterest, tất cả các chân được kết nối với các bảng, nhưng không có chân nào được kết nối với chân khác, & không có bảng nào được kết nối với bảng khác. Chân & bảng là 2 lớp nút. Các thành viên của các lớp này có thể được liên kết với các thành viên của lớp khác, nhưng không thể liên kết với các thành viên của cùng lớp. Đồ thị Pinterest được báo cáo có 3 tỷ nút & 18 tỷ cạnh. PinSage, a graph convolutional network (GCN), was 1 of 1st documented highly scaled GNNs used in an enterprise system [9]. This was used in Pinterest’s recommendation systems to overcome past challenges of applying graph-learning models to massive graphs. Compared to baseline methods, tests on this system showed it improved user engagement by 30%. Specifically, PinSage was used to predict which objects should be recommended to be included in a user’s graph. However, GNNs can also be used to predict what an object is, e.g. whether it contains a dog or mountain, based on the rest of nodes in graph & how they are connected. Do a deep dive on GCNs, of which PinSage is an extension, in Chap. 3.

– PinSage, 1 mạng tích chập đồ thị (GCN), là 1 trong những GNN có quy mô lớn đầu tiên được ghi nhận sử dụng trong hệ thống doanh nghiệp [9]. Mạng này được sử dụng trong các hệ thống đề xuất của Pinterest để vượt qua những thách thức trước đây khi áp dụng các mô hình học đồ thị vào các đồ thị lớn. So với các phương pháp cơ sở, các thử nghiệm trên hệ thống này cho thấy nó đã cải thiện mức độ tương tác của người dùng lên 30%. Cụ thể, PinSage được sử dụng để dự đoán đối tượng nào nên được đề xuất đưa vào đồ thị của người dùng. Tuy nhiên, GNN cũng có thể được sử dụng để dự đoán 1 đối tượng là gì, ví dụ: nó chứa 1 con chó hay 1 ngọn núi, dựa trên các nút còn lại trong đồ thị & cách chúng được kết nối. Hãy tìm hiểu sâu hơn về GCN, trong đó PinSage là 1 phần mở rộng, trong Chương 3.

* 1.3.2. Drug discovery & molecular science. In chemistry & molecular sciences, a prominent problem has been representing molecules in a general, application-agnostic way, & inferring possible interfaces between molecules, e.g. proteins. For molecule representation, can see: drawings of molecules that are common in high school chemistry classes bear resemblance to a graph structure, consisting of nodes (atoms) & edges (atomic bonds), as shown in Fig. 1.11: In this molecule, can see individual atoms as nodes & atomic bonds as edges.

– Khám phá thuốc & khoa học phân tử. Trong hóa học & khoa học phân tử, 1 vấn đề nổi cộm là biểu diễn phân tử theo cách tổng quát, không phụ thuộc vào ứng dụng, & suy ra các giao diện khả dĩ giữa các phân tử, ví dụ như protein. Về biểu diễn phân tử, có thể thấy: các hình vẽ phân tử thường thấy trong các lớp hóa học trung học phổ thông có cấu trúc đồ thị, bao gồm các nút (nguyên tử) & các cạnh (liên kết nguyên tử), như thể hiện trong Hình 1.11: Trong phân tử này, có thể thấy các nguyên tử riêng lẻ là các nút & các liên kết nguyên tử là các cạnh.

Applying GNNs to these structures can, in certain circumstances, outperform traditional “fingerprint” methods for determining properties of a molecule. These traditional methods involve creation of features by domain experts to capture a molecule’s properties, e.g. interpreting presence or absence of certain molecules or atoms [10]. GNNs learn new data-driven features that can be used to group certain molecules together in new & unexpected ways or even to propose new molecules for synthesis. This is extremely important for predicting whether a chemical is toxic or safe for use or whether it has some downstream effects that can affect disease progression. Therefore, GNNs have shown themselves to be incredibly useful in field of drug discovery.

– Việc áp dụng GNN vào các cấu trúc này, trong 1 số trường hợp, có thể vượt trội hơn các phương pháp “dấu vân tay” truyền thống để xác định các đặc tính của 1 phân tử. Các phương pháp truyền thống này liên quan đến việc tạo ra các đặc điểm bởi các chuyên gia trong lĩnh vực để nắm bắt các đặc tính của 1 phân tử, ví dụ: diễn giải sự hiện diện hoặc vắng mặt của 1 số phân tử hoặc nguyên tử nhất định [10]. GNN học các đặc điểm mới dựa trên dữ liệu, có thể được sử dụng để nhóm các phân tử nhất định lại với nhau theo những cách mới & bất ngờ, hoặc thậm chí đề xuất các phân tử mới để tổng hợp. Điều này cực kỳ quan trọng để dự đoán liệu 1 hóa chất có độc hại hay an toàn để sử dụng hay liệu nó có 1 số tác động hạ lưu có thể ảnh hưởng đến sự tiến triển của bệnh hay không. Do đó, GNN đã chứng tỏ mình cực kỳ hữu ích trong lĩnh vực khám phá thuốc.

Drug discovery, especially for GNNs, can be understood as a graph prediction problem. *Graph prediction* tasks are those that require learning & predicting properties about entire graph. For drug discovery, aim: predict properties e.g. toxicity or treatment effectiveness (discriminative) or to suggest entirely new graphs that should be synthesized & tested (generative). To suggest these new graphs, drug discovery methods often combine GNNs with other generative models e.g. variational graph autoencoders (VGAEs), as shown, e.g., in Fig. 1.12: A GNN system used to predict new molecules [11]. Workflow here starts on left with a representation of a molecule as a graph. In middle parts of figure, this graph representation is transformed via a GNN into a latent representation. Latent representation is then transformed back to molecule to ensure: latent space can be decoded (right). Describe VGAEs in more detail in Chap. 5 & show how we can use these to predict molecules.

– Khám phá thuốc, đặc biệt đối với GNN, có thể được hiểu là 1 bài toán dự đoán đồ thị. Nhiệm vụ *Dự đoán đồ thị* là những nhiệm vụ yêu cầu học & dự đoán các thuộc tính về toàn bộ đồ thị. Đối với khám phá thuốc, mục tiêu: dự đoán các thuộc tính, ví dụ như độc tính hoặc hiệu quả điều trị (phân biệt) hoặc đề xuất các đồ thị hoàn toàn mới cần được tổng hợp & thử nghiệm (sinh). Để đề xuất các đồ thị mới này, các phương pháp khám phá thuốc thường kết hợp GNN với các mô hình sinh khác, ví dụ như bộ mã hóa tự động đồ thị biến phân (VGAE), như được hiển thị, ví dụ, trong Hình 1.12: Hệ thống GNN được sử dụng để dự đoán các phân tử mới [11]. Quy trình làm việc ở đây bắt đầu ở bên trái với biểu diễn của 1 phân tử dưới dạng đồ thị. Ở phần giữa của hình, biểu diễn đồ thị này được chuyển đổi thông qua GNN thành biểu diễn tiềm ẩn. Biểu diễn tiềm ẩn sau đó được chuyển đổi trở lại thành phân tử để đảm bảo: không gian tiềm ẩn có thể được giải mã (bên phải). Mô tả VGAE chi tiết hơn trong Chương 5 & chỉ ra cách chúng ta có thể sử dụng chúng để dự đoán các phân tử.

- * 1.3.3. Mechanical reasoning. Develop rudimentary intuition about mechanics & physics of world around us at a remarkably young age & without any formal training in subject. Do not need to write down a set of equations to know how to catch a bouncing ball. Do not even have to be in presence of a physical ball. Given a series of snapshots of a bouncing ball, can predict reasonably well where ball is going to end up.

– Lý luận cơ học. Phát triển trực giác cơ bản về cơ học & vật lý của thế giới xung quanh chúng ta ngay từ khi còn rất nhỏ & mà không cần bất kỳ sự đào tạo chính thức nào về môn học này. Không cần phải viết ra 1 loạt phương trình để biết cách bắt 1 quả bóng đang nảy. Thậm chí không cần phải ở gần 1 quả bóng thực tế. Chỉ cần 1 loạt ảnh chụp nhanh về 1 quả bóng đang nảy, bạn có thể dự đoán khá chính xác vị trí quả bóng sẽ rơi.

While these problems might seem trivial for us, they are critical for many physical industries, including manufacturing & autonomous driving. E.g., autonomous driving systems need to anticipate what will happen in a traffic scene consisting of many moving objects. Until recently, this task was typically treated as a problem of computer vision. However, more recent approaches have begun to use GNNs [12]. These GNN-based methods demonstrate: including relational information, e.g. how limbs are connected, can enable algorithms to develop physical intuition about how a person or animal moves with higher accuracy & less data.

– Mặc dù những vấn đề này có vẻ tầm thường đối với chúng ta, nhưng chúng lại rất quan trọng đối với nhiều ngành công nghiệp vật lý, bao gồm sản xuất & lái xe tự động. E.g., hệ thống lái xe tự động cần dự đoán những gì sẽ xảy ra trong 1 cảnh giao thông gồm nhiều vật thể chuyển động. Cho đến gần đây, nhiệm vụ này thường được coi là 1 vấn đề của thị giác máy tính. Tuy nhiên, các phương pháp tiếp cận gần đây hơn đã bắt đầu sử dụng GNN [12]. Các phương pháp dựa trên GNN này chứng minh: việc bao gồm thông tin quan hệ, ví dụ như cách các chi được kết nối, có thể cho phép các thuật toán phát triển trực giác vật lý về cách 1 người hoặc động vật di chuyển với độ chính xác cao hơn & ít dữ liệu hơn.

In Fig. 1.13: A graph representation of a mechanical body, taken from Sanchez-Gonzalez [13]. Body's segments are represented as nodes, & mechanical forces binding them are edges., give an example of how a body can be thought of as a “mechanical” graph. Input graphs for these physical reasoning systems have elements that reflect problem. E.g., when reasoning about a human or animal body, a graph could consist of nodes that represent points on body where limbs connect. For systems of free bodies, nodes of a graph could be individual objects e.g. bouncing balls. Edges of graph then represent physical relationship (e.g., gravitational forces, elastic springs, or rigid connections) between nodes. Given these inputs, GNNs learn to predict future states of a set of objects without explicitly calling on physical/mechanical laws [13]. These methods are a form of *edge prediction*, i.e., they predict how nodes connect over time. Furthermore, these models have to be dynamic to account for temporal evolution of system. Consider these problems in detail in Chap. 6.

– Trong Hình 1.13: Biểu diễn đồ thị của 1 vật thể cơ học, lấy từ Sanchez-Gonzalez [13]. Các đoạn của vật thể được biểu diễn là các nút, & lực cơ học liên kết chúng là các cạnh., đưa ra 1 ví dụ về cách 1 vật thể có thể được coi là 1 đồ thị “cơ học”. Đồ thị đầu vào cho các hệ thống suy luận vật lý này có các thành phần phản ánh vấn đề. Ví dụ: khi suy luận về cơ thể người hoặc động vật, đồ thị có thể bao gồm các nút biểu diễn các điểm trên cơ thể nơi các chi kết nối. Đối với các hệ thống vật thể tự do, các nút của đồ thị có thể là các vật thể riêng lẻ, ví dụ: quả bóng nảy. Các cạnh của đồ thị sau đó biểu diễn mối quan hệ vật lý (ví dụ: lực hấp dẫn, lò xo đàn hồi hoặc kết nối cứng) giữa các nút. Với các đầu vào này, GNN học cách dự đoán trạng thái tương lai của 1 tập hợp các vật thể mà không cần gọi rõ ràng các định luật cơ học vật lý [13]. Các phương pháp này là 1 dạng *dự đoán cạnh*, tức là chúng dự đoán cách các nút kết nối theo thời gian. Hơn nữa, các mô hình này phải mang tính động để tính đến sự tiến hóa theo thời gian của hệ thống. Hãy xem xét chi tiết những vấn đề này trong Chương 6.

- o 1.4. When to use a GNN? Have explored real-world applications of GNNs, identify some underlying characteristics that make problems suitable for graph-based solutions. While cases of previous section clearly involved data that was naturally modeled as a graph, crucial to recognize that GNNs can also be effectively applied to problems where graph-like nature may not be immediately obvious.

– Khi nào nên sử dụng GNN? Đã khám phá các ứng dụng thực tế của GNN, xác định 1 số đặc điểm cơ bản giúp bài toán phù hợp với các giải pháp dựa trên đồ thị. Mặc dù các trường hợp trong phần trước rõ ràng liên quan đến dữ liệu được mô hình hóa tự nhiên dưới dạng đồ thị, nhưng điều quan trọng là phải nhận ra rằng GNN cũng có thể được áp dụng hiệu quả

cho các bài toán mà bản chất giống đồ thị có thể không rõ ràng ngay lập tức.

So, instead of simply stating GNNs are useful for graph problems, this section will help you recognize patterns & relationships within your data that could benefit from graph-based modeling, even if those relationships aren't immediately apparent. Essentially, there are 3 types of criteria for identifying GNN problems: implicit relationships & interdependencies; high dimensionality & sparsity; & complex nonlocal interactions.

– Vì vậy, thay vì chỉ đơn thuần nêu rằng GNN hữu ích cho các bài toán đồ thị, phần này sẽ giúp bạn nhận ra các mẫu & mối quan hệ trong dữ liệu của mình mà mô hình dựa trên đồ thị có thể mang lại lợi ích, ngay cả khi những mối quan hệ đó không rõ ràng ngay lập tức. Về cơ bản, có 3 loại tiêu chí để xác định các bài toán GNN: mối quan hệ ngầm & phụ thuộc lẫn nhau; tính đa chiều cao & thưa thớt; & tương tác phi cục bộ phức tạp.

* **1.4.1. Implicit relationships & interdependencies.** Graphs are versatile data structures that can model a wide range of relationships. Even when a problem doesn't initially appear to be graph-like, even if your dataset is tabular, it is beneficial to explore whether implicit relationships or interdependencies might exist that could be represented explicitly. Implicit relationships are connections that are not immediately documented or obvious within data but can still play a significant role in understanding underlying patterns & behaviors.

– **Mối quan hệ ngầm & sự phụ thuộc lẫn nhau.** Đồ thị là cấu trúc dữ liệu đa năng có thể mô hình hóa 1 loạt các mối quan hệ. Ngay cả khi 1 vấn đề ban đầu có vẻ không giống đồ thị, ngay cả khi tập dữ liệu của bạn ở dạng bảng, việc khám phá xem liệu có tồn tại các mối quan hệ ngầm hoặc sự phụ thuộc lẫn nhau có thể được biểu diễn 1 cách rõ ràng hay không vẫn rất hữu ích. Mối quan hệ ngầm là những kết nối không được ghi chép ngay lập tức hoặc hiển nhiên trong dữ liệu nhưng vẫn có thể đóng 1 vai trò quan trọng trong việc hiểu các mô hình & hành vi cơ bản.

Key indicators. To determine if your problem might benefit from modeling implicit relationships with graphs, consider whether there are hidden or indirect connections between entities in your dataset. E.g., in customer behavior analysis, customers may appear as independent entities in a tabular dataset containing their purchases, demographics, & other details. However, they could be connected through social media influence, peer recommendations, or shared purchasing patterns, forming an underlying network of interactions.

– **Các chỉ số chính.** Để xác định xem vấn đề của bạn có thể được hưởng lợi từ việc mô hình hóa các mối quan hệ ngầm định bằng biểu đồ hay không, xem xét liệu có các kết nối ẩn hoặc gián tiếp giữa các thực thể trong tập dữ liệu của bạn hay không. Ví dụ: trong phân tích hành vi khách hàng, khách hàng có thể xuất hiện dưới dạng các thực thể độc lập trong 1 tập dữ liệu dạng bảng chứa thông tin mua hàng, nhân khẩu học & các chi tiết khác của họ. Tuy nhiên, họ có thể được kết nối thông qua ảnh hưởng trên mạng xã hội, khuyến nghị của đồng nghiệp hoặc các mô hình mua sắm chung, tạo thành 1 mạng lưới tương tác cơ bản.

Another indicator is presence of entities that share common attributes or activities without a direct or documented relationship. In case of investors, e.g., 2 or more investors may not have any formal connection but might frequently co-invest in same companies under similar conditions. Such patterns of co-investment could indicate a shared strategy or influence. In this scenario, a graph representation can be created where nodes represent individual investors, & edges are formed between nodes when 2 or more investors co-invest in the same company. Additional attributes, e.g., investment size, timing, or types of companies invested in can be added to nodes or edges, allowing GNNs to identify patterns, trends, or even potential collaboration opportunities.

– 1 chỉ báo khác là sự hiện diện của các thực thể có chung thuộc tính hoặc hoạt động mà không có mối quan hệ trực tiếp hoặc được ghi chép lại. E.g., trong trường hợp nhà đầu tư, 2 hoặc nhiều nhà đầu tư có thể không có bất kỳ mối liên hệ chính thức nào nhưng thường xuyên cùng đầu tư vào cùng 1 công ty trong các điều kiện tương tự. Các mô hình đồng đầu tư như vậy có thể chỉ ra 1 chiến lược hoặc ảnh hưởng chung. Trong trường hợp này, có thể tạo biểu đồ biểu diễn, trong đó các nút đại diện cho các nhà đầu tư cá nhân, & các cạnh được hình thành giữa các nút khi 2 hoặc nhiều nhà đầu tư cùng đầu tư vào cùng 1 công ty. Các thuộc tính bổ sung, ví dụ: quy mô đầu tư, thời điểm đầu tư hoặc loại hình công ty được đầu tư, có thể được thêm vào các nút hoặc cạnh, cho phép GNN xác định các mô hình, xu hướng hoặc thậm chí các cơ hội hợp tác tiềm năng.

Additionally, consider whether data involves entities that are interconnected through shared references or co-occurrence patterns. Document & text data may not immediately suggest a graph structure, but if documents cite each other or share common topics or authors, they can be represented as nodes in a graph, with edges reflecting these relationships. Similarly, terms within documents can form co-occurrence networks, which are useful for tasks e.g. keyword extraction, document classification, or topic modeling.

– Ngoài ra, cần nhắc xem dữ liệu có bao gồm các thực thể được kết nối với nhau thông qua các tham chiếu chung hay các mẫu đồng hiện diện hay không. Dữ liệu tài liệu & văn bản có thể không gợi ý ngay lập tức 1 cấu trúc đồ thị, nhưng nếu các tài liệu trích dẫn lẫn nhau hoặc có chung chủ đề hoặc tác giả, chúng có thể được biểu diễn dưới dạng các nút trong đồ thị, với các cạnh phản ánh các mối quan hệ này. Tương tự, các thuật ngữ trong tài liệu có thể tạo thành các mạng đồng hiện diện, hữu ích cho các tác vụ như trích xuất từ khóa, phân loại tài liệu hoặc mô hình hóa chủ đề.

By identifying these key indicators in your data, you can uncover hidden or implicit relationships that can be represented explicitly through graphs. Such representations allow for more advanced analyses using GNNs, which can effectively capture & model these relationships, leading to more accurate predictions & deeper insights into data.

– Bằng cách xác định các chỉ số chính này trong dữ liệu, bạn có thể khám phá các mối quan hệ ẩn hoặc ngầm định có thể được biểu diễn rõ ràng thông qua biểu đồ. Các biểu diễn như vậy cho phép phân tích nâng cao hơn bằng cách sử dụng GNN, có thể nắm bắt hiệu quả & mô hình hóa các mối quan hệ này, dẫn đến dự đoán chính xác hơn & hiểu biết sâu sắc hơn về dữ liệu.

* **1.4.2. High dimensionality & sparsity.** Graph-based models are particularly effective in handling high-dimensional data

where many features may be sparse or missing. These models excel in situations where there are underlying structure connecting sparse entities, allowing for more meaningful analysis & improved performance.

– Các mô hình dựa trên đồ thị đặc biệt hiệu quả trong việc xử lý dữ liệu đa chiều, trong đó nhiều đặc điểm có thể thừa thớt hoặc bị thiếu. Các mô hình này đặc biệt hiệu quả trong các tình huống có cấu trúc cơ bản kết nối các thực thể thừa thớt, cho phép phân tích có ý nghĩa hơn & cải thiện hiệu suất.

Key indicators. To determine if your problem involves high-dimensional & sparse data suitable for GNNs, consider whether your dataset contains numerous entities with limited direct interactions or relationships. E.g., in recommender systems, user-item interaction data may appear tabular, but it is inherently sparse – most users only interact with a small subset of available items. By representing users & items as nodes & representing their interactions (e.g., purchases or clicks) as edges, GNNs can exploit network effects to make more accurate recommendations. These models can also address cold-start problem by uncovering both explicit & implicit relationships, leading to better performance in recommending new items to users or engaging new users with existing items.

– **Các chỉ số chính.** Để xác định xem vấn đề của bạn có liên quan đến dữ liệu đa chiều & thừa thớt phù hợp với GNN hay không, xem xét liệu tập dữ liệu của bạn có chứa nhiều thực thể với các tương tác hoặc mối quan hệ trực tiếp hạn chế hay không. Ví dụ: trong các hệ thống đề xuất, dữ liệu tương tác giữa người dùng & sản phẩm có thể xuất hiện dưới dạng bảng, nhưng bản chất của nó là thừa thớt – hầu hết người dùng chỉ tương tác với 1 tập hợp con nhỏ các sản phẩm khả dụng. Bằng cách biểu diễn người dùng & sản phẩm dưới dạng các nút & biểu diễn các tương tác của họ (ví dụ: mua hàng hoặc nhấp chuột) dưới dạng các cạnh, GNN có thể khai thác hiệu ứng mạng để đưa ra các đề xuất chính xác hơn. Các mô hình này cũng có thể giải quyết vấn đề khởi động nguội bằng cách khám phá cả các mối quan hệ rõ ràng & ngầm định, dẫn đến hiệu suất tốt hơn trong việc đề xuất sản phẩm mới cho người dùng hoặc thu hút người dùng mới sử dụng các sản phẩm hiện có.

Another indicator that your problem may be suitable for graph-based models is when data represents entities that are sparsely connected but share significant characteristics. In drug discovery, e.g., molecules are represented as graphs, with atoms as nodes & chemical bonds as edges. This representation captures inherent sparsity of molecular structures, where most atoms form only a few bonds, & large portions of molecule may be distant from each other in graph. Traditional ML methods often struggle to predict properties of new molecules due to this sparsity, as they don't account for full structural context.

– 1 dấu hiệu khác cho thấy vấn đề của bạn có thể phù hợp với các mô hình dựa trên đồ thị là khi dữ liệu biểu diễn các thực thể được kết nối thừa thớt nhưng có chung các đặc điểm quan trọng. E.g., trong khám phá thuốc, các phân tử được biểu diễn dưới dạng đồ thị, với các nguyên tử là nút & các liên kết hóa học là cạnh. Cách biểu diễn này nắm bắt được tính thừa thớt vốn có của các cấu trúc phân tử, trong đó hầu hết các nguyên tử chỉ tạo thành 1 vài liên kết, & các phần lớn phân tử có thể nằm cách xa nhau trên đồ thị. Các phương pháp ML truyền thống thường gặp khó khăn trong việc dự đoán các đặc tính của các phân tử mới do tính thừa thớt này, vì chúng không tính đến toàn bộ bối cảnh cấu trúc.

Graph-based models, particularly GNNs, overcome these challenges by capturing both local atomic environments & global molecular structures. GNNs learn hierarchical features from fine-grained atomic interactions to broader molecular properties, & their ability to remain invariant to ordering of atoms ensures consistent predictions. By using graph structure of molecules, GNNs make accurate predictions from sparse, connected data, thereby accelerating drug discovery process.

– Các mô hình dựa trên đồ thị, đặc biệt là GNN, khắc phục những thách thức này bằng cách nắm bắt cả môi trường nguyên tử cục bộ & cấu trúc phân tử toàn cục. GNN học các đặc điểm phân cấp từ các tương tác nguyên tử chi tiết đến các đặc tính phân tử rộng hơn, & khả năng duy trì tính bất biến theo thứ tự nguyên tử đảm bảo các dự đoán nhất quán. Bằng cách sử dụng cấu trúc đồ thị của phân tử, GNN đưa ra các dự đoán chính xác từ dữ liệu thừa thớt và kết nối, do đó đẩy nhanh quá trình khám phá thuốc.

By recognizing these key indicators in your data, you can identify situations where graph-based models can effectively handle high-dimensional & sparse datasets. Representing such data as graphs allows GNNs to capture & use underlying structures, resulting in more accurate predictions & deeper insights across various applications.

– Bằng cách nhận diện các chỉ số chính này trong dữ liệu, bạn có thể xác định các tình huống mà mô hình dựa trên đồ thị có thể xử lý hiệu quả các tập dữ liệu đa chiều & thừa thớt. Việc biểu diễn dữ liệu dưới dạng đồ thị cho phép GNN nắm bắt & sử dụng các cấu trúc cơ bản, mang lại dự đoán chính xác hơn & hiểu biết sâu sắc hơn trên nhiều ứng dụng khác nhau.

* **1.4.3. Complex, nonlocal interactions.** Certain problems require underlying how distant elements in a dataset influence each other. In these cases, GNNs provide a framework to capture these complex interactions, where predicted value or label of a particular data point depends not just on features of its immediate neighbors but also on those of other related data points. This capability is especially useful when relationships extend beyond direct connections to involve multiple levels or degrees of separation.

– 1 số vấn đề đòi hỏi phải hiểu rõ cách các phần tử ở xa trong 1 tập dữ liệu ảnh hưởng lẫn nhau. Trong những trường hợp này, mạng nơ-ron nhân tạo (GNN) cung cấp 1 khuôn khổ để nắm bắt những tương tác phức tạp này, trong đó giá trị dự đoán hoặc nhãn của 1 điểm dữ liệu cụ thể không chỉ phụ thuộc vào các đặc điểm của các điểm lân cận mà còn phụ thuộc vào các đặc điểm của các điểm dữ liệu liên quan khác. Khả năng này đặc biệt hữu ích khi các mối quan hệ vượt ra ngoài các kết nối trực tiếp, bao gồm nhiều cấp độ hoặc mức độ phân tách.

However, some standard GNNs, which rely primarily on local message passing, may struggle to capture long-range dependencies effectively. Advanced architectures or modifications, e.g. those incorporating global attention, nonlocal aggregation, or hierarchical message-passing, can better address these challenges [14].

Key indicators. To determine if your problem involves complex, nonlocal interactions suitable for GNNs, consider

whether outcome or behavior of 1 entity depends on attributes or actions of identities that are not directly connected to it but may be indirectly connected through other entities. E.g., in supply chain optimization, a delay in 1 supplier may not only affect its immediate downstream customers but could cascade through multiple levels of network, influencing distributors & final consumers.

– **Các chỉ số chính.** Để xác định xem vấn đề của bạn có liên quan đến các tương tác phức tạp, phi cục bộ phù hợp với GNN hay không, xem xét liệu kết quả hoặc hành vi của 1 thực thể có phụ thuộc vào các thuộc tính hoặc hành động của các danh tính không được kết nối trực tiếp với nó nhưng có thể được kết nối gián tiếp thông qua các thực thể khác hay không. E.g., trong tối ưu hóa chuỗi cung ứng, sự chậm trễ của 1 nhà cung cấp không chỉ ảnh hưởng đến các khách hàng hạ nguồn trực tiếp mà còn có thể lan truyền qua nhiều cấp độ mạng lưới, ảnh hưởng đến các nhà phân phối & người tiêu dùng cuối cùng.

Another indicator is whether problem involves scenarios where information, influence, or effects propagate through a network over time. In healthcare & epidemiology, e.g., a disease outbreak might spread from a small cluster of patients through their interactions with shared healthcare providers, common environments, or overlapping social networks. Such propagation requires an approach that captures indirect transmission pathways of information or effects.

– 1 chỉ báo khác là liệu vấn đề có liên quan đến các kịch bản mà thông tin, ảnh hưởng hoặc tác động lan truyền qua mạng lưới theo thời gian hay không. Trong chăm sóc sức khỏe & dịch tễ học, ví dụ, 1 đợt bùng phát dịch bệnh có thể lây lan từ 1 nhóm nhỏ bệnh nhân thông qua tương tác của họ với các nhà cung cấp dịch vụ chăm sóc sức khỏe chung, môi trường chung hoặc các mạng lưới xã hội chồng chéo. Sự lan truyền như vậy đòi hỏi 1 phương pháp tiếp cận nắm bắt các con đường truyền thông tin hoặc tác động gián tiếp.

To close this section, in determining whether your problem is a good candidate for a GNN, ask yourself these questions:

1. Are there implicit relationships or interdependencies in my data that I could model?
2. Do interactions between entities exhibit complex, nonlocal dependencies that go beyond immediate connections?
3. Is data high-dimensional & sparse, with a need to capture underlying relational structures?

If answer to any of these questions is yes, consider framing your problem as a graph & applying GNNs to unlock new insights & predictive capabilities.

– Để kết thúc phần này, khi xác định xem vấn đề của bạn có phù hợp để áp dụng mô hình mạng nơ-ron nhân tạo (GNN) hay không, tự hỏi mình những câu hỏi sau:

1. Liệu có mối quan hệ ngầm định hoặc phụ thuộc lẫn nhau nào trong dữ liệu của tôi mà tôi có thể mô hình hóa không?
2. Liệu các tương tác giữa các thực thể có biểu hiện sự phụ thuộc phức tạp, phi cục bộ, vượt ra ngoài các kết nối tức thời không?
3. Liệu dữ liệu có đa chiều & thưa thớt, cần phải nắm bắt các cấu trúc quan hệ cơ bản không?

Nếu câu trả lời cho bất kỳ câu hỏi nào trong số này là có, cân nhắc việc định hình vấn đề của bạn dưới dạng biểu đồ & áp dụng GNN để khám phá những hiểu biết mới & khả năng dự đoán.

- o 1.5. Understanding how GNNs operate. In this section, explore how GNNs work, starting from initial collection of raw data to final deployment of trained models. Examine each step, highlighting processes of data handling, model building, & unique message-passing technique that sets GNNs apart from traditional DL models.

* 1.5.1. Mental model for training a GNN. Our mental model covers data sourcing, graph representation, preprocessing, & model development workflow. Start with raw data & end up with a trained GNN model & its outputs. Fig. 1.14: Mental model of GNN project. Start with raw data, which is transformed into a graph data model that can be stored in a graph database or used in a graph processing system. From graph processing system (& some graph databases), exploratory data analysis & visualization can be done. Finally, for graph ML, data is preprocessed into a form that can be submitted for training illustrates & visualizes topics related to these stages, annotated with chaps in which these topics appear.

– Mô hình tinh thần để huấn luyện GNN. Mô hình tinh thần của chúng tôi bao gồm việc tìm nguồn dữ liệu, biểu diễn đồ thị, tiền xử lý và quy trình phát triển mô hình. Bắt đầu với dữ liệu thô và kết thúc bằng 1 mô hình GNN đã được huấn luyện và kết quả của nó. Hình 1.14: Mô hình tinh thần của dự án GNN. Bắt đầu với dữ liệu thô, được chuyển đổi thành mô hình dữ liệu đồ thị có thể được lưu trữ trong cơ sở dữ liệu đồ thị hoặc được sử dụng trong hệ thống xử lý đồ thị. Từ hệ thống xử lý đồ thị (và 1 số cơ sở dữ liệu đồ thị), có thể thực hiện phân tích dữ liệu thăm dò và trực quan hóa. Cuối cùng, đối với học máy đồ thị, dữ liệu được tiền xử lý thành 1 biểu mẫu có thể được gửi để huấn luyện minh họa và trực quan hóa các chủ đề liên quan đến các giai đoạn này, được chú thích bằng các chương trong đó các chủ đề này xuất hiện.

While not all workflows include every step or stage of this process, most will incorporate at least some elements. At different stages of a model development project, different parts of this process will typically be used. E.g., when *training* a model, data analysis & visualization may be needed to make design decisions, but when *deploying* a model, it may only be necessary to stream raw data & quickly preprocess it for ingestion into a model. Though this book touches on earlier stages in this mental model, bulk of book is focused on how to train different types of GNNs. When other topics are discussed, they serve to support this main focus.

– Mặc dù không phải tất cả quy trình làm việc đều bao gồm mọi bước hoặc giai đoạn của quy trình này, nhưng hầu hết đều sẽ kết hợp ít nhất 1 số yếu tố. Ở các giai đoạn khác nhau của 1 dự án phát triển mô hình, các phần khác nhau của quy trình này thường sẽ được sử dụng. Ví dụ: khi *training* 1 mô hình, phân tích dữ liệu & trực quan hóa có thể cần thiết để đưa ra quyết định thiết kế, nhưng khi *deployment* 1 mô hình, có thể chỉ cần truyền dữ liệu thô & xử lý nhanh dữ liệu trước để đưa vào mô hình. Mặc dù cuốn sách này đề cập đến các giai đoạn trước đó của mô hình tư duy này, phần lớn nội dung sách tập trung vào cách huấn luyện các loại GNN khác nhau. Khi các chủ đề khác được thảo luận, chúng sẽ hỗ trợ cho trọng tâm chính này.

Mental model shows core tasks of applying GNNs to ML problems, & we return to this process repeatedly through rest of book. Examine this diagram from end to end.

– Mô hình tinh thần cho thấy các nhiệm vụ cốt lõi của việc áp dụng mạng nơ-ron nhân tạo (GNN) vào các bài toán ML, & chúng ta sẽ quay lại quá trình này nhiều lần trong suốt phần còn lại của cuốn sách. Hãy xem xét sơ đồ này từ đầu đến cuối.

1st step in training a GNN in structuring this raw data into a graph format, if it is not already. This requires deciding which entities in data to represent as nodes & edges, as well as determining features to assign to them. Decision must also be made about data storage – whether to use a graph database, processing system, or other formats.

– Bước đầu tiên trong quá trình huấn luyện GNN là cấu trúc dữ liệu thô này thành định dạng đồ thị, nếu dữ liệu chưa được định dạng. Điều này đòi hỏi phải quyết định những thực thể nào trong dữ liệu sẽ được biểu diễn dưới dạng nút & cạnh, cũng như xác định các đặc điểm cần gán cho chúng. Quyết định cũng cần được đưa ra về lưu trữ dữ liệu – sử dụng cơ sở dữ liệu đồ thị, hệ thống xử lý hay các định dạng khác.

For ML, data must be preprocessed for training & inference, involving tasks e.g. sampling, batching, & splitting data into training, validation, & test sets. Throughout this book, use PyTorch Geometric (PyG), which offers specialized classes for preprocessing & data splitting while preserving graph's structure. Preprocessing is covered in most chaps, with more-in-depth explanations available in Appendix B.

– Đối với ML, dữ liệu phải được xử lý trước để huấn luyện & suy luận, bao gồm các tác vụ như lấy mẫu, xử lý hàng loạt, & chia dữ liệu thành các tập huấn luyện, xác thực, & kiểm tra. Trong suốt cuốn sách này, sử dụng PyTorch Geometric (PyG), cung cấp các lớp chuyên biệt để xử lý trước & chia tách dữ liệu trong khi vẫn bảo toàn cấu trúc đồ thị. Phần lớn các chương đều đề cập đến tiền xử lý, với các giải thích chi tiết hơn có sẵn trong Phụ lục B.

After processing data, can then move on to model training. In this book, cover several architectures & training types:

- Chaps. 2-3 discuss convolutional GNNs, where 1st use a GCN layer to produce graph embeddings (Chap. 2) & then train a full GCN & GraphSAGE models (Chap. 3).
- Chap. 4 explains graph attention networks (GATs), which adds attention to our GNNs.
- Chap. 5 introduces GNNs for unsupervised & generative problems, where we train & use a variational graph autoencoder (VGAE).

- Chap. 6 then explores advanced concept of spatiotemporal GNNs, based on graphs that evolve over time. Train a neural relational inference (NRI) model, which combines an autoencoder structure with a RNN.

Most of examples provided for GNNs mentioned so far are illustrated with code examples which use small-scale graphs that can fit into memory on a laptop or desktop computer.

- In Chap. 7, delve into strategies for handling data that exceeds processing capacity of a single machine.
- In Chap. 8, close with some considerations for graph & GNN projects, e.g. practical aspects of working with graph data, as well as how to convert nongraph data into a graph format.

– Sau khi xử lý dữ liệu, có thể chuyển sang huấn luyện mô hình. Trong cuốn sách này, chúng tôi sẽ đề cập đến 1 số kiến trúc & loại huấn luyện:

- Chương 2-3 thảo luận về mạng GNN tích chập, trong đó đầu tiên sử dụng 1 lớp GCN để tạo nhúng đồ thị (Chương 2) & sau đó huấn luyện 1 mô hình GCN đầy đủ & GraphSAGE (Chương 3).
- Chương 4 giải thích về mạng chú ý đồ thị (GAT), giúp tăng cường sự chú ý cho các GNN của chúng tôi.
- Chương 5 giới thiệu GNN cho các bài toán không giám sát & sinh, trong đó chúng tôi huấn luyện & sử dụng bộ mã hóa tự động đồ thị biến phân (VGAE).
- Chương 6 sau đó khám phá khái niệm nâng cao về mạng GNN không gian - thời gian, dựa trên các đồ thị phát triển theo thời gian. Huấn luyện 1 mô hình suy luận quan hệ thần kinh (NRI), kết hợp cấu trúc bộ mã hóa tự động với RNN. Hầu hết các ví dụ được cung cấp cho GNN đã đề cập cho đến nay đều được minh họa bằng các ví dụ mã sử dụng đồ thị quy mô nhỏ, có thể chứa trong bộ nhớ của máy tính xách tay hoặc máy tính để bàn.
- Trong Chương 7, đi sâu vào các chiến lược xử lý dữ liệu vượt quá khả năng xử lý của 1 máy tính.
- Trong Chương 8, kết thúc bằng 1 số cân nhắc cho các dự án đồ thị & GNN, ví dụ: các khía cạnh thực tế khi làm việc với dữ liệu đồ thị, cũng như cách chuyển đổi dữ liệu không phải đồ thị sang định dạng đồ thị.

- * 1.5.2. Unique mechanisms of a GNN model. Although there are a variety of GNN architectures at this point, they all tackle same problem of dealing with graph data in a way that is permutation invariant. They do this via encoding & exchanging information across graph structure during learning process.

– Cơ chế độc đáo của mô hình GNN. Mặc dù hiện tại có nhiều kiến trúc GNN khác nhau, tất cả đều giải quyết cùng 1 vấn đề là xử lý dữ liệu đồ thị theo cách bất biến hoán vị. Chúng thực hiện điều này thông qua việc mã hóa & trao đổi thông tin trên toàn bộ cấu trúc đồ thị trong quá trình học.

In a conventional neural network CNN, 1st need to initialize a set of parameters & functions. These include number of layers, size of layers, learning rate, loss function, batch size, & other hyperparameters. (These are all treated in detail in other books on DL, so assume familiar with those terms). Once defined these features, then train our network by iteratively updating weights of network, as shown in Fig. 1.15: Process for training a GNN, which is similar to training most other DL models.

– Trong 1 mạng nơ-ron nhân tạo thông thường (CNN), trước tiên cần khởi tạo 1 tập hợp các tham số & hàm. Chúng bao gồm số lớp, kích thước lớp, tốc độ học, hàm mất mát, kích thước lô, & các siêu tham số khác. (Tất cả những điều này đều được trình bày chi tiết trong các sách khác về DL, vì vậy coi như bạn đã quen thuộc với các thuật ngữ đó). Sau khi

xác định các đặc điểm này, huấn luyện mạng bằng cách cập nhật trọng số của mạng theo từng bước, như thể hiện trong Hình 1.15: Quy trình huấn luyện GNN, tương tự như huấn luyện hầu hết các mô hình DL khác.

Explicitly, perform following steps:

1. Input our data.
2. Pass data through neural network layers that transform data according to parameters of layer & an activation rule.
3. Output a representation from final layer of network.
4. Backpropagation error, & adjust parameters accordingly.
5. Repeat these steps a fixed number of *epochs* (process by which data is passed forward & backward to train a neural network).

For tabular data, these steps are exactly as listed, as shown in Fig. 1.16: Comparison of (simple) non-GNN & GNN. GNNs have a layer that distributes data among its vertices. For graph-based or relational data, these steps are similar except that each epoch relates to 1 iteration of message passing.

– Thực hiện các bước sau 1 cách rõ ràng:

1. Nhập dữ liệu.
2. Truyền dữ liệu qua các lớp mạng nơ-ron để biến đổi dữ liệu theo các tham số của lớp & 1 quy tắc kích hoạt.
3. Xuất ra 1 biểu diễn từ lớp cuối cùng của mạng.
4. Lỗi lan truyền ngược, & điều chỉnh các tham số cho phù hợp.
5. Lặp lại các bước này với số lượng cố định *epoch* (quy trình mà dữ liệu được truyền tới & lùi để huấn luyện mạng nơ-ron).

Đối với dữ liệu dạng bảng, các bước này chính xác như được liệt kê trong Hình 1.16: So sánh (đơn giản) không phải GNN & GNN. GNN có 1 lớp phân phối dữ liệu giữa các đỉnh của nó. Đối với dữ liệu dựa trên đồ thị hoặc dữ liệu quan hệ, các bước này tương tự nhau, ngoại trừ việc mỗi epoch liên quan đến 1 lần lặp truyền thông điệp.

* 1.5.3. **Message passing.** *Message passing*, which is touched on throughout book, is a central mechanism in GNNs that enables nodes to communicate & share information across a graph [15]. This process allows GNNs to learn rich, informative representations of graph-structured data, which is essential for tasks e.g. node classification, link prediction, & graph-level prediction. Fig. 1.17: Elements of our message passing layer. Each message passing layer consists of an aggregation, a transformation, & an update step: 1. Input initial graph with node, edges, & features. 2. Collect all features from neighboring nodes, known as messages, for each node. 3. Aggregate messages using invariant functions e.g. sum, max, or mean. 4. Transform messages using a neural network to create new node features. 5. Update all features in graph with new node features. illustrates steps involved in typical message-passing layer.

– Truyền tin nhắn. *Truyền tin nhắn*, được đề cập trong toàn bộ cuốn sách, là 1 cơ chế trung tâm trong GNN cho phép các nút giao tiếp & chia sẻ thông tin trên 1 đồ thị [15]. Quá trình này cho phép GNN học các biểu diễn phong phú, nhiều thông tin về dữ liệu có cấu trúc đồ thị, điều này rất cần thiết cho các tác vụ ví dụ như phân loại nút, dự đoán liên kết, & dự đoán cấp đồ thị. Hình 1.17: Các thành phần của lớp truyền tin nhắn của chúng tôi. Mỗi lớp truyền tin nhắn bao gồm 1 phép tổng hợp, 1 phép biến đổi, & 1 bước cập nhật: 1. Đầu vào đồ thị ban đầu với nút, cạnh, & các đặc điểm. 2. Thu thập tất cả các đặc điểm từ các nút lân cận, được gọi là các thông điệp, cho mỗi nút. 3. Tổng hợp các thông điệp bằng các hàm bất biến ví dụ như tổng, cực đại hoặc trung bình. 4. Biến đổi các thông điệp bằng mạng nơ-ron để tạo các đặc điểm nút mới. 5. Cập nhật tất cả các đặc điểm trong đồ thị bằng các đặc điểm nút mới. minh họa các bước liên quan đến lớp truyền tin nhắn thông thường.

Message-passing process begins with Input (step 1) of initial graph, where every node & edge have their own features. In Collect step (step 2), each node gathers information from its immediate neighbors – these pieces of information are referred to as “messages”. This step ensures that each node has access to features of its neighbors, which are crucial for understanding local graph structure. Next, in Aggregate step (step 3), collected messages from neighboring nodes are combined using an invariant function, e.g. sum, mean, or max. This aggregation consolidates information from a node’s neighborhood into a single vector, capturing most relevant details about its local environment.

– Quá trình truyền thông điệp bắt đầu với Đầu vào (bước 1) của đồ thị khởi tạo, trong đó mỗi nút & cạnh đều có các đặc trưng riêng. Trong bước Thu thập (bước 2), mỗi nút thu thập thông tin từ các nút lân cận trực tiếp của nó – những thông tin này được gọi là “thông điệp”. Bước này đảm bảo rằng mỗi nút có quyền truy cập vào các đặc trưng của các nút lân cận, điều này rất quan trọng để hiểu cấu trúc đồ thị cục bộ. Tiếp theo, trong bước Tổng hợp (bước 3), các thông điệp được thu thập từ các nút lân cận được kết hợp bằng 1 hàm bất biến, ví dụ: tổng, trung bình hoặc tối đa. Quá trình tổng hợp này hợp nhất thông tin từ vùng lân cận của 1 nút thành 1 vectơ duy nhất, nắm bắt các chi tiết quan trọng nhất về môi trường cục bộ của nó.

In Transform step (step 4), aggregated messages are processed by a neural network to produce a new representation for each node. This transformation allows GNN to learn complex interactions & patterns within graph by applying nonlinear functions to aggregated information.

– Trong bước Biến đổi (bước 4), các thông điệp tổng hợp được xử lý bởi mạng nơ-ron để tạo ra 1 biểu diễn mới cho mỗi nút. Phép biến đổi này cho phép GNN học các tương tác phức tạp & các mẫu trong đồ thị bằng cách áp dụng các hàm phi tuyến tính vào thông tin tổng hợp.

Finally, during Update step (step 5), features of each node in graph are replaced or updated with these new representations. This completes 1 round of message passing, incorporating information from neighboring nodes to refine each node’s features.

– Cuối cùng, trong bước Cập nhật (bước 5), các đặc trưng của mỗi nút trong đồ thị được thay thế hoặc cập nhật bằng các biểu diễn mới này. Điều này hoàn tất 1 vòng truyền thông điệp, kết hợp thông tin từ các nút lân cận để tinh chỉnh các đặc trưng của từng nút.

Each message-passing layer in a GNN allows nodes to gather information from nodes that are further away, or more “hops” away, in graph. Repeating these steps over multiple layers enables GNN to capture more complex dependencies & long-range interactions within graph.

– Mỗi lớp truyền thông điệp trong GNN cho phép các nút thu thập thông tin từ các nút ở xa hơn, hoặc cách xa hơn “các bước nhảy”, trong đồ thị. Việc lặp lại các bước này trên nhiều lớp cho phép GNN nắm bắt các mối quan hệ phụ thuộc phức tạp hơn & các tương tác tầm xa trong đồ thị.

By using message passing, GNNs efficiently encode graph structure & data into useful representations for a variety of downstream tasks. Advanced architectures, e.g. those incorporating global attention or hierarchical message passing, further enhance model’s ability to capture long-range dependencies across graph, enabling more robust performance on diverse applications.

– Bằng cách sử dụng kỹ thuật truyền thông điệp, GNN mã hóa hiệu quả cấu trúc đồ thị & dữ liệu thành các biểu diễn hữu ích cho nhiều tác vụ hạ nguồn. Các kiến trúc tiên tiến, ví dụ như kiến trúc tích hợp sự chú ý toàn cục hoặc truyền thông điệp phân cấp, sẽ nâng cao hơn nữa khả năng của mô hình trong việc nắm bắt các phụ thuộc tầm xa trên toàn bộ đồ thị, cho phép hiệu suất mạnh mẽ hơn trên nhiều ứng dụng khác nhau.

o Summary.

* GNNs are specialized tools for handling relational, or relationship-centric, data, particularly in scenarios where traditional neural networks struggle due to complexity & diversity of graph structures.

– GNN là công cụ chuyên dụng để xử lý dữ liệu quan hệ hoặc dữ liệu tập trung vào mối quan hệ, đặc biệt là trong các tình huống mà mạng nơ-ron truyền thống gặp khó khăn do tính phức tạp & đa dạng của cấu trúc đồ thị.

* GNNs have found significant applications in areas e.g. recommendation engines, drug discovery, & mechanical reasoning, showcasing their versatility in handling large & complex relational data for enhanced insights & predictions.

– GNN đã tìm thấy những ứng dụng quan trọng trong các lĩnh vực như công cụ đề xuất, khám phá thuốc, & suy luận cơ học, thể hiện tính linh hoạt của chúng trong việc xử lý dữ liệu quan hệ phức tạp & lớn để có được những hiểu biết sâu sắc & dự đoán tốt hơn.

* Specific GNN tasks include node prediction, edge prediction, graph prediction, & graph representation through embedding techniques.

– Các nhiệm vụ cụ thể của GNN bao gồm dự đoán nút, dự đoán cạnh, dự đoán đồ thị và biểu diễn đồ thị thông qua các kỹ thuật nhúng.

* Specific GNN tasks include node prediction, edge prediction, graph prediction, & graph representation through embedding techniques.

– Các nhiệm vụ cụ thể của GNN bao gồm dự đoán nút, dự đoán cạnh, dự đoán đồ thị và biểu diễn đồ thị thông qua các kỹ thuật nhúng.

* GNNs are best used when data is represented as a graph, indicating a strong emphasis on relationships & connections between data points. They are not ideal for individual, standalone data entries where relational information is insignificant.

– GNN được sử dụng tốt nhất khi dữ liệu được biểu diễn dưới dạng đồ thị, thể hiện sự nhấn mạnh vào mối quan hệ & kết nối giữa các điểm dữ liệu. Chúng không lý tưởng cho các mục dữ liệu riêng lẻ, độc lập, nơi thông tin quan hệ không đáng kể.

* When deciding if a GNN solution is a good fit for your problem, consider cases that have characteristics e.g. implicit relationships, high-dimensionality, sparsity, & complex nonlocal interactions. By understanding these fundamentals, practitioners can evaluate suitability of GNNs for their specific problems, implement them effectively, & recognize their tradeoffs & limitations in real-world applications.

– Khi quyết định xem giải pháp GNN có phù hợp với vấn đề của bạn hay không, xem xét các trường hợp có đặc điểm như mối quan hệ ngầm định, đa chiều, thưa thớt, & tương tác phi cục bộ phức tạp. Bằng cách hiểu những nguyên tắc cơ bản này, người thực hành có thể đánh giá tính phù hợp của GNN cho các vấn đề cụ thể, triển khai chúng 1 cách hiệu quả, & nhận ra những đánh đổi & hạn chế của chúng trong các ứng dụng thực tế.

* Messages passing is a core mechanism of GNNs which enables them to encode & exchange information across a graph’s structure, allowing for meaningful node, edge, & graph-level predictions. Each layer of a GNN represents 1 step of message passing, with various aggregation functions to combine messages effectively, providing insights & representations useful for ML tasks.

– Truyền thông điệp là 1 cơ chế cốt lõi của GNN, cho phép chúng mã hóa & trao đổi thông tin trên toàn bộ cấu trúc đồ thị, cho phép đưa ra các dự đoán có ý nghĩa ở cấp độ nút, cạnh và đồ thị. Mỗi lớp của GNN đại diện cho 1 bước truyền thông điệp, với nhiều hàm tổng hợp khác nhau để kết hợp các thông điệp 1 cách hiệu quả, cung cấp thông tin chi tiết & biểu diễn hữu ích cho các tác vụ ML.

● 2. Graph embeddings. Covers:

1. Exploring graph embeddings & their importance
2. Creating node embeddings using non-GNN & GNN methods
3. Comparing node embeddings on a semi-supervised problem

4. Taking a deeper dive into embedding methods

Graph embeddings are essential tools in graph-based ML. They transform intricate structure of graphs – be it the entire graph, individual nodes, or edges – into a more manageable, lower-dimensional space. Do this to compress a complex dataset into a form that is easier to work with, without losing its inherent patterns & relationships, information to which we will apply a GNN or other ML method.

– Bao gồm:

1. Khám phá những đồ thị & tầm quan trọng của chúng
2. Tạo những nút bằng phương pháp không phải GNN & GNN
3. So sánh những nút trong bài toán bán giám sát
4. Đi sâu hơn vào các phương pháp những

Những đồ thị là công cụ thiết yếu trong học máy dựa trên đồ thị. Chúng chuyển đổi cấu trúc phức tạp của đồ thị – có thể là toàn bộ đồ thị, từng nút hoặc cạnh – thành 1 không gian ít chiều hơn, dễ quản lý hơn. Thực hiện điều này để nén 1 tập dữ liệu phức tạp thành 1 dạng dễ làm việc hơn, mà không làm mất các mẫu & mối quan hệ vốn có của nó, thông tin mà chúng ta sẽ áp dụng GNN hoặc phương pháp học máy khác.

Graphs encapsulate relationships & interactions within networks, whether they are social networks, biological networks, or any system where entities are interconnected. Embeddings capture these real-life relationships in a compact form, facilitating tasks e.g. visualization, clustering, or predictive modeling.

– Đồ thị gói gọn các mối quan hệ & tương tác trong các mạng lưới, dù là mạng xã hội, mạng sinh học hay bất kỳ hệ thống nào mà các thực thể được kết nối với nhau. Những nắm bắt các mối quan hệ thực tế này 1 cách cô đọng, tạo điều kiện thuận lợi cho các tác vụ như trực quan hóa, phân cụm hoặc mô hình hóa dự đoán.

There are numerous strategies to derive these embeddings, each with its unique approach & application: from classical graph algorithms that use network's topology, to linear algebra techniques that decompose matrices representing graph, & more advanced methods e.g. GNNs [1]. GNNs stand out because they can integrate embedding process directly into learning algorithm itself.

– Có nhiều chiến lược để tạo ra các những này, mỗi chiến lược có cách tiếp cận & ứng dụng riêng: từ các thuật toán đồ thị cổ điển sử dụng cấu trúc mạng, đến các kỹ thuật đại số tuyến tính phân tích các ma trận biểu diễn đồ thị, & các phương pháp tiên tiến hơn, ví dụ như GNN [1]. GNN nổi bật vì chúng có thể tích hợp quá trình những trực tiếp vào chính thuật toán học.

In traditional ML workflows, embeddings are generated as a separate step, serving as a dimensionality-reduction technique in tasks e.g. regression or classification. However, GNNs merge embedding generation with model's learning process. As network processes inputs through its layers, embeddings are refined & updated, making learning phase & embedding phase inseparable. I.e., GNNs learn most informative representative of graph data during training time.

– Trong quy trình làm việc ML truyền thống, những được tạo ra như 1 bước riêng biệt, đóng vai trò là kỹ thuật giảm chiều trong các tác vụ như hồi quy hoặc phân loại. Tuy nhiên, GNN kết hợp việc tạo những với quy trình học của mô hình. Khi mạng xử lý dữ liệu đầu vào qua các lớp của nó, các những được tinh chỉnh & cập nhật, khiến giai đoạn học & giai đoạn những trở nên không thể tách rời. Tức là, GNN học dữ liệu biểu diễn đồ thị mang tính thông tin nhất trong thời gian đào tạo.

Using graph embeddings can significantly enhance your DS & ML projects, especially when dealing with complex networked data. By capturing essence of graph in a lower-dimensional space, embeddings make it feasible to apply a variety of other ML techniques to graph data, opening up a world of possibilities for analysis & model building.

– Việc sử dụng những đồ thị có thể cải thiện đáng kể các dự án DS & ML của bạn, đặc biệt là khi xử lý dữ liệu mạng phức tạp. Bằng cách nắm bắt bản chất của đồ thị trong không gian ít chiều hơn, những cho phép áp dụng nhiều kỹ thuật ML khác vào dữ liệu đồ thị, mở ra 1 thế giới khả thi cho việc phân tích & xây dựng mô hình.

In this chap, begin with an introduction to graph embeddings & a case study on a graph of political book purchases. Start with Node2Vec (N2V) to establish a baseline with a non-GNN approach, guiding you through its practical application. In Sect. 2.2, shift to GNNs, offering a hands-on introduction to GNN-based embeddings, including setup, preprocessing, & visualization. Sect. 2.3 provides a comparative analysis of N2V & GNN embeddings, highlighting their applications. Chap then rounds off with a discussion of theoretical aspects of these embedding methods, with a special focus on principles behind N2V & message-passing mechanism in GNNs. Process we take in this chap is illustrated in Fig. 2.1: Summary of process & objectives in Chap. 2: 1. Preprocess Political Books dataset for embedding. 2. Use N2V & GCN to create embeddings from preprocessed data. 3. Prepare N2V embeddings & GCN embeddings for semi-supervised classification. 4. Embeddings are used as features in a random forest classifier (tabular features) & a GCN classifier (node features).

– Trong chương này, bắt đầu bằng phần giới thiệu về những đồ thị & nghiên cứu điển hình về đồ thị mua sách chính trị. Bắt đầu với Node2Vec (N2V) để thiết lập đường cơ sở với phương pháp không phải GNN, hướng dẫn bạn ứng dụng thực tế. Trong Phần 2.2, chuyển sang GNN, cung cấp phần giới thiệu thực hành về những dựa trên GNN, bao gồm thiết lập, tiền xử lý, & trực quan hóa. Phần 2.3 cung cấp phân tích so sánh về những N2V & GNN, làm nổi bật các ứng dụng của chúng. Sau đó, chương kết thúc bằng phần thảo luận về các khía cạnh lý thuyết của các phương pháp những này, đặc biệt tập trung vào các nguyên tắc đằng sau cơ chế truyền tin nhắn N2V & trong GNN. Quy trình chúng tôi thực hiện trong chương này được minh họa trong Hình 2.1: Tóm tắt các mục tiêu của quy trình & trong Chương 2: 1. Tiền xử lý tập dữ liệu Sách chính trị để những. 2.

Sử dụng N2V & GCN để tạo nhúng từ dữ liệu đã được xử lý trước. 3. Chuẩn bị nhúng N2V & nhúng GCN cho phân loại bán giám sát. 4. Nhúng được sử dụng làm các đặc trưng trong bộ phân loại rừng ngẫu nhiên (các đặc trưng dạng bảng) & bộ phân loại GCN (các đặc trưng dạng nút).

- o 2.1. Creating embeddings with Node2Vec. Understanding relationships within a network is a core task in many fields, from social network analysis to biology & recommendation systems. In this section, explore how to create node embeddings using Node2Vec (N2V), a technique inspired by Word2Vec from NLP [2]. N2V captures context of nodes within a graph by simulating random walks, allowing us to understand neighborhood relationships between nodes in a low-dimensional space. This approach is effective for identifying patterns, clustering similar nodes, & preparing data for ML tasks.

– Tạo nhúng với Node2Vec. Hiểu các mối quan hệ trong mạng là 1 nhiệm vụ cốt lõi trong nhiều lĩnh vực, từ phân tích mạng xã hội đến sinh học & hệ thống khuyến nghị. Trong phần này, khám phá cách tạo nhúng nút bằng Node2Vec (N2V), 1 kỹ thuật lấy cảm hứng từ Word2Vec trong NLP [2]. N2V nắm bắt ngữ cảnh của các nút trong đồ thị bằng cách mô phỏng các bước ngẫu nhiên, cho phép chúng ta hiểu các mối quan hệ lân cận giữa các nút trong không gian ít chiều. Phương pháp này hiệu quả trong việc xác định các mẫu, phân cụm các nút tương tự và chuẩn bị dữ liệu cho các tác vụ ML.

To make this process accessible, use Node2Vec Python library, which is beginner-friendly, although it may be slower on larger graphs. N2V helps create embeddings that capture structural relationships between nodes, which we can then visualize to uncover insights about graph's structure. Our workflow involves several steps:

1. *Load data & set N2V parameters.* Start by loading our graph data & initializing N2V with specific parameters to control random walks, e.g. walk length & number of walks per node.
2. *Create embeddings.* N2V generates node embeddings by performing random walks on graph, effectively summarizing each node's local neighborhood into a vector format.
3. *Transform embeddings.* Resulting embeddings are saved & then transformed into a format suitable for visualization.
4. *Visualize embeddings in 2D.* Use UMAP, a dimensionality reduction technique, to project these embeddings into 2D, making it easier to visualize & interpret results.

– Để quá trình này dễ hiểu hơn, sử dụng thư viện Python Node2Vec, thư viện này thân thiện với người mới bắt đầu, mặc dù có thể chậm hơn trên các đồ thị lớn hơn. N2V giúp tạo các nhúng nắm bắt mối quan hệ cấu trúc giữa các nút, sau đó chúng ta có thể trực quan hóa để khám phá những hiểu biết sâu sắc về cấu trúc của đồ thị. Quy trình làm việc của chúng tôi bao gồm 1 số bước:

1. *Tải dữ liệu & đặt tham số N2V.* Bắt đầu bằng cách tải dữ liệu đồ thị & khởi tạo N2V với các tham số cụ thể để kiểm soát các bước ngẫu nhiên, ví dụ: độ dài bước & số lần bước trên mỗi nút.
2. *Tạo nhúng.* N2V tạo nhúng nút bằng cách thực hiện các bước ngẫu nhiên trên đồ thị, tóm tắt hiệu quả vùng lân cận cục bộ của mỗi nút thành định dạng vector.
3. *Biến đổi nhúng.* Các nhúng kết quả được lưu & sau đó được chuyển đổi thành định dạng phù hợp để trực quan hóa.
4. *Hình dung các nhúng trong 2D.* Sử dụng UMAP, 1 kỹ thuật giảm chiều, để chiếu các nhúng này vào 2D, giúp hình dung & diễn giải kết quả dễ dàng hơn.

Our data is Political Books dataset, which comprises books (nodes) connected by frequent co-purchases on Amazon.com during 2004 US election period (edges) [3]. Using this dataset provides a compelling example of how N2V can reveal underlying patterns in co-purchasing behavior, potentially reflecting broader ideological groupings among book buyers [4]. Table 2.1: Overview of Political Books dataset. provides key information about Political Books graph. The Political Books dataset contains the following:

1. Nodes: represent books about US politics sold by Amazon.com.
2. Edges: indicate frequent co-purchasing by same buyers, as suggested by Amazon's "customers who bought this book also bought these other books" feature.

In Fig. 2.2: Graph visualization of Political Books dataset. Right-leaning books (nodes) are in a lighter shade & are clustered in top half of figure, left-leaning are darker shaded circles & clustered in lower half of figure, & neutral political stance are dark squares & appear in middle. When 2 nodes are connected, it indicates that they have been purchased together frequently on Amazon.com., books are shaded based on their political alignment – darker shade for liberal, lighter shade for conservative, & striped for neutral. Categories were assigned by MARK NEWMAN through a qualitative analysis of book descriptions & reviews posted on Amazon.

– Dữ liệu của chúng tôi là tập dữ liệu Sách Chính trị, bao gồm các cuốn sách (nút) được kết nối bởi các giao dịch mua chung thường xuyên trên Amazon.com trong giai đoạn bầu cử Hoa Kỳ năm 2004 (cạnh) [3]. Việc sử dụng tập dữ liệu này cung cấp 1 ví dụ thuyết phục về cách N2V có thể tiết lộ các mô hình cơ bản trong hành vi mua chung, có khả năng phản ánh các nhóm ý thức hệ rộng hơn giữa những người mua sách [4]. Bảng 2.1: Tổng quan về tập dữ liệu Sách Chính trị. cung cấp thông tin chính về biểu đồ Sách Chính trị. Tập dữ liệu Sách Chính trị chứa các thông tin sau:

1. Nút: biểu diễn các cuốn sách về chính trị Hoa Kỳ do Amazon.com bán.
2. Cạnh: biểu thị việc mua chung thường xuyên của cùng 1 người mua, như được gợi ý bởi tính năng "khách hàng đã mua cuốn sách này cũng đã mua những cuốn sách khác này" của Amazon.

Trong Hình 2.2: Biểu đồ trực quan của tập dữ liệu Sách Chính trị. Sách thiên hữu (nút) được tô sáng hơn & tập trung ở nửa trên của hình, sách thiên tả là các vòng tròn tô đậm hơn & tập trung ở nửa dưới của hình, & lập trường chính trị thần kinh là các ô vuông đậm & xuất hiện ở giữa. Khi 2 nút được kết nối, điều đó cho thấy chúng đã được mua cùng nhau thường xuyên trên

Amazon.com. Sách được tô màu dựa trên khuynh hướng chính trị của chúng – tô đậm hơn cho khuynh hướng tự do, tô nhạt hơn cho khuynh hướng bảo thủ, & có sọc cho khuynh hướng thần kinh. Các danh mục được phân loại bởi MARK NEWMAN thông qua phân tích định tính các mô tả sách & bài đánh giá được đăng trên Amazon.

This dataset, compiled by VALDIS KREBS & available through GNN in Action repository or Carnegie Mellon University website, contains 105 books (nodes) & 441 edges (co-purchases). If want to learn more about background of this dataset, KREBS has written an article with this information [4].

– Bộ dữ liệu này, do VALDIS KREBS biên soạn & có sẵn trên kho lưu trữ GNN in Action hoặc trang web của Đại học Carnegie Mellon, chứa 105 sách (nút) & 441 cạnh (mua chung). Nếu muốn tìm hiểu thêm về bối cảnh của bộ dữ liệu này, KREBS đã viết 1 bài báo với thông tin này [4].

Using N2V, aim to explore structure of this collection of books, uncovering insights based on political learnings & potential associations between different book categories. By visualizing embeddings created by N2V, can gain a better understanding of how books are grouped & which ones might share a common audience, providing valuable insights into consumer behavior during a politically changed period.

– Sử dụng N2V, chúng tôi hướng đến việc khám phá cấu trúc của bộ sưu tập sách này, tìm ra những hiểu biết sâu sắc dựa trên các bài học chính trị & những mối liên hệ tiềm năng giữa các thể loại sách khác nhau. Bằng cách trực quan hóa các phân nhúng do N2V tạo ra, chúng tôi có thể hiểu rõ hơn về cách sách được nhóm lại & những cuốn nào có thể có chung đối tượng độc giả, từ đó cung cấp những hiểu biết giá trị về hành vi người tiêu dùng trong giai đoạn biến động chính trị.

From visualization, data is already clustered in a logical way. This is thanks to *Kamada-Kawai algorithm* graph algorithm, which exploits topological data only without metadata & is useful for visualizing graph. This graph visualization technique positions nodes in a way that reflects their connections, aiming for an arrangement where closely connected nodes are near each other but less connected nodes are farther apart. It achieves this by treating nodes like points connected by springs, iteratively adjusting their positions until “tension” in springs is minimized. This results in a layout that naturally reveals clusters & relationships within graph based purely on its structure.

– Từ trực quan hóa, dữ liệu đã được phân cụm theo 1 cách logic. Điều này là nhờ thuật toán đồ thị *Kamada-Kawai*, thuật toán này chỉ khai thác dữ liệu tô pô mà không cần siêu dữ liệu & rất hữu ích cho việc trực quan hóa đồ thị. Kỹ thuật trực quan hóa đồ thị này định vị các nút theo cách phản ánh kết nối của chúng, hướng đến 1 sự sắp xếp trong đó các nút có kết nối chặt chẽ sẽ ở gần nhau nhưng các nút ít kết nối hơn sẽ ở xa nhau hơn. Nó đạt được điều này bằng cách coi các nút như các điểm được kết nối bởi lò xo, điều chỉnh vị trí của chúng theo từng bước cho đến khi “lực căng” trong lò xo được giảm thiểu. Điều này dẫn đến 1 bố cục tự nhiên thể hiện các cụm & mối quan hệ trong đồ thị chỉ dựa trên cấu trúc của nó.

For Political Books dataset, Kamada-Kawai algorithm helps us visualize books (nodes) based on how often they are co-purchased on Amazon, without using any external information e.g. political alignment or book titles. This gives us an initial view of how books are grouped together by buying behavior. In next steps, use methods e.g. N2V to create embeddings that capture more detailed patterns & further distinguish different book groups.

– Đối với tập dữ liệu Sách Chính trị, thuật toán Kamada-Kawai giúp chúng tôi trực quan hóa các cuốn sách (nút) dựa trên tần suất chúng được mua chung trên Amazon, mà không cần sử dụng bất kỳ thông tin bên ngoài nào, ví dụ như liên kết chính trị hoặc tiêu đề sách. Điều này cung cấp cho chúng tôi cái nhìn ban đầu về cách sách được nhóm lại với nhau theo hành vi mua. Trong các bước tiếp theo, sử dụng các phương pháp, ví dụ như N2V, để tạo các nhúng nắm bắt các mẫu chi tiết hơn & phân biệt rõ hơn các nhóm sách khác nhau.

* 2.1.1. Loading data, setting parameters, & creating embeddings. Use *Node2Vec*, *NetworkX* libraries for our 1st hands-on encounter with graph embeddings. After installing these packages using *pip*, load our dataset’s graph data, which is stored in *.gml* format (Graph Modeling Language, GML), using *NetworkX* library & generate embeddings with *Node2Vec* library.

– **Đang tải dữ liệu, thiết lập tham số, & tạo nhúng.** Sử dụng thư viện *Node2Vec*, *NetworkX* cho lần thực hành đầu tiên với nhúng đồ thị. Sau khi cài đặt các gói này bằng *pip*, tải dữ liệu đồ thị của tập dữ liệu, được lưu trữ ở định dạng *.gml* (Ngôn ngữ Mô hình Đồ thị, GML), bằng thư viện *NetworkX* & tạo nhúng bằng thư viện *Node2Vec*.

GML is a simple, human-readable plain text file format used to represent graph structures. It stores information about nodes, edges, & their attributes in a structured way, making it easy to read & write graph data. E.g., a *.gml* file might contain a list of nodes (e.g., books in our dataset) & edges (connections representing co-purchases) along with additional properties e.g. labels or weights. This format is widely used for exchanging graph data between different software & tools. By loading *.gml* file with *NetworkX*, can easily manipulate & analyze graph in Python.

– GML là 1 định dạng tệp văn bản thuần túy đơn giản, dễ đọc, được sử dụng để biểu diễn các cấu trúc đồ thị. Nó lưu trữ thông tin về các nút, cạnh và thuộc tính của chúng theo 1 cấu trúc nhất định, giúp việc đọc và ghi dữ liệu đồ thị trở nên dễ dàng. Ví dụ: tệp *.gml* có thể chứa danh sách các nút (ví dụ: sách trong tập dữ liệu của chúng tôi) và các cạnh (các kết nối biểu diễn các giao dịch mua chung) cùng với các thuộc tính bổ sung, ví dụ: nhãn hoặc trọng số. Định dạng này được sử dụng rộng rãi để trao đổi dữ liệu đồ thị giữa các phần mềm và công cụ khác nhau. Bằng cách tải tệp *.gml* với *NetworkX*, bạn có thể dễ dàng thao tác và phân tích đồ thị trong Python.

In *Node2Vec* library’s *Node2Vec* function, can use following parameters to specify calculations done & properties of output embedding:

- *Size of embedding (dimensions)*: Think of this as how detailed each node’s profile is, as in how many different traits you are noting down. Standard detail level is 128 traits, but you can tweak this based on how complex you want each node’s profile to be.
- *Length of each walk (Walk Length)*: This is about how far each random walk through your graph goes, with 80 steps being usual journey. If want to see more of neighborhood around a node, increase this number.

- *Number of walks per node (Num Walks)*: This tells us how many times we will take a walk starting from each node. Starting with 10 walks gives a good overview, but if you want a fuller picture of a node's surroundings, consider going on more walks.
- *Backtracking control (Return Parameter p)*: This setting helps decide if our walk should circle back to where it has been. Setting it at 1 keeps things balanced, but adjusting it can make your walks more or less exploratory.
- *Exploration Depth (In-Out Parameter q)*: This one is about choosing between taking in broader neighborhood scene (e.g., a BFS with $q > 1$) or diving deep into specific paths (e.g., a DFS with $q < 1$), with 1 being a mix of both.

Adjust these settings based on what you are looking to understand about your nodes & their connections. Want more depth? Tweak exploration depth. Looking for broader context? Adjust walk length & number of walks. In addition, keep in mind: size of your embeddings should match level of detail you need. In general, it is a good idea to try different combinations of these parameters to see effect on embeddings.

– Trong hàm `Node2Vec` của thư viện `Node2Vec`, bạn có thể sử dụng các tham số sau để chỉ định các phép tính được thực hiện & các thuộc tính của những đầu ra:

- *Kích thước nhúng (kích thước)*: Hãy coi đây là mức độ chi tiết của hồ sơ từng nút, tức là số lượng đặc điểm khác nhau mà bạn đang ghi lại. Mức độ chi tiết tiêu chuẩn là 128 đặc điểm, nhưng bạn có thể điều chỉnh tùy theo độ phức tạp mà bạn muốn hồ sơ của mỗi nút đạt được.
- *Độ dài mỗi lần đi (Độ dài lần đi)*: Đây là khoảng cách mà mỗi lần đi ngẫu nhiên qua đồ thị của bạn đi được, với 80 bước là hành trình thông thường. Nếu muốn xem thêm về vùng lân cận xung quanh 1 nút, tăng con số này.
- *Số lần đi trên mỗi nút (Số lần đi)*: Con số này cho chúng ta biết chúng ta sẽ đi bao nhiêu lần bắt đầu từ mỗi nút. Bắt đầu với 10 lần đi bộ sẽ cho 1 cái nhìn tổng quan tốt, nhưng nếu bạn muốn có cái nhìn đầy đủ hơn về môi trường xung quanh của 1 nút, cân nhắc đi bộ nhiều lần hơn.
- *Điều khiển quay lui (Tham số trả về p)*: Thiết lập này giúp quyết định xem việc đi bộ của chúng ta có nên quay lại vị trí ban đầu hay không. Đặt giá trị này ở mức 1 sẽ giữ mọi thứ cân bằng, nhưng việc điều chỉnh nó có thể khiến việc đi bộ của bạn mang tính khám phá nhiều hơn hoặc ít hơn.
- *Độ sâu khám phá (Tham số vào-ra q)*: Thiết lập này liên quan đến việc lựa chọn giữa việc xem xét toàn cảnh khu vực lân cận rộng hơn (ví dụ: BFS có $q > 1$) hoặc đi sâu vào các đường dẫn cụ thể (ví dụ: DFS có $q < 1$), với 1 là sự kết hợp của cả hai.

Điều chỉnh các thiết lập này dựa trên những gì bạn muốn hiểu về các nút & kết nối của chúng. Muốn có thêm chiều sâu? Hãy tinh chỉnh độ sâu khám phá. Muốn có bối cảnh rộng hơn? Hãy điều chỉnh độ dài & số lần đi bộ. Ngoài ra, nhớ rằng: kích thước nhúng phải phù hợp với mức độ chi tiết bạn cần. Nhìn chung, bạn nên thử kết hợp các thông số này theo nhiều cách khác nhau để xem hiệu quả của việc nhúng.

For this exercise, use 1st 4 parameters. Deeper details on these parameters are found in Sect. 2.4. Code in Listing 2.1: **Generating N2V embeddings** begins by loading graph into a variable called `book_graph`, using `read_gml` method from `NetworkX` library. Next, a N2V model is initialized with loaded graph. This model is set up with specific parameters: it will create 64-dimensional embeddings for each node, use walks of 30 steps along, perform 200 walks starting from each node to gather context, & run these operations in parallel across 4 workers to speed up process.

– Với bài tập này, sử dụng 4 tham số đầu tiên. Chi tiết sâu hơn về các tham số này được tìm thấy trong Mục 2.4. Mã trong Liệt kê 2.1: **Tạo nhúng N2V** bắt đầu bằng cách tải đồ thị vào 1 biến có tên là `book_graph`, sử dụng phương thức `read_gml` từ thư viện `NetworkX`. Tiếp theo, 1 mô hình N2V được khởi tạo với đồ thị đã tải. Mô hình này được thiết lập với các tham số cụ thể: nó sẽ tạo nhúng 64 chiều cho mỗi nút, sử dụng các bước đi 30 bước dọc theo, thực hiện 200 bước đi bắt đầu từ mỗi nút để thu thập ngữ cảnh, & chạy các thao tác này song song trên 4 worker để tăng tốc quá trình.

N2V model is then trained with additional parameters defined in `fit` method. This involves setting a context window size of 10 nodes around each target node to learn embeddings, considering all nodes at least once `min_count = 1`, & processing 4 words (nodes, in this context) each time during training.

– Mô hình N2V sau đó được huấn luyện với các tham số bổ sung được xác định trong phương thức `fit`. Phương thức này bao gồm việc thiết lập kích thước cửa sổ ngữ cảnh là 10 nút xung quanh mỗi nút mục tiêu để học các phép nhúng, xem xét tất cả các nút ít nhất 1 lần `min_count = 1`, & xử lý 4 từ (nút, trong ngữ cảnh này) mỗi lần trong quá trình huấn luyện.

Once trained, access node embeddings using `model`'s `mv` method (reflecting its NLP heritage, `wv` stands for word vectors). For our downstream tasks, we map each node to its embedding using a dictionary comprehension.

– Sau khi được đào tạo, truy cập các nhúng nút bằng phương pháp `mv` của `model` (phản ánh di sản NLP của nó, `wv` là viết tắt của các vectơ từ). Đối với các tác vụ hạ nguồn, chúng tôi ánh xạ từng nút vào nhúng của nó bằng cách sử dụng 1 thuật toán hiểu từ điển.

```
1 import NetworkX as nx
2 from Node2Vec import Node2Vec
3 book_graph = nx.read_gml('polbooks.gml')
4 node2vec = Node2Vec(book_graph, dimensions = 64, walk_length = 30, num_walks = 200, workers = 4)
5 model = node2vec.fit(window = 10, min_count = 1, batch_words = 4)
6 embeddings = {str(node) : model.mv[str(node)] for node in gml_graph.nodes()}
```

1. Loads graph data from a GML file into a NetworkX graph object
2. Initializes N2V model with specified parameters for input graph
3. Trains N2V model

4. Extracts & stores node embeddings generated by N2V model in a dictionary.

- * 2.1.2. **Demystifying embeddings.** Explore what these embeddings are & why they are valuable. An *embedding* is a dense numerical vector that represents identity of a node, edge, or graph in way that captures essential information about its structure & relationships. In our context, an embedding created by N2V captures a node's position & neighborhood within graph using topological information, i.e., it summarizes how node is connected to others, effectively capturing its role & importance in network. Later, when use GNNs to create embeddings, they will also encapsulate node's features, providing an even richer representation that includes both structure & attributes. Get deeper into theoretical aspects of embedding in Sect. 2.4.

– **Giải mã các phép nhúng.** Khám phá các phép nhúng này là gì & tại sao chúng có giá trị. 1 phép nhúng là 1 vectơ số dày đặc biểu diễn danh tính của 1 nút, cạnh hoặc đồ thị theo cách nắm bắt thông tin thiết yếu về cấu trúc & các mối quan hệ của nó. Trong ngữ cảnh của chúng ta, 1 phép nhúng do N2V tạo ra nắm bắt vị trí & lân cận của 1 nút trong đồ thị bằng thông tin tôpô, tức là nó tóm tắt cách nút được kết nối với các nút khác, từ đó nắm bắt hiệu quả vai trò & tầm quan trọng của nút đó trong mạng. Sau này, khi sử dụng GNN để tạo các phép nhúng, chúng cũng sẽ đóng gói các đặc trưng của nút, cung cấp 1 biểu diễn phong phú hơn, bao gồm cả cấu trúc & các thuộc tính. Tìm hiểu sâu hơn về các khía cạnh lý thuyết của phép nhúng trong Phần 2.4.

These embeddings are powerful because they transform complex, high-dimensional graph data into a fixed-size vector format that can be easily in various analyses & ML tasks. E.g., they allow us to perform exploratory data analysis by revealing patterns, clusters, & relationships within graph. Beyond this, embeddings can be directly used as features in ML models, where each dimension of vector represents a distinct feature. This is particularly useful in applications where understanding structure & connections between data points, e.g. in social networks or recommendation systems, can significantly improve model performance.

– Các phép nhúng này rất mạnh mẽ vì chúng chuyển đổi dữ liệu đồ thị phức tạp, nhiều chiều thành định dạng vector có kích thước cố định, có thể dễ dàng thực hiện trong nhiều phân tích & tác vụ ML. Ví dụ: chúng cho phép chúng ta thực hiện phân tích dữ liệu thăm dò bằng cách khám phá các mẫu, cụm, & mối quan hệ trong đồ thị. Hơn thế nữa, các phép nhúng có thể được sử dụng trực tiếp làm đặc trưng trong các mô hình ML, trong đó mỗi chiều của vector đại diện cho 1 đặc trưng riêng biệt. Điều này đặc biệt hữu ích trong các ứng dụng mà việc hiểu cấu trúc & kết nối giữa các điểm dữ liệu, ví dụ như trong mạng xã hội hoặc hệ thống khuyến nghị, có thể cải thiện đáng kể hiệu suất mô hình.

To illustrate, consider node representing book *Losing Bin Laden* in our Political Books dataset. Using command `model.wv['Losing Bin Laden']`, we retrieve its dense vector embedding. This vector, shown in Fig. 2.3: **Extracting embedding for nodes associated with political book *Losing Bin Laden*.** Output is a dense vector represented as a Python list., captures various aspects of book's role within network of co-purchased books, providing a compact, informative representation that can be used for further analysis or as input to other models.

– Để minh họa, xem xét nút biểu diễn cuốn sách "Losing Bin Laden" trong tập dữ liệu Sách Chính trị của chúng tôi. Sử dụng lệnh `model.wv['Losing Bin Laden']`, chúng tôi lấy được những vector dày đặc của nó. Vector này, được hiển thị trong Hình 2.3: **Trích xuất nhúng cho các nút liên kết với cuốn sách chính trị "Losing Bin Laden".** Đầu ra là 1 vector dày đặc được biểu diễn dưới dạng danh sách Python., nắm bắt các khía cạnh khác nhau về vai trò của cuốn sách trong mạng lưới các cuốn sách được mua chung, cung cấp 1 biểu diễn cô đọng, giàu thông tin, có thể được sử dụng để phân tích sâu hơn hoặc làm đầu vào cho các mô hình khác.

These embeddings can be used for exploratory data analysis to see patterns & relationships in a graph. However, their usage extends further. 1 common application is to use these vectors as features in a ML problem that uses tabular data. In that case, each element in our embedding array will become a distinct feature column in tabular data. This can add a rich representation of each node to complement other attributes in model training. In next sect, look at how to visualize these embeddings to gain deeper insights into patterns & relationships they represent.

– Các nhúng này có thể được sử dụng để phân tích dữ liệu thăm dò nhằm xem xét các mẫu & mối quan hệ trong biểu đồ. Tuy nhiên, ứng dụng của chúng còn mở rộng hơn nữa. 1 ứng dụng phổ biến là sử dụng các vectơ này làm đặc trưng trong bài toán học máy sử dụng dữ liệu bảng. Trong trường hợp đó, mỗi phần tử trong mảng nhúng của chúng ta sẽ trở thành 1 cột đặc trưng riêng biệt trong dữ liệu bảng. Điều này có thể bổ sung thêm 1 biểu diễn phong phú cho mỗi nút để bổ sung cho các thuộc tính khác trong quá trình huấn luyện mô hình. Trong phần tiếp theo, xem cách trực quan hóa các nhúng này để hiểu sâu hơn về các mẫu & mối quan hệ mà chúng biểu diễn.

- * 2.1.3. **Transforming & visualizing embeddings.** Visualization methods e.g. Uniform Manifold Approximation & Projection (UMAP) are powerful tools for reducing high-dimensional datasets into lower-dimensional space [5]. UMAP is particularly effective for identifying inherent clusters & visualizing complex structures that are difficult to perceive in high-dimensional data. Compared to other methods, e.g. t-SNE, UMAP excels in preserving both local & global structures, making it ideal for revealing patterns & relationships across different scales in data.

– **Biến đổi & trực quan hóa nhúng.** Các phương pháp trực quan hóa, ví dụ như Xấp xỉ Đa tạp Đồng nhất & Chiếu (UMAP) là những công cụ mạnh mẽ để thu gọn các tập dữ liệu nhiều chiều thành không gian ít chiều hơn [5]. UMAP đặc biệt hiệu quả trong việc xác định các cụm cố hữu & trực quan hóa các cấu trúc phức tạp khó nhận biết trong dữ liệu nhiều chiều. So với các phương pháp khác, ví dụ như t-SNE, UMAP vượt trội trong việc bảo toàn cả cấu trúc cục bộ & toàn cục, khiến nó trở nên lý tưởng để khám phá các mẫu & mối quan hệ trên các quy mô khác nhau trong dữ liệu.

While N2V generates embeddings by capturing network structure of our data, UMAP takes these high-dimensional embeddings & maps them onto a lower-dimensional space (typically 2D or 3D). This mapping aims to keep similar nodes close together while also preserving broader structural relationships, providing a more comprehensive visualization of graph's topology. After obtaining our N2V embeddings & converting them into a numerical array, we initialize UMAP

model with 2 components to project our data onto a 2D plane. By carefully selecting parameters e.g. number of neighbors & minimum distance. UMAP can balance between revealing fine-grained local relationships & maintaining global distances between clusters.

– Trong khi N2V tạo ra các nhúng bằng cách nắm bắt cấu trúc mạng của dữ liệu, UMAP sử dụng các nhúng đa chiều này & ánh xạ chúng vào không gian đa chiều thấp hơn (thường là 2D hoặc 3D). Việc ánh xạ này nhằm mục đích giữ các nút tương tự gần nhau đồng thời bảo toàn các mối quan hệ cấu trúc rộng hơn, cung cấp hình ảnh trực quan toàn diện hơn về cấu trúc đồ thị. Sau khi thu thập các nhúng N2V & chuyển đổi chúng thành 1 mảng số, chúng tôi khởi tạo mô hình UMAP với 2 thành phần để chiếu dữ liệu lên mặt phẳng 2D. Bằng cách lựa chọn cẩn thận các tham số, ví dụ: số lượng lân cận & khoảng cách tối thiểu, UMAP có thể cân bằng giữa việc thể hiện các mối quan hệ cục bộ chi tiết & duy trì khoảng cách toàn cục giữa các cụm.

By using UMAP, gain a more accurate & interpretable visualization of our graph embeddings as shown in Listing 2.2. Visualizing embeddings using UMAP: 1. Transforms embeddings into a list of vectors for UMAP. 2. Initializes & fits UMAP. 3. Plots nodes with UMAP embeddings & color by their value. allowing us to explore & analyze patterns, clusters, & structures more effectively than with traditional methods e.g. t-SNE.

– Bằng cách sử dụng UMAP, bạn có thể có được hình ảnh trực quan chính xác hơn & dễ hiểu hơn về các nhúng đồ thị của mình như được hiển thị trong Liệt kê 2.2. Hình ảnh trực quan các nhúng bằng UMAP: 1. Chuyển đổi các nhúng thành danh sách các vectơ cho UMAP. 2. Khởi tạo & phù hợp với UMAP. 3. Vẽ các nút bằng các nhúng UMAP & tô màu theo giá trị của chúng. cho phép chúng ta khám phá & phân tích các mẫu, cụm, & cấu trúc hiệu quả hơn so với các phương pháp truyền thống, ví dụ: t-SNE.

Resultant Fig. 2.4: Embeddings of Political Books dataset graph generated by N2V & visualized using UMAP. Shape & shading variations distinguish 3 political classes. encapsulates political book graph's embeddings as distilled by N2V & subsequently visualized through UMAP. Nodes appear in different shades according to their political alignment. Visualization unfolds a discernible structure, with potential clusters that correspond to various political leanings.

– Kết quả Hình 2.4: Đồ thị nhúng của tập dữ liệu Sách Chính trị được tạo bởi N2V & trực quan hóa bằng UMAP. Hình dạng & các biến thể tô bóng phân biệt 3 lớp chính trị. đóng gói các nhúng của đồ thị sách chính trị được tinh lọc bởi N2V & sau đó được trực quan hóa bằng UMAP. Các nút xuất hiện với các sắc thái khác nhau tùy theo sự liên kết chính trị của chúng. Trực quan hóa mở ra 1 cấu trúc rõ ràng, với các cụm tiềm năng tương ứng với các bài học chính trị khác nhau.

Might wonder why don't just reduce dimensions of N2V embeddings from 64 to 2 & visualize them directly, by passing UMAP altogether? In Listing 2.3: Visualizing 2D N2V embeddings without t-SNE, show this approach, applying a 2D N2V transformation directly to our `book_graph` object.

– Có thể bạn đang thắc mắc tại sao không giảm kích thước của các nhúng N2V từ 64 xuống còn 2 & trực quan hóa chúng trực tiếp bằng cách truyền hoàn toàn UMAP? Trong Liệt kê 2.3: Trực quan hóa các nhúng N2V 2D mà không cần t-SNE, trình bày cách tiếp cận này, áp dụng phép biến đổi N2V 2D trực tiếp vào đối tượng `book_graph` của chúng ta.

`dimensions` parameter is set to 2, aiming for a direct 2D representation suitable for immediate visualization without further dimensionality reduction. Other parameters are kept the same.

– Tham số `dimensions` được đặt thành 2, hướng đến biểu diễn 2D trực tiếp, phù hợp để trực quan hóa ngay lập tức mà không cần giảm kích thước thêm. Các tham số khác được giữ nguyên.

Once model is fitted with specified window & word batch settings, extract 2D embeddings & store them in a dictionary keyed by string representation of each node. This enables a direct mapping from node to its embedding vector.

– Sau khi mô hình được điều chỉnh với các thiết lập hàng loạt của số & từ được chỉ định, trích xuất các nhúng 2D & lưu trữ chúng trong 1 từ điển được khóa bằng chuỗi biểu diễn của mỗi nút. Điều này cho phép ánh xạ trực tiếp từ nút đến vectơ nhúng của nó.

Extracted 2D points are compiled into a NumPy array & plotted. Use standard Matplotlib library to create a scatterplot of these points using prepared color scheme to represent political leaning of each node visually.

– Các điểm 2D được trích xuất sẽ được biên dịch thành 1 mảng NumPy & vẽ đồ thị. Sử dụng thư viện Matplotlib chuẩn để tạo biểu đồ phân tán các điểm này bằng cách sử dụng bảng màu đã chuẩn bị sẵn để thể hiện trực quan khuynh hướng chính trị của từng nút.

Outcome shows how books are separated by political leanings, similar to UMAP result, but where books are more bunched together Fig. 2.5: Embeddings of Political Books dataset graph generated & visualized by N2V for 2D. Shape & shading variations distinguish 3 political classes. Here, see a similar clustering by political leaning as earlier in Fig. 2.5 but more bunched together. 2 embeddings are then shown in Fig. 2.6: Comparison of embeddings generated by N2V & t-SNE & a direct visualization of 2D Node2Vec.

– Kết quả cho thấy sách được phân loại theo khuynh hướng chính trị, tương tự như kết quả UMAP, nhưng sách được gom lại với nhau nhiều hơn Hình 2.5: Đồ thị nhúng của tập dữ liệu Sách Chính trị được tạo & trực quan hóa bằng N2V cho 2D. Các biến thể hình & tô bóng phân biệt 3 lớp chính trị. Ở đây, xem 1 phân cụm tương tự theo khuynh hướng chính trị như trước đó trong Hình 2.5 nhưng được gom lại với nhau nhiều hơn. 2 nhúng sau đó được hiển thị trong Hình 2.6: So sánh các nhúng được tạo bởi N2V & t-SNE & trực quan hóa trực tiếp của Node2Vec 2D.

Clear: both methods know to separate books into groups based on political leanings. N2V is less expressive in how it separates books, bunching them together across 2D. Meanwhile, UMAP is better for spreading out books in 2D. Relevant benefit or information contained within these dimensions depends on task at hand.

– Rõ ràng: cả hai phương pháp đều biết cách phân loại sách thành các nhóm dựa trên khuynh hướng chính trị. N2V kém biểu cảm hơn trong cách phân loại sách, gom chúng lại với nhau trên không gian 2 chiều. Trong khi đó, UMAP tốt hơn trong việc phân bổ sách trên không gian 2 chiều. Lợi ích hoặc thông tin liên quan chứa trong các chiều này phụ thuộc

vào nhiệm vụ được giao.

- * 2.1.4. Beyond visualization: Applications & considerations of N2V embeddings. While visualizing N2V embeddings offers intuitive insights into dataset's structure, their usage extends far beyond graphical representation. N2V is an embedding method designed specifically for graphs; it captures both local & global structural properties of nodes by simulating random walks through graph. This process allows N2V to create dense, numerical vectors that summarize position & context of each node within overall network.

– Vượt ra ngoài trực quan hóa: Ứng dụng & cân nhắc về nhúng N2V. Mặc dù trực quan hóa nhúng N2V mang lại cái nhìn sâu sắc trực quan về cấu trúc của tập dữ liệu, nhưng ứng dụng của chúng còn vượt xa việc biểu diễn đồ họa. N2V là 1 phương pháp nhúng được thiết kế riêng cho đồ thị; nó nắm bắt cả các thuộc tính cấu trúc cục bộ & toàn cục của các nút bằng cách mô phỏng các bước ngẫu nhiên qua đồ thị. Quá trình này cho phép N2V tạo ra các vectơ số dày đặc, tóm tắt vị trí & ngữ cảnh của mỗi nút trong toàn bộ mạng.

These embeddings can then serve as feature-rich inputs for a variety of ML tasks, e.g. classification, recommendation, or clustering. E.g. in our Political Books dataset, embeddings could help predict a book's political leaning based on its co-purchase patterns or could recommend books to users with similar political interests. They might even be used to forecast future sales based on content of a book.

– Những nhúng này sau đó có thể đóng vai trò là dữ liệu đầu vào giàu tính năng cho nhiều tác vụ ML, ví dụ như phân loại, đề xuất hoặc phân cụm. Ví dụ: trong tập dữ liệu Sách Chính trị của chúng tôi, các nhúng có thể giúp dự đoán khả năng học hỏi chính trị của 1 cuốn sách dựa trên các mô hình mua chung hoặc có thể đề xuất sách cho người dùng có cùng sở thích chính trị. Chúng thậm chí có thể được sử dụng để dự báo doanh số bán hàng trong tương lai dựa trên nội dung của 1 cuốn sách.

However, important to understand nature of N2V's learning approach, which is *transductive*. Transductive learning is designed to work only with specific dataset it was trained on & cannot generalize to new, unseen nodes without retraining model. This characteristic makes N2V highly effective for static datasets where all nodes & edges are known up front but less suitable for dynamic settings where new data points or connections frequently appear. Essentially, N2V focuses on extracting detailed patterns & relationships from existing graph rather than developing a model that can easily adapt to new data.

– Tuy nhiên, điều quan trọng là phải hiểu bản chất của phương pháp học tập của N2V, đó là *transductive*. Học tập transductive được thiết kế để chỉ hoạt động với tập dữ liệu cụ thể mà nó được huấn luyện & không thể khái quát hóa sang các nút mới, chưa được biết đến mà không cần huấn luyện lại mô hình. Đặc điểm này khiến N2V rất hiệu quả đối với các tập dữ liệu tĩnh

trong đó tất cả các nút & cạnh đều được biết trước nhưng ít phù hợp hơn với các thiết lập động, nơi các điểm dữ liệu hoặc kết nối mới thường xuyên xuất hiện. Về cơ bản, N2V tập trung vào việc trích xuất các mẫu & mối quan hệ chi tiết từ đồ thị hiện có thay vì phát triển 1 mô hình có thể dễ dàng thích ứng với dữ liệu mới.

While this transductive nature has its limitations, it also offer significant advantages. Because n2V uses full structure of graph during training, it can capture intricate relationships & dependencies that might be missed by more generalized methods. This makes N2V particularly powerful for tasks where complete, fixed structure of data is known & stable. However, to apply N2V effectively, it is crucial to ensure that graph data is represented in a way that captures all relevant features. In some cases, additional edges or nodes may need to be added to graph to fully represent underlying relationships.

– Mặc dù bản chất chuyển đổi này có những hạn chế, nhưng nó cũng mang lại những lợi thế đáng kể. Vì n2V sử dụng toàn bộ cấu trúc đồ thị trong quá trình huấn luyện, nó có thể nắm bắt được các mối quan hệ phức tạp & các mối phụ thuộc mà các phương pháp tổng quát hơn có thể bỏ sót. Điều này làm cho N2V đặc biệt mạnh mẽ đối với các tác vụ mà cấu trúc dữ liệu hoàn chỉnh, cố định đã biết & ổn định. Tuy nhiên, để áp dụng N2V hiệu quả, điều quan trọng là phải đảm bảo dữ liệu đồ thị được biểu diễn theo cách nắm bắt được tất cả các đặc điểm liên quan. Trong 1 số trường hợp, có thể cần thêm các cạnh hoặc nút bổ sung vào đồ thị để biểu diễn đầy đủ các mối quan hệ cơ bản.

For those interested in a deeper understanding of transductive models & how N2V's approach compares to other methods, further details are provided in Sect. 2.4.2, which explore tradeoffs between transductive & inductive learning in greater depth [6, 7], helping you understand when each approach is most appropriate.

– Đối với những người quan tâm đến việc hiểu sâu hơn về các mô hình chuyển đổi & cách tiếp cận của N2V so sánh với các phương pháp khác, các chi tiết bổ sung được cung cấp trong Phần 2.4.2, khám phá sự đánh đổi giữa học chuyển đổi & quy nạp sâu hơn [6, 7], giúp bạn hiểu khi nào thì mỗi cách tiếp cận là phù hợp nhất.

While N2V is effective for generating embeddings that capture structure of a fixed graph, real-world data often demands a more flexible & generalizable approach. This need brings us to our 1st GNN architecture for creating node embeddings. Unlike N2V, which is a transductive method limited to specific nodes & edges in training data, GNNs can learn in an *inductive* manner, i.e., GNNs are capable of generalizing to new, unseen nodes or edges without requiring retraining on entire graph.

– Mặc dù N2V hiệu quả trong việc tạo ra các phép nhúng nắm bắt cấu trúc của 1 đồ thị cố định, dữ liệu thực tế thường đòi hỏi 1 phương pháp linh hoạt hơn & có thể khái quát hóa. Nhu cầu này đưa chúng ta đến kiến trúc GNN đầu tiên để tạo ra các phép nhúng nút. Không giống như N2V, vốn là phương pháp chuyển đổi giới hạn ở các nút & cạnh cụ thể trong dữ liệu huấn luyện, GNN có thể học theo cách *quy nạp*, tức là GNN có khả năng khái quát hóa sang các nút hoặc cạnh mới, chưa được biết đến mà không cần phải huấn luyện lại toàn bộ đồ thị.

GNNs achieve this by not only understanding network's complex structure but also by incorporating node features & relationships into learning process. This approach allows GNNs to adapt dynamically to changes in graph, making them well-suited for applications where data is continually evolving. Shift from N2V to GNNs represents a key transition

from focusing on deep analysis within a static dataset to a broader applicability across diverse, evolving networks. This adaptability sets stage for a wider range of graph-based ML applications that require flexibility & scalability. In next sect, explore how GNNs go beyond capabilities of N2V & other transductive methods, allowing for more versatile & powerful models that can handle dynamic nature of real-world data.

– GNN đạt được điều này không chỉ bằng cách hiểu cấu trúc phức tạp của mạng mà còn bằng cách kết hợp các đặc điểm nút & mối quan hệ vào quá trình học. Cách tiếp cận này cho phép GNN thích ứng linh hoạt với những thay đổi trong đồ thị, khiến chúng phù hợp với các ứng dụng mà dữ liệu liên tục phát triển. Sự chuyển đổi từ N2V sang GNN thể hiện 1 bước chuyển đổi quan trọng từ việc tập trung vào phân tích sâu trong 1 tập dữ liệu tĩnh sang khả năng ứng dụng rộng rãi hơn trên các mạng lưới đa dạng và đang phát triển. Khả năng thích ứng này đặt nền tảng cho 1 loạt các ứng dụng học máy dựa trên đồ thị đòi hỏi tính linh hoạt & khả năng mở rộng. Trong phần tiếp theo, khám phá cách GNN vượt ra ngoài khả năng của N2V & các phương pháp chuyển đổi khác, cho phép tạo ra các mô hình linh hoạt & mạnh mẽ hơn có thể xử lý bản chất động của dữ liệu thực tế.

- o 2.2. Creating embeddings with a GNN. While N2V provides a powerful method for generating embeddings by capturing local & global structure of a graph, it is fundamentally a transductive approach, i.e., it cannot easily generalize to unseen nodes or edges without retraining. Although there have been extensions to N2V that enable it to work in inductive settings, GNNs are inherently designed for inductive learning. I.e., they can learn general patterns from graph data that allow them to make predictions or to generate embeddings for new nodes or edges without need to retrain entire model. This gives GNNs a significant edge in scenarios where flexibility & adaptability are crucial.

– Tạo nhúng với GNN. Mặc dù N2V cung cấp 1 phương pháp mạnh mẽ để tạo nhúng bằng cách nắm bắt cấu trúc cục bộ & toàn cục của đồ thị, nhưng về cơ bản, nó là 1 phương pháp chuyển đổi, tức là nó không thể dễ dàng khái quát hóa sang các nút hoặc cạnh chưa biết mà không cần đào tạo lại. Mặc dù đã có các phần mở rộng cho N2V cho phép nó hoạt động trong các thiết lập quy nạp, GNN vốn được thiết kế cho việc học quy nạp. Tức là, chúng có thể học các mẫu chung từ dữ liệu đồ thị, cho phép chúng đưa ra dự đoán hoặc tạo nhúng cho các nút hoặc cạnh mới mà không cần đào tạo lại toàn bộ mô hình. Điều này mang lại cho GNN 1 lợi thế đáng kể trong các tình huống mà tính linh hoạt & khả năng thích ứng là rất quan trọng.

GNNs not only incorporate structural information of graph, like N2V, but they also use node features to create richer representations. This dual capacity allows GNNs to learn both complex relationships within graph & specific characteristics of individual nodes, enabling them to excel in tasks where both types of information are important.

– GNN không chỉ kết hợp thông tin cấu trúc của đồ thị, như N2V, mà còn sử dụng các đặc điểm của nút để tạo ra các biểu diễn phong phú hơn. Khả năng kép này cho phép GNN học cả các mối quan hệ phức tạp trong đồ thị & các đặc điểm cụ thể của từng nút, cho phép chúng vượt trội trong các tác vụ mà cả hai loại thông tin đều quan trọng.

That said, while GNNs have demonstrated impressive performance across many applications, they do not universally outperform methods e.g. N2V in all cases. E.g., N2V & other random walk-based methods can sometimes perform better in scenarios where labeled data is scarce or noisy, thanks to their ability to work with just graph structure without needing additional node features.

– Tuy nhiên, mặc dù GNN đã chứng minh hiệu suất ấn tượng trong nhiều ứng dụng, chúng không phải lúc nào cũng vượt trội hơn các phương pháp như N2V trong mọi trường hợp. E.g., N2V & các phương pháp dựa trên bước đi ngẫu nhiên khác đôi khi có thể hoạt động tốt hơn trong các tình huống dữ liệu được gắn nhãn khan hiếm hoặc nhiễu, nhờ khả năng hoạt động chỉ với cấu trúc đồ thị mà không cần thêm các đặc trưng nút.

- * 2.2.1. Constructing embeddings. Unlike N2V, GNNs learn graph representations & perform tasks e.g. node classification or link prediction simultaneously during training. Information from entire is processed through successive GNN layers, each refining node embeddings without requiring a separate step for their creation.

– Xây dựng nhúng. Không giống như N2V, GNN học các biểu diễn đồ thị & thực hiện đồng thời các tác vụ như phân loại nút hoặc dự đoán liên kết trong quá trình huấn luyện. Thông tin từ toàn bộ được xử lý qua các lớp GNN kế tiếp, mỗi lớp tinh chỉnh các nhúng nút mà không cần 1 bước riêng biệt để tạo chúng.

To demonstrate how a GNN extracts features from graph data, perform a straightforward pass-through using an untrained model to generate preliminary embeddings. Even without optimization typically involved in training, this approach will show how GNNs use message passing to update embeddings, capturing both graph's structure & its node features. When optimization is added, these embeddings become tailored to specific tasks e.g. node classification or link prediction.

– Để minh họa cách GNN trích xuất các đặc điểm từ dữ liệu đồ thị, thực hiện 1 phép truyền trực tiếp đơn giản bằng 1 mô hình chưa được huấn luyện để tạo ra các nhúng sơ bộ. Ngay cả khi không có quá trình tối ưu hóa thường được sử dụng trong huấn luyện, phương pháp này vẫn sẽ cho thấy cách GNN sử dụng việc truyền thông điệp để cập nhật các nhúng, nắm bắt cả cấu trúc của đồ thị và các đặc điểm nút của nó. Khi được tối ưu hóa, các nhúng này sẽ được điều chỉnh cho phù hợp với các tác vụ cụ thể, ví dụ như phân loại nút hoặc dự đoán liên kết.

- Defining our GNN architecture. Initiate our process by defining a simple GCN architecture, as shown in Listing 2.4: SimpleGNN class. Our SimpleGNN class inherits from `torch.nn.Module` & is composed of 2 GCN-Conv layers, which are building blocks of our GNN. This architecture is shown in Fig. 2.7: Architecture diagram of SimpleGNN model, consisting of 1st layer, a message passing layer `self.conv1`, an activation `torch.relu`, a dropout layer `torch.dropout`, & a 2nd message passing layer.

– Định nghĩa kiến trúc GNN của chúng ta. Bắt đầu quy trình bằng cách định nghĩa 1 kiến trúc GCN đơn, như được hiển thị trong Liệt kê 2.4: SimpleGNN class. Lớp SimpleGNN của chúng ta kế thừa từ `torch.nn.Module` & bao gồm 2 lớp GCN-Conv, là các khối xây dựng nên GNN của chúng ta. Kiến trúc này được hiển thị trong Hình 2.7: Sơ đồ kiến trúc của

mô hình SimpleGNN, bao gồm lớp thứ nhất, lớp truyền tin nhấn `self.conv1`, lớp kích hoạt `torch.relu`, lớp dropout `torch.dropout`, & lớp truyền tin nhấn thứ hai.

Talk about architecture aspects specific to GNNs. Activation & dropout are common in many DL scenarios. GNN layers, however, are different from conventional DL layers in a fundamental way. Core principle that allows GNNs to learn from graph data is message passing. For each GNN layer, in addition to updating layer's weights, a "message" is gathered from every node or edge neighborhood & used to update an embedding. Essentially, each node sends messages to its neighbors & simultaneously receives messages from them. For every node, its new embedding is computed by combining its own features with aggregated messages from its neighbors, through a combination of nonlinear transformations.

– Nói về các khía cạnh kiến trúc đặc thù của GNN. Activation & dropout là phổ biến trong nhiều kịch bản DL. Tuy nhiên, các lớp GNN khác biệt cơ bản so với các lớp DL thông thường. Nguyên lý cốt lõi cho phép GNN học từ dữ liệu đồ thị là truyền thông điệp. Đối với mỗi lớp GNN, ngoài việc cập nhật trọng số của lớp, 1 "thông điệp" được thu thập từ mọi nút hoặc lân cận cạnh & được sử dụng để cập nhật nhúng. Về cơ bản, mỗi nút gửi thông điệp đến các nút lân cận & đồng thời nhận thông điệp từ chúng. Đối với mỗi nút, nhúng mới của nó được tính toán bằng cách kết hợp các đặc trưng của chính nó với các thông điệp tổng hợp từ các nút lân cận, thông qua sự kết hợp của các phép biến đổi phi tuyến tính.

In this example, we are going to be using a graph convolutional network (GCN) to act as our message-passing GNN layers. Describe GCNs in much more detail i Chap. 3. For now, just need to know that GCNs act as message-passing layers that are critical in constructing embeddings.

– Trong ví dụ này, chúng ta sẽ sử dụng mạng tích chập đồ thị (GCN) để hoạt động như các lớp GNN truyền thông điệp. Mô tả chi tiết hơn về GCN trong Chương 3. Hiện tại, chúng ta chỉ cần biết rằng GCN hoạt động như các lớp truyền thông điệp, đóng vai trò quan trọng trong việc xây dựng các hàm nhúng.

• **Data preparation.** Next, prepare our data. Start with same graph from previous section, `books_gml`, in its `NetworkX` form. Have to convert this `NetworkX` object into a tensor form that is suitable to use with PyTorch operations. Because PyTorch Geometric (PyG) has many functions that convert graph objects, we can do this quite simply with `data = from_NetworkX(gml_graph)`. Method `from_NetworkX` specifically translates edge lists & `node.edge` attributes into PyTorch tensors.

– Chuẩn bị dữ liệu. Tiếp theo, chuẩn bị dữ liệu. Bắt đầu với cùng 1 đồ thị từ phần trước, `books_gml`, ở dạng `NetworkX`. Phải chuyển đổi đối tượng `NetworkX` này sang dạng tensor phù hợp để sử dụng với các phép toán PyTorch. Vì PyTorch Geometric (PyG) có nhiều hàm chuyển đổi đối tượng đồ thị, chúng ta có thể thực hiện việc này khá đơn giản với `data = from_NetworkX(gml_graph)`. Phương thức `from_NetworkX` đặc biệt chuyển đổi các thuộc tính danh sách cạnh & `node.edge` thành tensor PyTorch.

For GNNs, generating node embeddings requires initializing node features. In our case, we do not have any predefined node features. When no node features are available or they are not informative, it is common practice to initialize node features randomly. A more effective approach: use *Xavier initialization*, which sets initial node features with values drawn from a distribution that keeps variety of activations consistent across layers. This technique ensures: model starts with a balanced representation, preventing problems e.g. vanishing or exploding gradients.

– Đối với GNN, việc tạo nhúng nút đòi hỏi phải khởi tạo các đặc trưng nút. Trong trường hợp của chúng tôi, chúng tôi không có bất kỳ đặc trưng nút nào được xác định trước. Khi không có đặc trưng nút nào khả dụng hoặc chúng không cung cấp thông tin, việc khởi tạo các đặc trưng nút 1 cách ngẫu nhiên là 1 phương pháp phổ biến. 1 cách tiếp cận hiệu quả hơn: sử dụng *Xavier initialization*, phương pháp này đặt các đặc trưng nút ban đầu với các giá trị được lấy từ 1 phân phối duy trì tính nhất quán của các phép kích hoạt trên các lớp. Kỹ thuật này đảm bảo: mô hình bắt đầu với 1 biểu diễn cân bằng, ngăn ngừa các vấn đề như gradient biến mất hoặc bùng nổ.

By initializing `data.x` with Xavier initialization, provide GNN with a starting point that allows it to learn meaningful node embeddings from noninformative features. During training, network adjusts these initial values to minimize loss function. When loss function is aligned with a specific target, e.g. node prediction, embeddings learned from initial random features will become tailored to task at hand, resulting in more effective representations. Randomize node features using following:

– Khởi tạo, cung cấp cho GNN 1 điểm khởi đầu cho phép nó học các phép nhúng nút có ý nghĩa từ các đặc trưng không mang tính thông tin. Trong quá trình huấn luyện, mạng sẽ điều chỉnh các giá trị ban đầu này để giảm thiểu hàm mất mát. Khi hàm mất mát được căn chỉnh với 1 mục tiêu cụ thể, ví dụ: dự đoán nút, các phép nhúng học được từ các đặc trưng ngẫu nhiên ban đầu sẽ được điều chỉnh cho phù hợp với tác vụ hiện tại, mang lại hiệu quả biểu diễn cao hơn. Ngẫu nhiên hóa các đặc trưng nút bằng cách sử dụng các bước sau:

```
1 data.x = torch.randn((data.num_nodes, 64), dtype = torch.float)
2 'nn.init.xavier_uniform_(data.x)'
```

We could have also used embeddings from N2V exercise to use as node features. Recall `node_embeddings` object from Sect. 2.1.3:

– Chúng ta cũng có thể sử dụng các nhúng từ bài tập N2V để làm đặc trưng nút. Hãy nhớ lại đối tượng `node_embeddings` từ Mục 2.1.3:

```
node_embeddings = [embeddings[str(node)] for node in gml_graph.nodes()]
```

From this, can convert node embedding to a PyTorch tensor object & assign it to node feature object, `data.x`:

– Từ đó, có thể chuyển đổi nhúng nút thành đối tượng tensor PyTorch & gán nó cho đối tượng tính năng nút, `data.x`:

```
node_features = torch.tensor(node_embedding, dtype = torch.float)
```

```
data.x = node_features
```

• Passing graph through GNN. With structure of our GNN model defined & our graph data formatted for PyG, proceed to embedding generation step. Initialize our model, `SimpleGNN`, specifying number of features for each node & size of hidden channels within network.

– Truyền đồ thị qua GNN. Với cấu trúc mô hình GNN đã được xác định & dữ liệu đồ thị được định dạng cho PyG, tiến hành bước tạo nhúng. Khởi tạo mô hình `SimpleGNN`, chỉ định số lượng đặc trưng cho mỗi nút & kích thước của các kênh ẩn trong mạng.

```
model = SimpleGNN(num_features = data.x.shape[1], hidden_channels = 64)
```

Here, specify 64 hidden channels because we want to compare resulting embeddings to the ones we produced using `node2vec` method, which had 64 dimensions. Because 2nd GNN layer is last layer, output will be a 64-element vector.

– Ở đây, ta chỉ định 64 kênh ẩn vì chúng ta muốn so sánh các nhúng kết quả với các nhúng chúng ta tạo ra bằng phương pháp `node2vec`, có 64 chiều. Vì lớp GNN thứ 2 là lớp cuối cùng, đầu ra sẽ là 1 vectơ 64 phần tử.

Once initialized, we switch model to evaluation mode using `model.eval()`. This mode is used during inference or validation phases when we want to make predictions or assess model performance without modifying model's parameters. Specifically, `model.eval()` turns off certain behaviors specific to training, e.g. *dropout*, which randomly deactivates some neurons to prevent overfitting, & *batch normalization*, which normalizes inputs across a mini-batch. By disabling these features, model provides consistent & deterministic outputs, ensuring: evaluation accurately reflects its true performance on unseen data.

– Sau khi khởi tạo, chúng tôi chuyển mô hình sang chế độ đánh giá bằng `model.eval()`. Chế độ này được sử dụng trong các giai đoạn suy luận hoặc xác thực khi chúng tôi muốn đưa ra dự đoán hoặc đánh giá hiệu suất mô hình mà không cần sửa đổi các tham số của mô hình. Cụ thể, `model.eval()` sẽ tắt 1 số hành vi cụ thể trong quá trình huấn luyện, ví dụ: *dropout*, vô hiệu hóa ngẫu nhiên 1 số nơ-ron để ngăn ngừa quá khớp, & *batch normalization*, chuẩn hóa đầu vào trên 1 mini-batch. Bằng cách vô hiệu hóa các tính năng này, mô hình cung cấp đầu ra nhất quán & xác định, đảm bảo: đánh giá phản ánh chính xác hiệu suất thực tế của nó trên dữ liệu chưa được biết đến.

Important to disable gradient computations because they are not necessary for forward pass & embedding generation. So, we employ `torch.no_grad()`, which ensures: computational graph that records operations for backpropagation is not constructed, preventing us from accidentally changing performance.

– Điều quan trọng là phải tắt tính toán gradient vì chúng không cần thiết cho việc tạo nhúng & truyền tiếp. Vì vậy, chúng tôi sử dụng `torch.no_grad()`, đảm bảo: đồ thị tính toán ghi lại các thao tác cho lan truyền ngược không được xây dựng, ngăn chúng tôi vô tình thay đổi hiệu suất.

Next, pass our node-feature matrix `data.x` & edge index `data.edge_index` through model. Result is `gnn_embeddings`, a tensor where each row corresponds to embedding of a node in our graph – a numerical representation learned by our GNN, ready for downstream tasks e.g. visualization or classification:

– Tiếp theo, truyền ma trận đặc trưng nút `data.x` & chỉ số cạnh `data.edge_index` qua mô hình. Kết quả là `gnn_embeddings`, 1 tenxơ trong đó mỗi hàng tương ứng với việc nhúng 1 nút vào đồ thị của chúng ta – 1 biểu diễn số được học bởi GNN, sẵn sàng cho các tác vụ tiếp theo, ví dụ như trực quan hóa hoặc phân loại:

```
model.eval()
```

```
with torch.no_grad():
```

```
    gnn_embeddings = model(data.x, data.edge_index)
```

After producing these embeddings, use UMAP to visualize them. Since we have been working with PyTorch tensor data types running on a GPU, need to convert our embeddings to a NumPy array data type to use analysis methods outside of PyTorch, which are done on a CPU:

– Sau khi tạo các nhúng này, sử dụng UMAP để trực quan hóa chúng. Vì chúng ta đang làm việc với các kiểu dữ liệu tensor PyTorch chạy trên GPU, cần chuyển đổi các nhúng sang kiểu dữ liệu mảng NumPy để sử dụng các phương pháp phân tích bên ngoài PyTorch, vốn được thực hiện trên CPU:

```
gnn_embeddings_np = gnn_embeddings.detach().cpu().numpy()
```

With this convention, we can produce UMAP calculations & visualization following process we used in N2V case. Resulting scatterplot (Fig. 2.8: Visualization of embeddings generated from passing a graph through a GNN.) is a 1st glimpse at clusters within our graph. Add different shadings based on each node's label (left-, right-, or neural-leaning) to see that similar leaning books are fairly well grouped, given that these embeddings were constructed from topology alone.

– Với quy ước này, chúng ta có thể tạo ra các phép tính UMAP & trực quan hóa theo quy trình đã sử dụng trong trường hợp N2V. Biểu đồ phân tán kết quả (Hình 2.8: Trực quan hóa các nhúng được tạo ra khi truyền đồ thị qua mạng GNN.) là cái nhìn đầu tiên về các cụm trong đồ thị của chúng ta. Hãy thêm các hiệu ứng tô bóng khác nhau dựa trên nhãn của mỗi nút (ngiên trái, phải hoặc nghiêng nơ-ron) để thấy rằng các sách nghiêng tương tự được nhóm khá tốt, vì các nhúng này chỉ được xây dựng từ topo.

Next, discuss both how GNN embeddings are used & how they differ from those produced with N2V.

– Tiếp theo, thảo luận về cách sử dụng nhúng GNN & cách chúng khác với nhúng được tạo bằng N2V như thế nào.

* 2.2.2. GNN vs. N2V embeddings. Through this book, predominantly use GNNs to generate embeddings because this embedding process is intrinsic to a GNN's architecture. While embeddings play a pivotal role in methodologies & applications

we explore in rest of book, their presence is often subtle & not always highlighted. This approach allows us to focus on broader concepts & applications of GNN-based ML without getting slowed down by technicalities. Nonetheless, important to acknowledge: underlying power & adaptability of embeddings are central to advanced techniques & insights we get into throughout text.

– **GNN so với những N2V.** Trong cuốn sách này, chúng tôi chủ yếu sử dụng GNN để tạo những vì quá trình những này là 1 phần không thể thiếu trong kiến trúc của GNN. Mặc dù những đóng vai trò then chốt trong các phương pháp luận & ứng dụng mà chúng tôi sẽ đề cập trong phần còn lại của cuốn sách, nhưng sự hiện diện của chúng thường rất tinh tế & không phải lúc nào cũng được nhấn mạnh. Cách tiếp cận này cho phép chúng tôi tập trung vào các khái niệm rộng hơn & ứng dụng của ML dựa trên GNN mà không bị chậm lại bởi các vấn đề kỹ thuật. Tuy nhiên, điều quan trọng cần lưu ý là: sức mạnh tiềm ẩn & khả năng thích ứng của những là trọng tâm của các kỹ thuật tiên tiến & những hiểu biết sâu sắc mà chúng tôi sẽ tìm hiểu trong suốt cuốn sách.

GNN-produced node embeddings are particularly powerful because they enable us to tackle a broad range of graph-related tasks by using their inductive nature. Inductive learning allows these embeddings to generalize to new, unseen nodes or even entirely new graphs without needing to retrain model. In contrast, N2V embeddings are limited to specific graphs they were trained on & can't easily adapt to new data. Reiterate key ways in which GNN embeddings differ from other embedding methods, e.g. N2V [1, 3].

– Những nút do GNN tạo ra đặc biệt mạnh mẽ vì chúng cho phép chúng ta giải quyết 1 loạt các tác vụ liên quan đến đồ thị bằng cách sử dụng tính chất quy nạp của chúng. Học quy nạp cho phép các những này tổng quát hóa sang các nút mới, chưa từng thấy hoặc thậm chí là các đồ thị hoàn toàn mới mà không cần phải đào tạo lại mô hình. Ngược lại, những N2V bị giới hạn trong các đồ thị cụ thể mà chúng được đào tạo & không thể dễ dàng thích ứng với dữ liệu mới. Nhắc lại những điểm chính mà những GNN khác với các phương pháp những khác, ví dụ: N2V [1, 3].

- **Adaptability of new graphs.** 1 of critical features of GNN embeddings is their adaptability. Because GNNs learn a function that maps node features to embeddings, this function can be applied to nodes in new graphs without needing to be retrained, provided nodes have similar feature spaces. This inductive capability is particularly valuable in dynamic environments where graph may evolve over time or in applications where model needs to be applied to different but structurally similar graphs. N2V, on other hand, needs to be reapplied for each new graph or set of nodes.

– **Khả năng thích ứng của đồ thị mới.** 1 trong những đặc điểm quan trọng của những GNN là khả năng thích ứng. Vì GNN học 1 hàm ánh xạ các đặc trưng của nút với các những, hàm này có thể được áp dụng cho các nút trong đồ thị mới mà không cần phải đào tạo lại, miễn là các nút có không gian đặc trưng tương tự. Khả năng quy nạp này đặc biệt hữu ích trong các môi trường động, nơi đồ thị có thể phát triển theo thời gian hoặc trong các ứng dụng cần áp dụng mô hình cho các đồ thị khác nhau nhưng có cấu trúc tương tự. Mặt khác, N2V cần được áp dụng lại cho mỗi đồ thị hoặc tập hợp nút mới.

- **Enhanced feature integration.** GNNs inherently consider node features during embedding process, allowing for a complex & nuanced representation of each node. This integration of node features, alongside structural information, offers a more comprehensive view compared to N2V & other methods that focus on a graph's topology. This capability makes GNN embeddings particularly suited for tasks where node features contain significant additional information.

– **Tích hợp tính năng nâng cao.** GNN vốn đã xem xét các tính năng nút trong quá trình những, cho phép biểu diễn phức tạp & sắc thái của từng nút. Việc tích hợp các tính năng nút này, cùng với thông tin cấu trúc, mang lại cái nhìn toàn diện hơn so với các phương pháp N2V & khác tập trung vào cấu trúc của đồ thị. Khả năng này làm cho những GNN đặc biệt phù hợp cho các tác vụ mà các tính năng nút chứa thông tin bổ sung đáng kể.

- **Task-specific optimization.** GNNs embeddings are trained alongside specific tasks, e.g. node classification, link prediction, or even graph classification. Through end-to-end training, GNN model learns to optimize embeddings for task at hand, leading to potentially higher performance & efficiency compared to using pre-generated embeddings e.g. those from N2V.

– **Tối ưu hóa theo tác vụ.** Các những GNN được huấn luyện cùng với các tác vụ cụ thể, ví dụ: phân loại nút, dự đoán liên kết, hoặc thậm chí phân loại đồ thị. Thông qua quá trình huấn luyện đầu cuối, mô hình GNN học cách tối ưu hóa các những cho tác vụ đang thực hiện, dẫn đến hiệu suất & hiệu quả cao hơn so với việc sử dụng các những được tạo sẵn, ví dụ như từ N2V.

That said, while GNN embeddings offer clear advantages in terms of adaptability & applicability to new data, N2V embeddings have their strengths, particularly in capturing nuanced patterns within a specific graph's structure. In practice, choice between GNN & N2V embeddings may depend on specific requirements of task, nature of graph data, & constraints of computational environment.

– Tuy nhiên, trong khi những GNN mang lại những lợi thế rõ ràng về khả năng thích ứng & khả năng ứng dụng cho dữ liệu mới, những N2V cũng có những điểm mạnh riêng, đặc biệt là trong việc nắm bắt các mẫu hình phức tạp trong cấu trúc đồ thị cụ thể. Trên thực tế, việc lựa chọn giữa những GNN & N2V có thể phụ thuộc vào các yêu cầu cụ thể của tác vụ, bản chất của dữ liệu đồ thị, & các ràng buộc của môi trường tính toán.

For tasks where graph structure is static & well-defined, N2V might provide a simpler & computationally efficient solution. Conversely, for dynamic graphs, large-scale applications, or scenarios requiring incorporation of node features, GNNs will often be the more robust & versatile choice. Additionally, when task itself is not well-defined & work is exploratory, N2V is likely faster & easier to use.

– **Đối với các tác vụ có cấu trúc đồ thị tĩnh & được xác định rõ ràng,** N2V có thể cung cấp 1 giải pháp đơn giản hơn & hiệu quả về mặt tính toán. Ngược lại, **đối với các đồ thị động, ứng dụng quy mô lớn hoặc các tình huống yêu cầu tích hợp các đặc điểm nút,** GNN thường là lựa chọn mạnh mẽ & linh hoạt hơn. Ngoài ra, khi bản thân tác vụ không được

xác định rõ & công việc mang tính khám phá, N2V có thể nhanh hơn & dễ sử dụng hơn.

Have now successfully built our 1st GNN embedding. This is key 1st step for all GNN models, & everything from this point will build on it. In next sect, give an example of some of these next steps & show how to use embeddings to solve a ML problem.

– Chúng ta đã xây dựng thành công mô hình nhúng GNN đầu tiên. Đây là bước đầu tiên quan trọng cho tất cả các mô hình GNN, & mọi thứ từ đây sẽ được xây dựng dựa trên nó. Trong phần tiếp theo, đưa ra ví dụ về 1 số bước tiếp theo & trình bày cách sử dụng nhúng để giải quyết bài toán học máy.

- 2.3. Using Node Embeddings. Semi-supervised learning, which involves a combination of labeled & unlabeled data, provides a valuable opportunity to compare different embedding techniques. In this chap, explore how GNN & N2V embeddings can be used to predict labels when majority of data lacks labels.

– Sử dụng Nhúng Nút. Học bán giám sát, bao gồm sự kết hợp giữa dữ liệu có nhãn & không có nhãn, mang đến 1 cơ hội quý giá để so sánh các kỹ thuật nhúng khác nhau. Trong chương này, khám phá cách sử dụng nhúng GNN & N2V để dự đoán nhãn khi phần lớn dữ liệu thiếu nhãn.

Our task involves Political Books dataset `books_graph`, where nodes represent political books & edges indicate co-purchase relationships. To make process clearer, review steps taken so far & outline our next steps, as illustrated in Fig. 2.9: Overview of steps taken in Chap. 2: 1. Preprocess Political Books dataset for embedding. 2. Use N2V & GCN to create embeddings from preprocessed data. 3. Prepare N2V embeddings & GCN embeddings for semi-supervised classification. 4. Embeddings are used as features in a random forest classifier (tabular features) & a GCN classifier (node features).

– Nhiệm vụ của chúng tôi liên quan đến tập dữ liệu Sách Chính trị `books_graph`, trong đó các nút biểu diễn các cuốn sách chính trị & các cạnh biểu diễn mối quan hệ đồng mua. Để làm rõ quy trình, xem lại các bước đã thực hiện cho đến nay & phác thảo các bước tiếp theo, như minh họa trong Hình 2.9: Tổng quan các bước đã thực hiện trong Chương 2: 1. Tiền xử lý tập dữ liệu Sách Chính trị để nhúng. 2. Sử dụng N2V & GCN để tạo nhúng từ dữ liệu đã được xử lý trước. 3. Chuẩn bị nhúng N2V & nhúng GCN cho phân loại bán giám sát. 4. Nhúng được sử dụng làm đặc trưng trong bộ phân loại rừng ngẫu nhiên (đặc trưng dạng bảng) & bộ phân loại GCN (đặc trưng nút).

Began with `books_graph` dataset in graph format & performed light preprocessing to prepare data for embedding. For N2V, this involved converting dataset from a `.gml` file to a `NetworkX` format. For GNN-based embeddings, converted `NetworkX` graph into a PyTorch tensor & initialized node features using Xavier initialization to ensure balanced variability across layers.

– Bắt đầu với tập dữ liệu `books_graph` ở định dạng đồ thị & thực hiện tiền xử lý nhẹ để chuẩn bị dữ liệu cho nhúng. Đối với N2V, việc này bao gồm chuyển đổi tập dữ liệu từ tệp `.gml` sang định dạng `NetworkX`. Đối với nhúng dựa trên GNN, chuyển đổi đồ thị `NetworkX` thành tensor PyTorch & khởi tạo các đặc trưng nút bằng khởi tạo Xavier để đảm bảo tính biến thiên cân bằng giữa các lớp.

After preparing data, we generated embeddings using both N2V & GCNs. Now, in this sect, apply these embeddings to a semi-supervised classification problem. This involves further processing to define classification task, where only 20% of book labels are retained, simulating a realistic scenario with sparse labeled data.

– Sau khi chuẩn bị dữ liệu, chúng tôi đã tạo ra các nhúng sử dụng cả N2V & GCN. Trong phần này, chúng tôi áp dụng các nhúng này vào 1 bài toán phân loại bán giám sát. Điều này bao gồm việc xử lý thêm để xác định tác vụ phân loại, trong đó chỉ giữ lại 20% nhãn sách, mô phỏng 1 kịch bản thực tế với dữ liệu được gán nhãn thưa thớt.

Use 2 sets of embeddings (N2V & GCN) with 2 different classifiers: a random forest classifier (to use embeddings as tabular features) & a GCN classifier (to use graph structure & node features). Goal: predict political orientation of books, with remaining 80% of labels inferred based on given embeddings.

– Sử dụng 2 bộ nhúng (N2V & GCN) với 2 bộ phân loại khác nhau: 1 bộ phân loại rừng ngẫu nhiên (để sử dụng nhúng làm đặc trưng dạng bảng) & 1 bộ phân loại GCN (để sử dụng cấu trúc đồ thị & đặc trưng nút). Mục tiêu: dự đoán khuynh hướng chính trị của sách, với 80% nhãn còn lại được suy ra dựa trên các nhúng đã cho.

* 2.3.1. Data preprocessing. To start, we do a little more preprocessing to our `books_gml` dataset Listing 2.5: Preprocessing for semi-supervised problem. Must format labels in a suitable way for learning process. Because all nodes are labeled, we also have to set up semi-supervised problem by randomly selecting nodes from which we hide labels.

– Tiền xử lý dữ liệu. Để bắt đầu, chúng ta thực hiện thêm 1 chút tiền xử lý cho tập dữ liệu `books_gml` Liệt kê 2.5: Tiền xử lý cho bài toán bán giám sát. Phải định dạng nhãn theo cách phù hợp cho quá trình học. Vì tất cả các nút đều được gán nhãn, chúng ta cũng phải thiết lập bài toán bán giám sát bằng cách chọn ngẫu nhiên các nút mà chúng ta ẩn nhãn. Nodes associated with attribute 'c' are classified as 'right', while those with 'l' are classified as 'left'. Nodes that do not fit these criteria, including those with neutral or unspecified attributes, are categorized as 'neutral'. These

- 2.4. Under Hood.

PART 3: GRAPH NEURAL NETWORKS GNNs

- 3. Graph convolutional networks & GraphSAGE.
- 4. Graph attention networks.
- 5. Graph autoencoders.

PART 3: ADVANCED TOPICS.

- 6. Dynamic graphs: Spatiotemporal GNNs.
- 7. Learning & inference at scale.
- 8. Considerations for GNN projects.
- A. Discovering graphs.
- B. Installing & configuring PyTorch Geometric.

1.2 MAXIME LABONNE. Hands-On Graph Neural Networks Using Python: Practical Techniques & Architectures for Building Powerful Graph & DL Apps with PyTorch. 2023

- Preface. In just 10 years, GNNs have become an essential & popular DL architecture. They have already had a significant impact various industries, e.g. in drug discovery, where GNNs predicted a new antibiotic, named halicin, & have improved estimated time of arrival calculations on Google Maps. Tech companies & universities are exploring potential of GNNs in various applications, including recommender systems, fake news detection, & chip design. GNNs have enormous potential & many yet-to-be-discovered applications, making them a critical tool for solving global problems.

– Chỉ trong 10 năm, GNN đã trở thành 1 kiến trúc DL thiết yếu & phổ biến. Chúng đã có tác động đáng kể đến nhiều ngành công nghiệp, ví dụ như trong lĩnh vực khám phá thuốc, nơi GNN dự đoán 1 loại kháng sinh mới, tên là halicin, & đã cải thiện khả năng tính toán thời gian đến ước tính trên Google Maps. Các công ty công nghệ & các trường đại học đang khám phá tiềm năng của GNN trong nhiều ứng dụng khác nhau, bao gồm hệ thống đề xuất, phát hiện tin giả, & thiết kế chip. GNN có tiềm năng to lớn & nhiều ứng dụng chưa được khám phá, khiến chúng trở thành 1 công cụ quan trọng để giải quyết các vấn đề toàn cầu.

In this book, aim to provide a comprehensive & practical overview of world of GNNs. Begin by exploring fundamental concepts of graph theory & graph learning & then delve into most widely used & well-established GNN architectures. As we progress, also cover latest advances in GNNs & introduce specialized architectures that are designed to tackle specific tasks, e.g. graph generation, link prediction, & more.

– Trong cuốn sách này, chúng tôi đặt mục tiêu cung cấp 1 cái nhìn tổng quan toàn diện & thực tiễn về thế giới GNN. Bắt đầu bằng việc khám phá các khái niệm cơ bản của lý thuyết đồ thị & học đồ thị & sau đó đi sâu vào các kiến trúc GNN được sử dụng rộng rãi nhất & đã được thiết lập tốt. Khi chúng ta tiến triển, chúng tôi cũng sẽ đề cập đến những tiến bộ mới nhất trong GNN & giới thiệu các kiến trúc chuyên biệt được thiết kế để giải quyết các nhiệm vụ cụ thể, ví dụ: tạo đồ thị, dự đoán liên kết, & nhiều hơn nữa.

In addition to these specialized chaps, provide hands-on experience through 3 practical projects. These projects will cover critical real-world applications of GNNs, including traffic forecasting, anomaly detection, & recommender systems. Through these projects, gain a deeper understanding of how GNNs work & also develop skills to implement them in practical scenarios.

– Ngoài các chuyên gia này, chương trình còn cung cấp kinh nghiệm thực tế thông qua 3 dự án thực tế. Các dự án này sẽ đề cập đến các ứng dụng quan trọng của GNN trong thế giới thực, bao gồm dự báo lưu lượng truy cập, phát hiện bất thường và hệ thống đề xuất. Thông qua các dự án này, bạn sẽ hiểu sâu hơn về cách thức hoạt động của GNN và phát triển kỹ năng triển khai chúng trong các tình huống thực tế.

Finally, this book provides a hands-on learning experience with readable code for every chap's techniques & relevant applications, which are readily accessible on GitHub & Google Colab. By end of this book, have a comprehensive understanding of field of graph learning & GNNs & will be well-equipped to design & implement these models for a wide range of applications.

– Cuối cùng, cuốn sách này cung cấp trải nghiệm học tập thực hành với mã nguồn dễ đọc cho mọi kỹ thuật & các ứng dụng liên quan, có thể dễ dàng truy cập trên GitHub & Google Colab. Khi hoàn thành cuốn sách này, bạn sẽ có hiểu biết toàn diện về lĩnh vực học đồ thị & Mạng lưới Mạng Thần kinh Nhân tạo (GNN) & được trang bị tốt để thiết kế & triển khai các mô hình này cho nhiều ứng dụng khác nhau.

- Who this book is for. This book is intended for individuals interested in learning about GNNs & how they can be applied to various real-world problems. This book is ideal for *data scientists*, *ML engineers*, & *AI professionals* who want to gain practical experience in designing & implementing GNNs. This book is written for individuals with prior knowledge of DL & ML. However, it provides a comprehensive introduction to fundamental concepts of graph theory & graph learning for those new to field. Also be useful for researchers & students in computer science, mathematics, & engineering who want to expand their knowledge in this rapidly growing area of research.

– Cuốn sách này dành cho những cá nhân quan tâm đến việc tìm hiểu về mạng lưới thần kinh nhân tạo (GNN) & cách chúng có thể được áp dụng cho các vấn đề thực tế khác nhau. Cuốn sách này lý tưởng cho các nhà khoa học dữ liệu, kỹ sư ML, & AI chuyên gia muốn tích lũy kinh nghiệm thực tế trong việc thiết kế & triển khai GNN. Cuốn sách này được viết cho những cá nhân đã có kiến thức về DL & ML. Tuy nhiên, nó cung cấp 1 giới thiệu toàn diện về các khái niệm cơ bản của lý thuyết đồ thị & học đồ thị cho những người mới bắt đầu. Nó cũng hữu ích cho các nhà nghiên cứu & sinh viên ngành khoa học máy tính, toán học, & kỹ thuật muốn mở rộng kiến thức trong lĩnh vực nghiên cứu đang phát triển nhanh chóng này.

- What this book covers. Chap. 1: Getting Started with Graph Learning, provides a comprehensive introduction to GNNs, including their importance in modern data analysis & ML. Chap starts by exploring relevance of graphs as a representation of data & their widespread use in various domains. It then delves into importance of graph learning, including different

applications & techniques. Finally, chap focuses on GNN architecture & highlights its unique features & performance compared to other methods.

– Chương 1: Bắt đầu với Học đồ thị, cung cấp phần giới thiệu toàn diện về GNN, bao gồm tầm quan trọng của chúng trong phân tích dữ liệu hiện đại & Học máy (ML). Chương này bắt đầu bằng việc khám phá tính liên quan của đồ thị như 1 phương tiện biểu diễn dữ liệu & ứng dụng rộng rãi của chúng trong nhiều lĩnh vực. Sau đó, chương đi sâu vào tầm quan trọng của học đồ thị, bao gồm các ứng dụng & kỹ thuật khác nhau. Cuối cùng, chương tập trung vào kiến trúc GNN & làm nổi bật các tính năng độc đáo & hiệu suất của nó so với các phương pháp khác.

Chap. 2: Graph Theory for Graph Neural Networks, covers basics of graph theory & introduces various types of graphs, including their properties & applications. This chap also covers fundamental graph concepts, e.g. adjacency matrix, graph measures, e.g. centrality, & graph algorithms, BFS & DFS.

– Chương 2: Lý thuyết Đồ thị cho Mạng Nơ-ron Đồ thị, bao gồm những kiến thức cơ bản về lý thuyết đồ thị & giới thiệu các loại đồ thị khác nhau, bao gồm các tính chất & ứng dụng của chúng. Chương này cũng đề cập đến các khái niệm cơ bản về đồ thị, ví dụ như ma trận kề, độ đo đồ thị, ví dụ như độ tập trung, & thuật toán đồ thị, BFS & DFS.

Chap. 3: Creating Node Representations with DeepWalk, focused on DeepWalk, a pioneer in applying ML to graph data. Main objective of DeepWalk architecture: generate node representations that other models can utilize for downstream tasks e.g. node classification. Chap covers 2 key components of DeepWalk – Word2Vec & random walks – with a particular emphasis on Word2Vec skip-gram model.

– Chương 3: Tạo Biểu diễn Nút với DeepWalk, tập trung vào DeepWalk, 1 công nghệ tiên phong trong việc áp dụng Học máy (ML) vào dữ liệu đồ thị. Mục tiêu chính của kiến trúc DeepWalk: tạo ra các biểu diễn nút mà các mô hình khác có thể sử dụng cho các tác vụ hạ nguồn, ví dụ như phân loại nút. Chương này đề cập đến 2 thành phần chính của DeepWalk – Word2Vec & random walks – đặc biệt nhấn mạnh vào mô hình Word2Vec skip-gram.

Chap. 4: Improving Embeddings with Biased Random Walks in Node2Vec, focused on Node2Vec architecture, which is based on DeepWalk architecture covered in previous chap. Chap covers modifications made to random walk generation in Node2Vec & how to select best parameters for a specific graph. Implementation of Node2Vec is compared to DeepWalk on Zachary’s Karate Club to highlight differences between 2 architectures. Chap concludes with a practical application of Node2Vec, building a movie recommendation system.

– Chương 4: Cải thiện nhúng với bước ngẫu nhiên có thiên vị trong Node2Vec, tập trung vào kiến trúc Node2Vec, dựa trên kiến trúc DeepWalk đã được đề cập trong chương trước. Chương này đề cập đến những thay đổi được thực hiện đối với việc tạo bước ngẫu nhiên trong Node2Vec & cách chọn tham số tốt nhất cho 1 đồ thị cụ thể. Việc triển khai Node2Vec được so sánh với DeepWalk trên Zachary’s Karate Club để làm nổi bật sự khác biệt giữa hai kiến trúc. Chương kết thúc bằng 1 ứng dụng thực tế của Node2Vec, xây dựng hệ thống đề xuất phim.

Chap. 5: Including Node Features with Vanilla Neural Networks, explores integration of additional information, e.g. node & edge features, into graph embeddings to produce more accurate results. Chap starts with a comparison of vanilla neural network’s performance on node features only, treated as tabular datasets. Then, experiment with adding topological information to neural networks, leading to creation of a simple vanilla GNN architecture.

– Chương 5: Bao gồm các đặc điểm nút với mạng nơ-ron nhân tạo thuần túy, khám phá việc tích hợp thông tin bổ sung, ví dụ như đặc điểm nút & cạnh, vào nhúng đồ thị để tạo ra kết quả chính xác hơn. Chương này bắt đầu bằng việc so sánh hiệu suất của mạng nơ-ron nhân tạo thuần túy chỉ trên các đặc điểm nút, được xử lý dưới dạng tập dữ liệu bảng. Sau đó, thử nghiệm thêm thông tin tô pô vào mạng nơ-ron, dẫn đến việc tạo ra 1 kiến trúc mạng nơ-ron nhân tạo thuần túy đơn giản.

Chap. 6: Introducing Graph Convolutional Networks, focuses on Graph Convolutional Network (GCN) architecture & its importance as a blueprint for GNNs. It covers limitations of previous vanilla GNN layers & explains motivation behind GCNs. Chap details how GCN layer works, its performance improvements over vanilla GNN layer, & its implementation on Cora & Facebook Page-Page datasets using PyTorch Geometric. Chap also touches upon task of node regression & benefits of transforming tabular data into a graph.

– Chương 6: Giới thiệu về Mạng Tích chập Đồ thị, tập trung vào kiến trúc Mạng Tích chập Đồ thị (GCN) & tầm quan trọng của nó như 1 bản thiết kế cho GNN. Chương này đề cập đến những hạn chế của các lớp GNN thuần túy trước đây & giải thích động lực đằng sau GCN. Chương này trình bày chi tiết cách thức hoạt động của lớp GCN, những cải tiến về hiệu suất so với lớp GNN thuần túy, & triển khai nó trên Cora & bộ dữ liệu Facebook Page-Page bằng PyTorch Geometric. Chương cũng đề cập đến nhiệm vụ của hồi quy nút & lợi ích của việc chuyển đổi dữ liệu dạng bảng thành dạng đồ thị.

Chap. 7: Graph Attention Networks, focuses on Graph Attention Networks (GATs), which are an improvements over GCNs. Chap explains how GATs work by using concepts of self-attention & provides a step-by-step understanding of graph attention layer. Chap also implements a graph attention layer from scratch using NumPy. Final section of chap discusses use of a GAT on 2 node classification datasets, Cora & CiteSeer, & compares accuracy with that of a GCN.

– Chương 7: Mạng lưới Chú ý Đồ thị, tập trung vào Mạng lưới Chú ý Đồ thị (GAT), 1 cải tiến so với GCN. Chương này giải thích cách thức hoạt động của GAT bằng cách sử dụng các khái niệm tự chú ý & cung cấp hiểu biết từng bước về lớp chú ý đồ thị. Chương cũng triển khai 1 lớp chú ý đồ thị từ đầu bằng NumPy. Phần cuối của chương thảo luận về việc sử dụng GAT trên các tập dữ liệu phân loại 2 nút, Cora & CiteSeer, & so sánh độ chính xác với GCN.

Chap. 8: Scaling up GNNs with GraphSAGE, focuses on GraphSAGE architecture & its ability to handle large graphs effectively. Chap covers 2 main ideas behind GraphSAGE, including its neighbor sampling technique & aggregation operators. Learn about variants proposed by tech companies e.g. Uber Eats & Pinterest, as well as benefits of GraphSAGE’s inductive approach. Chap concludes by implementing GraphSAGE for node classification & multi-label classification tasks.

– Chương 8: Mở rộng mạng lưới mô hình dữ liệu lớn (GNN) với GraphSAGE, tập trung vào kiến trúc GraphSAGE & khả năng xử lý đồ thị lớn hiệu quả. Chương này đề cập đến 2 ý tưởng chính đằng sau GraphSAGE, bao gồm kỹ thuật lấy mẫu lân cận & các toán tử tổng hợp. Tìm hiểu về các biến thể được đề xuất bởi các công ty công nghệ, ví dụ như Uber Eats & Pinterest, cũng như lợi ích của phương pháp quy nạp của GraphSAGE. Chương kết thúc bằng việc triển khai GraphSAGE cho các tác vụ phân loại nút & phân loại đa nhãn.

Chap. 9: Defining Expressiveness for Graph Classification, explores concept of expressiveness in GNNs & how it can be used to design better models. It introduces Weisfeiler-Leman (WL) test, which provides framework for understanding expressiveness in GNNs. Chap uses WL test to compare different GNN layers & determine most expressive one. Based on this result, a more powerful GNN is designed & implemented by PyTorch Geometric. Chap concludes with a comparison of different methods for graph classification on PROTEINS dataset.

– Chương 9: Định nghĩa tính biểu cảm cho phân loại đồ thị, khám phá khái niệm tính biểu cảm trong mạng lưới biểu diễn dữ liệu (GNN) & cách sử dụng nó để thiết kế các mô hình tốt hơn. Chương này giới thiệu kiểm định Weisfeiler-Leman (WL), cung cấp khuôn khổ để hiểu tính biểu cảm trong GNN. Chương sử dụng kiểm định WL để so sánh các lớp GNN khác nhau & xác định lớp biểu cảm nhất. Dựa trên kết quả này, 1 GNN mạnh mẽ hơn đã được thiết kế & triển khai bởi PyTorch Geometric. Chương kết thúc bằng việc so sánh các phương pháp phân loại đồ thị khác nhau trên tập dữ liệu PROTEIN.

Chap. 10: Predicting Links with GNNs, focuses on link prediction in graphs. It covers traditional techniques, e.g. matrix factorization & GNN-based methods. Chap explains concept of link prediction & its importance in social networks & recommender systems. Learn about limitations of traditional techniques & benefits of using GNN-based methods. Explore 3 GNN-based techniques from 2 different families, including node embeddings & subgraph representation. Finally, implement various link prediction techniques in PyTorch Geometric & choose best method for a given problem.

– Chương 10: Dự đoán Liên kết với GNN, tập trung vào dự đoán liên kết trong đồ thị. Chương này đề cập đến các kỹ thuật truyền thống, ví dụ như phân tích ma trận & các phương pháp dựa trên GNN. Chương này giải thích khái niệm dự đoán liên kết & tầm quan trọng của nó trong mạng xã hội & các hệ thống đề xuất. Tìm hiểu về những hạn chế của các kỹ thuật truyền thống & lợi ích của việc sử dụng các phương pháp dựa trên GNN. Khám phá 3 kỹ thuật dựa trên GNN từ 2 họ khác nhau, bao gồm những nút & biểu diễn đồ thị con. Cuối cùng, triển khai các kỹ thuật dự đoán liên kết khác nhau trong PyTorch Geometric & chọn phương pháp tốt nhất cho 1 bài toán nhất định.

Chap. 11: Generating Graphs Using GNNs, explores field of graph generation, which involves finding methods to create new graphs. Chap 1st introduces you to traditional techniques e.g. Erdős-Rényi & small-world models. Then focus on 3 families of solutions for GNN-based graph generation: VAE-based, autoregressive, & GAN-based models. Chap concludes with an implementation of a GAN-based framework with Reinforcement Learning (RL) to generate new chemical compounds using DeepChem library with TensorFlow.

– Chương 11: Tạo đồ thị bằng GNN, khám phá lĩnh vực tạo đồ thị, bao gồm việc tìm kiếm các phương pháp để tạo đồ thị mới. Chương 1 giới thiệu các kỹ thuật truyền thống, ví dụ như mô hình Erdos-Rényi và mô hình thế giới nhỏ. Sau đó, tập trung vào 3 nhóm giải pháp tạo đồ thị dựa trên GNN: mô hình dựa trên VAE, mô hình tự hồi quy và mô hình GAN. Chương kết thúc bằng việc triển khai 1 khuôn khổ dựa trên GAN với Học Tăng cường (RL) để tạo ra các hợp chất hóa học mới bằng thư viện DeepChem với TensorFlow.

Chap. 12: Learning from Heterogeneous Graphs, focuses on heterogeneous GNNs. Heterogeneous graphs contain different types of nodes & edges, in contrast in homogeneous graphs, which only involve 1 type of node & 1 type of edge. Chap begins by reviewing *Message Passing Neural Network* (MPNN) framework for homogeneous GNNs, then expands framework to heterogeneous networks. Finally, introduce a technique for creating a heterogeneous dataset, transforming homogeneous architectures into heterogeneous ones, & discussing an architecture specifically designed for processing heterogeneous networks.

– Chương 12: Học từ Đồ thị Không đồng nhất, tập trung vào Mạng nơ-ron nhân tạo không đồng nhất (GNN) không đồng nhất. Đồ thị không đồng nhất chứa các loại nút & cạnh khác nhau, trái ngược với đồ thị đồng nhất, chỉ bao gồm 1 loại nút & 1 loại cạnh. Chương bắt đầu bằng việc xem xét khuôn khổ Mạng nơ-ron truyền thông điệp (MPNN) cho GNN đồng nhất, sau đó mở rộng khuôn khổ sang các mạng không đồng nhất. Cuối cùng, giới thiệu 1 kỹ thuật để tạo 1 tập dữ liệu không đồng nhất, chuyển đổi các kiến trúc đồng nhất thành kiến trúc không đồng nhất, và thảo luận về 1 kiến trúc được thiết kế riêng để xử lý các mạng không đồng nhất.

Chap. 13: Temporal GNNs, focuses on Temporal GNNs, or Spatio-Temporal GNNs, which are a type of GNN that can handle graphs with changing edges & features over time. Chap 1st explains concept of dynamic graphs & applications of temporal GNNs, focusing on time series forecasting. Chap then moves on to application of temporal GNNs to web traffic forecasting to improve results using temporal information. Finally, chap describes another temporal GNN architecture specifically designed for dynamic graphs & applies it to task of epidemic forecasting.

– Chương 13: Mạng GNN thời gian, tập trung vào Mạng GNN thời gian, hay Mạng GNN không gian-thời gian, là 1 loại Mạng GNN có thể xử lý đồ thị với các cạnh & đặc trưng thay đổi theo thời gian. Chương 1 giải thích khái niệm về đồ thị động & ứng dụng của Mạng GNN thời gian, tập trung vào dự báo chuỗi thời gian. Sau đó, chương chuyển sang ứng dụng Mạng GNN thời gian vào dự báo lưu lượng truy cập web để cải thiện kết quả bằng cách sử dụng thông tin thời gian. Cuối cùng, chương này mô tả 1 kiến trúc Mạng GNN thời gian khác được thiết kế riêng cho đồ thị động & ứng dụng nó vào nhiệm vụ dự báo dịch bệnh.

- Chap. 14: Explaining GNNs, covers various techniques to better understands predictions & behavior of a GNN model. Chap highlights 2 popular explanation methods: GNNExplainer & integrated gradients. Then, see application of these techniques on a graph classification task using MUTAG dataset & a node classification task using Twitch social network.

- Chương 14: Giải thích về mạng GNN, bao gồm các kỹ thuật khác nhau để hiểu rõ hơn về dự đoán & hành vi của mô hình GNN. Chương này nêu bật 2 phương pháp giải thích phổ biến: GNNExplainer & gradient tích hợp. Sau đó, xem xét ứng dụng của các kỹ thuật này trên 1 bài toán phân loại đồ thị sử dụng tập dữ liệu MUTAG & 1 bài toán phân loại nút sử dụng mạng xã hội Twitch.
 - Chap. 15: Forecasting Traffic Using A3T-GCN, focuses on application of Temporal Graph Neural Networks in field of traffic forecasting. It highlights importance of accurate traffic forecasts in smart cities & challenges of traffic forecasting due to complex spatial & temporal dependencies. Chap covers steps involved in processing a new dataset to create a temporal graph & implementation of a new type of temporal GNN to predict future traffic speed. Finally, results are compared to a baseline solution to verify relevance of architecture.
 - Chương 15: Dự báo lưu lượng giao thông bằng A3T-GCN, tập trung vào ứng dụng của Mạng nơ-ron đồ thị thời gian trong lĩnh vực dự báo giao thông. Bài viết nhấn mạnh tầm quan trọng của việc dự báo giao thông chính xác trong các thành phố thông minh & những thách thức của việc dự báo giao thông do sự phụ thuộc phức tạp về không gian & thời gian. Chương này trình bày các bước xử lý tập dữ liệu mới để tạo đồ thị thời gian & triển khai 1 loại Mạng nơ-ron đồ thị thời gian mới để dự đoán tốc độ giao thông trong tương lai. Cuối cùng, kết quả được so sánh với giải pháp cơ sở để đánh giá tính phù hợp thực tế của kiến trúc.
 - Chap. 16: Recommending Books Using LightGCN, focuses on application of GNNs in recommender systems. Goal of recommender systems: provide personalized recommendations to users based on their interests & past interactions. GNNs are well-suited for this task as they can effectively incorporate complex relationships between users & items. In this chap, LightGCN architecture is introduced as a GNN specifically designed for recommender systems. Using Book-Crossing dataset, chap demonstrates how to build a book recommender system with collaborative filtering using LightGCN architecture.
 - Chương 16: Đề xuất sách bằng LightGCN, tập trung vào ứng dụng của GNN trong hệ thống đề xuất. Mục tiêu của hệ thống đề xuất: cung cấp các đề xuất được cá nhân hóa cho người dùng dựa trên sở thích & các tương tác trước đây của họ. GNN rất phù hợp cho nhiệm vụ này vì chúng có thể kết hợp hiệu quả các mối quan hệ phức tạp giữa người dùng & các mục. Trong chương này, kiến trúc LightGCN được giới thiệu như 1 GNN được thiết kế riêng cho hệ thống đề xuất. Sử dụng tập dữ liệu Book-Crossing, chương này trình bày cách xây dựng 1 hệ thống đề xuất sách với lọc cộng tác bằng kiến trúc LightGCN.
 - Chap. 17: Recommending Books Using LightGCN, focuses on application of GNNs in recommender systems. Goal of recommender systems: provide personalized recommendations to users based on their interests & past interactions. GNNs are well-suited for this tasks as they can effectively incorporate complex relationships between users & items. In this chap, LightGCN architecture is introduced as a GNN specifically designed for recommender systems. Using Book-Crossing dataset, chap demonstrates how to build a book recommender system with collaborative filtering using LightGCN architecture.
 - Chương 17: Đề xuất sách bằng LightGCN, tập trung vào ứng dụng của GNN trong các hệ thống đề xuất. Mục tiêu của hệ thống đề xuất: cung cấp các đề xuất được cá nhân hóa cho người dùng dựa trên sở thích & các tương tác trước đây của họ. GNN rất phù hợp cho nhiệm vụ này vì chúng có thể kết hợp hiệu quả các mối quan hệ phức tạp giữa người dùng & các mục. Trong chương này, kiến trúc LightGCN được giới thiệu như 1 GNN được thiết kế riêng cho các hệ thống đề xuất. Sử dụng tập dữ liệu Book-Crossing, chương này trình bày cách xây dựng 1 hệ thống đề xuất sách với lọc cộng tác bằng kiến trúc LightGCN.
 - Chap. 18: Unlocking Potential of GNNs for Real-World Applications, summarizes what we have learned throughout book, & looks ahead to future of GNNs.
 - Chương 18: Khai phá tiềm năng của GNN cho các ứng dụng thực tế, tóm tắt những gì chúng ta đã học được trong suốt cuốn sách, & hướng tới tương lai của GNN.
 - To get most out of this book. Should have a basic understanding of graph theory & ML concepts, e.g. supervised & unsupervised learning, training, & evaluation of models to maximize your learning experience. Familiarity with DL frameworks, e.g. PyTorch, will also be useful, although not essential, as book will provide a comprehensive introduction to mathematical concepts & their implementation.
 - Để tận dụng tối đa cuốn sách này. Bạn nên có hiểu biết cơ bản về lý thuyết đồ thị & các khái niệm ML, ví dụ: học có giám sát & học không giám sát, đào tạo, & đánh giá các mô hình để tối đa hóa trải nghiệm học tập của mình. Việc quen thuộc với các nền tảng DL, ví dụ: PyTorch, cũng sẽ hữu ích, mặc dù không bắt buộc, vì cuốn sách sẽ cung cấp phần giới thiệu toàn diện về các khái niệm toán học & cách triển khai chúng.
- Software covered in book: Python 3.8.15, PyTorch 1.13.1, PyTorch Geometric 2.2.0.

PART 1: INTRODUCTION TO GRAPH LEARNING.

In recent years, graph representation of data has become increasingly prevalent across various domains, from social networks to molecular biology. It is crucial to have a deep understanding of GNNs, which are designed specifically to handle graph-structured data, to unlock full potential of this representation.

– Trong những năm gần đây, biểu diễn dữ liệu dạng đồ thị ngày càng trở nên phổ biến trong nhiều lĩnh vực, từ mạng xã hội đến sinh học phân tử. Việc hiểu sâu sắc về mạng lưới mô hình dữ liệu (GNN), được thiết kế chuyên biệt để xử lý dữ liệu có cấu trúc đồ thị, là rất quan trọng để khai thác toàn bộ tiềm năng của biểu diễn này.

This 1st part part consists of 2 chaps & serves as a solid foundation for rest of book. It introduces concepts of graph learning & GNNs & their relevance in numerous tasks & industries. It also covers fundamental concepts of graph theory & its applications

in graph learning, e.g. graph centrality measures. This part also highlights unique features & performance of GNN architecture compared to other methods.

– Phần 1 này bao gồm 2 chương & đóng vai trò là nền tảng vững chắc cho phần còn lại của cuốn sách. Phần này giới thiệu các khái niệm về học đồ thị & Mạng lưới Mạng Toàn cầu (GNN) & sự liên quan của chúng trong nhiều tác vụ & ngành công nghiệp. Phần này cũng đề cập đến các khái niệm cơ bản của lý thuyết đồ thị & ứng dụng của nó trong học đồ thị, ví dụ như các phép đo tính trung tâm của đồ thị. Phần này cũng làm nổi bật các tính năng độc đáo & hiệu suất của kiến trúc GNN so với các phương pháp khác.

By end of this part, have a solid understanding of importance of GNNs in solving many real-world problems. Will be acquainted with essentials of graph learning & how it is used in various domains. Furthermore, will have a comprehensive overview of main concepts of graph theory used in later chaps. With this solid foundation, will be well equipped to move on to more advanced concepts in graph learning & GNNs in following parts of book.

– Đến cuối phần này, bạn sẽ hiểu rõ tầm quan trọng của mạng GNN trong việc giải quyết nhiều vấn đề thực tế. Bạn sẽ được làm quen với những kiến thức cơ bản về học đồ thị & cách thức ứng dụng nó trong nhiều lĩnh vực khác nhau. Hơn nữa, bạn sẽ có cái nhìn tổng quan toàn diện về các khái niệm chính của lý thuyết đồ thị được sử dụng trong các chương sau. Với nền tảng vững chắc này, bạn sẽ được trang bị tốt để tiếp tục học các khái niệm nâng cao hơn về học đồ thị & mạng GNN trong các phần tiếp theo của cuốn sách.

- **1. Getting Started with Graph Learning.** In this chap, delve into foundations of GNNs & understand why they are crucial tools in modern data analysis & ML. To that end, answer 3 essential questions providing us with a comprehensive understanding of GNNs.

– Trong chương này, chúng ta sẽ đi sâu vào nền tảng của GNN & hiểu tại sao chúng là công cụ quan trọng trong phân tích dữ liệu hiện đại & ML. Để đạt được mục tiêu đó, trả lời 3 câu hỏi thiết yếu, giúp chúng ta hiểu toàn diện về GNN.

1st, explore significance of graphs as a representation of data, & why they are widely used in various domains e.g. CS, biology, & finance. Next, delve into importance of graph learning, where we will understand different applications of graph learning & different families of graph learning techniques. Finally, focus on GNN family, highlighting its unique features, performance, & how it stands out compared to other methods.

– Đầu tiên, khám phá ý nghĩa của đồ thị trong việc biểu diễn dữ liệu, & lý do tại sao chúng được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau, ví dụ như Khoa học Máy tính, Sinh học, & Tài chính. Tiếp theo, đi sâu vào tầm quan trọng của học đồ thị, nơi chúng ta sẽ tìm hiểu các ứng dụng khác nhau của học đồ thị & các nhóm kỹ thuật học đồ thị khác nhau. Cuối cùng, tập trung vào họ GNN, làm nổi bật các tính năng độc đáo, hiệu suất của nó, & cách nó nổi bật so với các phương pháp khác.

By end of this chap, have a clear understanding of why GNNs are important & how they can be used to solve real-world problems. Will also be equipped with knowledge & skills you need to dive deeper into more advanced topics.

- **1.1. Why graphs?** 1st question we need to address: why interested in graphs in 1st place? Graph theory, mathematical study of graphs, has emerged as a fundamental tool for understanding complex systems & relationships. A graph is a visual representation of a collection of nodes (also called vertices) & edges that connect these nodes, providing a structure to represent entities & their relationships.

– Tại sao lại là đồ thị? Câu hỏi đầu tiên chúng ta cần giải quyết: tại sao lại quan tâm đến đồ thị ngay từ đầu? Lý thuyết đồ thị, nghiên cứu toán học về đồ thị, đã nổi lên như 1 công cụ cơ bản để hiểu các hệ thống phức tạp & các mối quan hệ. Đồ thị là biểu diễn trực quan của 1 tập hợp các nút (còn gọi là đỉnh) & các cạnh kết nối các nút này, cung cấp 1 cấu trúc để biểu diễn các thực thể & các mối quan hệ của chúng.

By representing a complex system as a network of entities with interactions, can analyze their relationships, allowing us to gain a deeper understanding of their underlying structures & patterns. Versatility of graphs makes them a popular choice in various domains, including:

- * CS, where graphs can be used to model structure of computer programs, making it easier to understand how different components of a system interact with each other
- * Physics, where graphs can be used to model physical systems & their interactions, e.g. relationship between particles & their properties
- * Biology, where graphs can be used to model biological systems, e.g. metabolic pathways, as a network of interconnected entities
- * Social sciences, where graphs can be used to study & understand complex social networks, including relationships between individuals in a community
- * Finance, where graphs can be used to analyze stock market trends & relationships between different financial instruments
- * Engineering, where graphs can be used to model & analyze complex systems, e.g. transportation networks & electrical power grids

These domains naturally exhibit a relational structure. E.g., graphs are a natural representation of social networks: nodes are users, & edges represent friendships. But graphs are so versatile they can also be applied to domains where relationship structure is less natural, unlocking new insights & understanding.

– Bằng cách biểu diễn 1 hệ thống phức tạp như 1 mạng lưới các thực thể có tương tác, chúng ta có thể phân tích các mối quan hệ của chúng, cho phép chúng ta hiểu sâu hơn về cấu trúc & các mô hình cơ bản của chúng. Tính linh hoạt của đồ thị khiến chúng trở thành lựa chọn phổ biến trong nhiều lĩnh vực, bao gồm:

- * Khoa học máy tính, trong đó đồ thị có thể được sử dụng để mô hình hóa cấu trúc của các chương trình máy tính, giúp dễ dàng hiểu được cách các thành phần khác nhau của 1 hệ thống tương tác với nhau.
- * Vật lý, trong đó đồ thị có thể được sử dụng để mô hình hóa các hệ thống vật lý & các tương tác của chúng, ví dụ: mối quan hệ giữa các hạt & các tính chất của chúng
- * Sinh học, trong đó đồ thị có thể được sử dụng để mô hình hóa các hệ thống sinh học, ví dụ: các con đường trao đổi chất, như 1 mạng lưới các thực thể được kết nối với nhau
- * Khoa học xã hội, trong đó đồ thị có thể được sử dụng để nghiên cứu & hiểu các mạng lưới xã hội phức tạp, bao gồm các mối quan hệ giữa các cá nhân trong 1 cộng đồng
- * Tài chính, trong đó đồ thị có thể được sử dụng để phân tích xu hướng thị trường chứng khoán & mối quan hệ giữa các công cụ tài chính khác nhau
- * Kỹ thuật, trong đó đồ thị có thể được sử dụng để mô hình hóa & phân tích các hệ thống phức tạp, ví dụ: Mạng lưới giao thông & lưới điện

Các miền này tự nhiên thể hiện 1 cấu trúc quan hệ. E.g., đồ thị là 1 biểu diễn tự nhiên của mạng xã hội: các nút là người dùng, & các cạnh biểu diễn tình bạn. Tuy nhiên, đồ thị rất linh hoạt nên chúng cũng có thể được áp dụng cho các miền mà cấu trúc quan hệ ít tự nhiên hơn, mở ra những hiểu biết mới & hiểu biết sâu sắc.

E.g., images can be represented as a graph, as in Fig. 1.2: Original image vs. graph representation of this image. Each pixel is a node, & edges represent relationships between neighboring pixels. This allows for application of graph-based algorithms to image processing & computer vision tasks.

– E.g., hình ảnh có thể được biểu diễn dưới dạng đồ thị, như trong Hình 1.2: Ảnh gốc so với biểu diễn đồ thị của ảnh này. Mỗi pixel là 1 nút, & các cạnh biểu diễn mối quan hệ giữa các pixel lân cận. Điều này cho phép áp dụng các thuật toán dựa trên đồ thị vào xử lý ảnh & các tác vụ thị giác máy tính.

Similarly, a sentence can be transformed into a graph, where nodes are words & edges represent relationships between adjacent words. This approach is useful in natural language processing & information retrieval tasks, where context & meaning of words are critical factors.

– Tại sao lại học đồ thị? Tương tự, 1 câu có thể được chuyển đổi thành đồ thị, trong đó các nút là các từ & các cạnh biểu diễn mối quan hệ giữa các từ liền kề. Cách tiếp cận này hữu ích trong xử lý ngôn ngữ tự nhiên & các tác vụ truy xuất thông tin, trong đó ngữ cảnh & ý nghĩa của từ là các yếu tố quan trọng.

Unlike text & images, graphs do not have a fixed structure. However, this flexibility also makes graphs more challenging to handle. Absence of a fixed structure means they can have an arbitrary number of nodes & edges, with no specific ordering. In addition, graphs can represent dynamic data, where connections between entities can change over time. E.g., relationships between users & products can change as they interact with each other. In this scenario, nodes & edges are updated to reflect changes in real world, e.g. new users, new products, & new relationships. In next sect, delve deeper into how to use graphs with ML to create valuable applications.

– Không giống như văn bản & hình ảnh, đồ thị không có cấu trúc cố định. Tuy nhiên, tính linh hoạt này cũng khiến việc xử lý đồ thị trở nên khó khăn hơn. Việc thiếu cấu trúc cố định đồng nghĩa với việc chúng có thể có số lượng nút & cạnh tùy ý, không theo thứ tự cụ thể. Ngoài ra, đồ thị có thể biểu diễn dữ liệu động, trong đó các kết nối giữa các thực thể có thể thay đổi theo thời gian. Ví dụ: mối quan hệ giữa người dùng & sản phẩm có thể thay đổi khi chúng tương tác với nhau. Trong trường hợp này, các nút & cạnh được cập nhật để phản ánh những thay đổi trong thế giới thực, ví dụ: người dùng mới, sản phẩm mới, & mối quan hệ mới. Trong phần tiếp theo, tìm hiểu sâu hơn về cách sử dụng đồ thị với Học máy (ML) để tạo ra các ứng dụng có giá trị.

o 1.2. Why graph learning? Graph learning is application of ML techniques to graph data. This study area encompasses a range of tasks aimed at understanding & manipulating graph-structured data. There are many graphs learning tasks, including:

- * **Node classification** is a task that involves predicting category (class) of a node in a graph. E.g., it can categorize online users or items based on their characteristics. In this task, model is trained on a set of labeled nodes & their attributes, & it uses this information to predict class of unlabeled nodes.
- * **Link prediction** is a task that involves predicting missing links between pairs of nodes in a graph. This is useful in knowledge graph completion, where goal: complete a graph of entities & their relationships. E.g., it can be used to predict relationships between people based on their social network connections (friend recommendation).
- * **Graph classification** is a task that involves categorizing different graphs into predefined categories. 1 example of this is in molecular biology, where molecular structures can be represented as graphs, & goal: predict their properties for drug design. In this task, model is trained on a set of labeled graphs & their attributes, & it uses this information to categorize unseen graphs.
- * **Graph generation** is a task that involves generating new graphs based on a set of desired properties. 1 of main applications is generating novel molecular structures for drug discovery. This is achieved by training a model on a set of existing molecular structures & then using it to generate new, unseen structures. Generated structures can be evaluated for their potential as drug candidates & further studied.

– Học đồ thị là ứng dụng các kỹ thuật ML vào dữ liệu đồ thị. Lĩnh vực nghiên cứu này bao gồm 1 loạt các nhiệm vụ nhằm mục đích hiểu & thao tác dữ liệu có cấu trúc đồ thị. Có nhiều nhiệm vụ học đồ thị, bao gồm:

- * **Phân loại nút** là 1 nhiệm vụ liên quan đến việc dự đoán loại (lớp) của 1 nút trong đồ thị. Ví dụ: nó có thể phân loại người dùng hoặc mục trực tuyến dựa trên các đặc điểm của họ. Trong nhiệm vụ này, mô hình được huấn luyện trên 1 tập

hợp các nút được gắn nhãn & thuộc tính của chúng, & nó sử dụng thông tin này để dự đoán loại của các nút chưa được gắn nhãn.

- * **Dự đoán liên kết** là 1 nhiệm vụ liên quan đến việc dự đoán các liên kết bị thiếu giữa các cặp nút trong đồ thị. Điều này hữu ích trong việc hoàn thành đồ thị tri thức, với mục tiêu: hoàn thành đồ thị các thực thể & mối quan hệ của chúng. Ví dụ: nó có thể được sử dụng để dự đoán mối quan hệ giữa mọi người dựa trên kết nối mạng xã hội của họ (khuyến nghị bạn bè).
- * **Phân loại đồ thị** là 1 nhiệm vụ liên quan đến việc phân loại các đồ thị khác nhau thành các loại được xác định trước. 1 ví dụ về điều này là trong sinh học phân tử, nơi các cấu trúc phân tử có thể được biểu diễn dưới dạng đồ thị, & mục tiêu: dự đoán các đặc tính của chúng để thiết kế thuốc. Trong nhiệm vụ này, mô hình được huấn luyện trên 1 tập hợp các đồ thị được gắn nhãn & các thuộc tính của chúng, & nó sử dụng thông tin này để phân loại các đồ thị chưa được biết đến.
- * **Tạo đồ thị** là 1 nhiệm vụ liên quan đến việc tạo ra các đồ thị mới dựa trên 1 tập hợp các đặc tính mong muốn. 1 trong những ứng dụng chính là tạo ra các cấu trúc phân tử mới để khám phá thuốc. Điều này đạt được bằng cách huấn luyện 1 mô hình trên 1 tập hợp các cấu trúc phân tử hiện có & sau đó sử dụng nó để tạo ra các cấu trúc mới, chưa được biết đến. Các cấu trúc được tạo ra có thể được đánh giá về tiềm năng của chúng như các ứng cử viên thuốc & nghiên cứu thêm.

Graph learning has many other practical applications that can have a significant impact. 1 of most well-known applications is *recommender systems*, where graph learning algorithms recommend relevant items to users based on their previous interactions & relationships with other items. Another important application is *traffic forecasting*, where graph learning can improve travel time predictions by considering complex relationships between different routes & modes of transportation.

– Học đồ thị có nhiều ứng dụng thực tế khác có thể mang lại tác động đáng kể. 1 trong những ứng dụng nổi tiếng nhất là hệ thống đề xuất, trong đó các thuật toán học đồ thị đề xuất các mục liên quan cho người dùng dựa trên các tương tác trước đó của họ & mối quan hệ với các mục khác. 1 ứng dụng quan trọng khác là dự báo giao thông, trong đó học đồ thị có thể cải thiện dự đoán thời gian di chuyển bằng cách xem xét các mối quan hệ phức tạp giữa các tuyến đường & phương thức vận tải khác nhau.

Versatility & potential of graph learning make it an exciting field of research & development. Study of graphs have advanced rapidly in recent years, driven by availability of large datasets, powerful computing resources, & advancements in ML & AI. As a result, can list 4 prominent families of graph learning techniques [1]:

1. **Graph signal processing**, which applies traditional signal processing methods to graphs, e.g. graph Fourier transform & spectral analysis. These techniques reveal intrinsic properties of graph, e.g. its connectivity & structure.
2. **Matrix factorization**, which seeks to find low-dimensional representations of large matrices. Goal of matrix factorization: identify latent factors or patterns that explain observed relationships in original matrix. This approach can provide a compact & interpretable representation of data.
3. **Random walk**, which refers to a mathematical concept used to model movement of entities in a graph. By simulating random walks over a graph, information about relationships between nodes can be gathered. This is why they are often used to generate training data for ML models.
4. **DL**, which is a subfield of ML that focuses on neural networks with multiple layers. DL methods can effectively encode & represent graph data as vectors. These vectors can then be used in various tasks with remarkable performance.

– Tính linh hoạt & tiềm năng của học đồ thị khiến nó trở thành 1 lĩnh vực nghiên cứu & phát triển thú vị. Nghiên cứu về đồ thị đã phát triển nhanh chóng trong những năm gần đây, nhờ vào sự sẵn có của các tập dữ liệu lớn, tài nguyên điện toán mạnh mẽ, & những tiến bộ trong ML & AI. Do đó, có thể liệt kê 4 nhóm kỹ thuật học đồ thị nổi bật [1]:

1. **Xử lý tín hiệu đồ thị**, áp dụng các phương pháp xử lý tín hiệu truyền thống vào đồ thị, ví dụ: biến đổi Fourier đồ thị & phân tích phổ. Các kỹ thuật này tiết lộ các đặc tính nội tại của đồ thị, ví dụ: tính kết nối & cấu trúc của nó.
2. **Phân tích ma trận**, tìm cách tìm ra các biểu diễn chiều thấp của các ma trận lớn. Mục tiêu của phân tích ma trận: xác định các yếu tố hoặc mô hình tiềm ẩn giải thích các mối quan hệ quan sát được trong ma trận gốc. Phương pháp này có thể cung cấp 1 biểu diễn dữ liệu & có thể diễn giải được.
3. **Bước ngẫu nhiên**, đề cập đến 1 khái niệm toán học được sử dụng để mô hình hóa chuyển động của các thực thể trong đồ thị. Bằng cách mô phỏng các bước ngẫu nhiên trên đồ thị, thông tin về mối quan hệ giữa các nút có thể được thu thập. Đây là lý do tại sao chúng thường được sử dụng để tạo dữ liệu huấn luyện cho các mô hình ML.
4. **DL**, 1 lĩnh vực con của ML tập trung vào các mạng nơ-ron nhiều lớp. Các phương pháp DL có thể mã hóa hiệu quả & biểu diễn dữ liệu đồ thị dưới dạng vectơ. Các vectơ này sau đó có thể được sử dụng trong nhiều tác vụ khác nhau với hiệu suất đáng kể.

Important: these techniques are not mutually exclusive & often overlap in their applications. In practice, they are often combined to form hybrid models that leverage strengths of each. E.g., matrix factorization & DL techniques might be used in combination to learn low-dimensional representations of graph-structured data.

– Điều quan trọng cần lưu ý: các kỹ thuật này không loại trừ lẫn nhau & thường chồng chéo trong ứng dụng. Trong thực tế, chúng thường được kết hợp để tạo thành các mô hình lai tận dụng thế mạnh của từng kỹ thuật. Ví dụ: kỹ thuật phân tích ma trận & DL có thể được sử dụng kết hợp để học các biểu diễn dữ liệu có cấu trúc đồ thị ít chiều.

As delve into world of graph learning, crucial to understand fundamental building block of any ML technique: dataset. Traditional tabular datasets, e.g. spreadsheets, represent data as rows & columns with each row representing a single data point. However, in many real-world scenarios, relationships between data points are just as meaningful as data points themselves. This is where graph datasets come in. Graph datasets represent data points as nodes in a graph & relationships between those data points as edges.

Fig. 1.3: Family tree as a tabular dataset vs. a graph dataset: this dataset represents information about 5 members of a family. Each member has 3 features (or attributes): name, age, & gender. However, tabular version of this dataset does not show connections between these people. On contrary, graph version represents them with edges, which allows us to understand relationships in this family. In many contexts, connections between nodes are crucial in understanding data, which is why representing data in graph form is becoming increasingly popular.

– Hình 1.3: Cây phả hệ dưới dạng tập dữ liệu bảng so với tập dữ liệu đồ thị: tập dữ liệu này biểu diễn thông tin về 5 thành viên trong 1 gia đình. Mỗi thành viên có 3 đặc điểm (hoặc thuộc tính): tên, tuổi, & giới tính. Tuy nhiên, phiên bản bảng của tập dữ liệu này không thể hiện mối liên hệ giữa những người này. Ngược lại, phiên bản đồ thị biểu diễn họ bằng các cạnh, cho phép chúng ta hiểu các mối quan hệ trong gia đình này. Trong nhiều bối cảnh, mối liên hệ giữa các nút rất quan trọng để hiểu dữ liệu, đó là lý do tại sao việc biểu diễn dữ liệu dưới dạng đồ thị ngày càng trở nên phổ biến.

Now have a basic understanding of graph ML & different types of tasks it involves, can move on to exploring 1 of most important approaches for solving these tasks: GNNs.

- o 1.3. Why GNNs? In this book, focus on DL family of graph learning techniques, often referred to as graph neural networks. GNNs are a new category of DL architecture & are specifically designed for graph-structured data. Unlike traditional DL algorithms, which have been primarily developed for text & images, GNNs are explicitly made to process & analyze graph dataset Fig. 1.4: High-level architecture of a GNN pipeline, with a graph as input & an output that corresponds to a given task: a. Node classification. b. Link prediction. c. Graph classification.

– Trong cuốn sách này, chúng tôi tập trung vào họ kỹ thuật học đồ thị DL, thường được gọi là mạng nơ-ron đồ thị. GNN là 1 loại kiến trúc DL mới & được thiết kế riêng cho dữ liệu có cấu trúc đồ thị. Không giống như các thuật toán DL truyền thống, vốn chủ yếu được phát triển cho văn bản & hình ảnh, GNN được thiết kế rõ ràng để xử lý & phân tích tập dữ liệu đồ thị Hình 1.4: Kiến trúc cấp cao của 1 đường ống GNN, với đồ thị làm đầu vào & đầu ra tương ứng với 1 tác vụ nhất định: a. Phân loại nút. b. Dự đoán liên kết. c. Phân loại đồ thị.

GNNs have emerged as a powerful tool for graph learning & have shown excellent results in various tasks & industries. 1 of most striking examples is how a CNN model identified a new antibiotic [2]. Model was trained on 2500 molecules & was tested on a library of 6000 compounds. It predicted that a molecule called halicin should be able to kill many antibiotic-resistant bacteria while having low toxicity to human cells. Based on this prediction, researchers used halicin to treat mice infected with antibiotic-resistant bacteria. They demonstrated its effectiveness & believe model could be used to design new drugs.

– Mạng nơ-ron nhân tạo (GNN) đã nổi lên như 1 công cụ mạnh mẽ cho việc học đồ thị & đã cho thấy kết quả xuất sắc trong nhiều nhiệm vụ & ngành công nghiệp. 1 trong những ví dụ nổi bật nhất là cách 1 mô hình CNN xác định 1 loại kháng sinh mới [2]. Mô hình được huấn luyện trên 2500 phân tử & đã được thử nghiệm trên thư viện gồm 6000 hợp chất. Nó dự đoán rằng 1 phân tử có tên là halicin có thể tiêu diệt nhiều loại vi khuẩn kháng kháng sinh trong khi có độc tính thấp đối với tế bào người. Dựa trên dự đoán này, các nhà nghiên cứu đã sử dụng halicin để điều trị cho chuột bị nhiễm vi khuẩn kháng kháng sinh. Họ đã chứng minh hiệu quả của nó & tin rằng mô hình có thể được sử dụng để thiết kế các loại thuốc mới.

How do GNNs work? Take example of a node classification task in a social network, like previous family tree. In a node classification task, GNNs take advantage of information from different sources to create a vector representation of each node in graph. This representation encompasses not only original node features (e.g. name, age, & gender) but also information from edge features (e.g. strength of relationships between nodes) & global features (e.g. network-wide statistics).

– Mạng GNN hoạt động như thế nào? Hãy lấy ví dụ về tác vụ phân loại nút trong mạng xã hội, chẳng hạn như cây phả hệ trước đây. Trong tác vụ phân loại nút, GNN tận dụng thông tin từ các nguồn khác nhau để tạo biểu diễn vectơ cho mỗi nút trong đồ thị. Biểu diễn này không chỉ bao gồm các đặc điểm nút gốc (ví dụ: tên, tuổi, & giới tính) mà còn bao gồm thông tin từ các đặc điểm biên (ví dụ: cường độ mối quan hệ giữa các nút) & các đặc điểm toàn cục (ví dụ: thống kê toàn mạng).

This is why GNNs are more efficient than traditional ML techniques on graphs. Instead of being limited to original attributes, GNNs enrich original node features with attributes from neighboring nodes, edges, & global features, making representation much more comprehensive & meaningful. New node representations are then used to perform a specific task, e.g. node classification, regression, or link prediction.

– Đây là lý do tại sao GNN hiệu quả hơn các kỹ thuật ML truyền thống trên đồ thị. Thay vì bị giới hạn ở các thuộc tính gốc, GNN làm giàu các đặc trưng nút gốc bằng các thuộc tính từ các nút lân cận, cạnh, & đặc trưng toàn cục, giúp biểu diễn trở nên toàn diện hơn & có ý nghĩa hơn nhiều. Các biểu diễn nút mới sau đó được sử dụng để thực hiện 1 tác vụ cụ thể, ví dụ: phân loại nút, hồi quy hoặc dự đoán liên kết.

Specifically, GNNs define a graph convolution operation that aggregates information from neighboring nodes & edges to update node representation. This operation is performed iteratively, allowing model to learn more complex relationships between nodes as number of iterations increases. E.g., Fig. 1.5: Input graph vs. computation graph representing how a GNN computes representation of node 5 based on its neighbors. shows how a GNN would calculate representation of node 5 using neighboring nodes.

– Cụ thể, GNN định nghĩa 1 phép toán tích chập đồ thị tổng hợp thông tin từ các nút & cạnh lân cận để cập nhật biểu diễn nút. Phép toán này được thực hiện lặp đi lặp lại, cho phép mô hình học các mối quan hệ phức tạp hơn giữa các nút khi số lần lặp tăng lên. Ví dụ: Hình 1.5: Đồ thị đầu vào so với đồ thị tính toán biểu diễn cách GNN tính toán biểu diễn của nút 5 dựa trên các nút lân cận. cho thấy cách GNN tính toán biểu diễn của nút 5 bằng cách sử dụng các nút lân cận.

Worth noting: Fig. 1.5 provides a simplified illustration of a computation graph. In reality, there are various kinds of GNNs & GNN layers, each of which has a unique structure & way of aggregating information from neighboring nodes. These different variants of GNNs also have their own advantages & limitations & are well-suited for specific types of graph data & tasks.

When selecting appropriate GNN architecture for a particular problem, crucial to understand characteristics of graph data & desired outcome.

– Lưu ý: Hình 1.5 cung cấp 1 minh họa đơn giản về đồ thị tính toán. Trên thực tế, có nhiều loại GNN & các lớp GNN, mỗi loại có 1 cấu trúc riêng & cách tổng hợp thông tin từ các nút lân cận. Các biến thể GNN khác nhau này cũng có những ưu điểm & hạn chế riêng & phù hợp với các loại dữ liệu đồ thị & tác vụ cụ thể. Khi lựa chọn kiến trúc GNN phù hợp cho 1 bài toán cụ thể, điều quan trọng là phải hiểu các đặc điểm của dữ liệu đồ thị & kết quả mong muốn.

More generally, GNNs, like other DL techniques, are most effective when applied to specific problems. These problems are characterized by high complexity, i.e., learning good representations is critical to solving task at hand. E.g., a highly complex task could be recommending right products among billions of options to millions of customers. On other hand, some problems, e.g. finding youngest member of our family tree, can be solved without any ML technique.

– Nói chung, GNN, giống như các kỹ thuật DL khác, hiệu quả nhất khi được áp dụng cho các vấn đề cụ thể. Những vấn đề này được đặc trưng bởi độ phức tạp cao, tức là việc học các biểu diễn tốt là rất quan trọng để giải quyết nhiệm vụ. E.g., 1 nhiệm vụ cực kỳ phức tạp có thể là đề xuất sản phẩm phù hợp giữa hàng tỷ lựa chọn cho hàng triệu khách hàng. Mặt khác, 1 số vấn đề, ví dụ như tìm thành viên trẻ nhất trong cây phả hệ gia đình, có thể được giải quyết mà không cần bất kỳ kỹ thuật ML nào.

Furthermore, GNNs require a substantial amount of data to perform effectively. Traditional ML techniques might be a better fit in cases where dataset is small, as they are less reliant on large amounts of data. However, these techniques do not scale as well as GNNs. GNNs can process bigger datasets thanks to parallel & distributed training. They can also exploit additional information more efficiently, which produces better results.

– Hơn nữa, GNN cần 1 lượng dữ liệu đáng kể để hoạt động hiệu quả. Các kỹ thuật ML truyền thống có thể phù hợp hơn trong trường hợp tập dữ liệu nhỏ, vì chúng ít phụ thuộc vào lượng dữ liệu lớn. Tuy nhiên, các kỹ thuật này không có khả năng mở rộng tốt như GNN. GNN có thể xử lý các tập dữ liệu lớn hơn nhờ đào tạo song song và phân tán. Chúng cũng có thể khai thác thông tin bổ sung hiệu quả hơn, mang lại kết quả tốt hơn.

- o **Summary.** In this chap, answered 3 main questions: why graphs, why graph learning, & why GNNs? 1st, explored versatility of graphs in representing various data types, e.g. social networks & transportation networks, but also text & images. Discussed different applications of graph learning, including node classification & graph classification, & highlighted 4 main families of graph learning techniques. Finally, emphasized significance of GNNs & their superiority over other techniques, especially regarding large, complex datasets. By answering these 3 main questions, aimed to provide a comprehensive overview of importance of GNNs & why they are becoming vital tools in ML.

– Trong chương này, chúng tôi đã trả lời 3 câu hỏi chính: tại sao lại dùng đồ thị, tại sao lại dùng học đồ thị, và tại sao lại dùng mạng lưới thần kinh nhân tạo (GNN)? Đầu tiên, chúng tôi khám phá tính linh hoạt của đồ thị trong việc biểu diễn các loại dữ liệu khác nhau, ví dụ như mạng xã hội và mạng lưới giao thông, cũng như văn bản và hình ảnh. Chúng tôi đã thảo luận về các ứng dụng khác nhau của học đồ thị, bao gồm phân loại nút và phân loại đồ thị, và làm nổi bật 4 nhóm kỹ thuật học đồ thị chính. Cuối cùng, chúng tôi nhấn mạnh tầm quan trọng của mạng lưới thần kinh nhân tạo (GNN) và sự vượt trội của chúng so với các kỹ thuật khác, đặc biệt là đối với các tập dữ liệu lớn và phức tạp. Bằng cách trả lời 3 câu hỏi chính này, chúng tôi mong muốn cung cấp 1 cái nhìn tổng quan toàn diện về tầm quan trọng của mạng lưới thần kinh nhân tạo (GNN) và lý do tại sao chúng đang trở thành những công cụ thiết yếu trong học máy.

In Chap. 2: Graph Theory for GNNs, dive deeper into basics of graph theory, which provides foundation for understanding GNNs. This chap will cover fundamental concepts of graph theory, including concepts e.g. adjacency matrices & degrees. Additionally, delve into different types of graphs & their applications, e.g. directed & undirected graphs, & weighted & unweighted graphs.

– Trong Chương 2: Lý thuyết Đồ thị cho Mạng Lưới Hàm Truyền Thống (GNN), tìm hiểu sâu hơn về những kiến thức cơ bản của lý thuyết đồ thị, vốn là nền tảng để hiểu về GNN. Chương này sẽ đề cập đến các khái niệm cơ bản của lý thuyết đồ thị, bao gồm các khái niệm như ma trận kề & bậc. Ngoài ra, tìm hiểu sâu hơn về các loại đồ thị khác nhau & ứng dụng của chúng, ví dụ như đồ thị có hướng & vô hướng, & trọng số & không trọng số.

- **2. Graph Theory for Graph Neural Networks.** Graph theory is a fundamental branch of mathematics that deals with study of graphs & networks. A graph is a visual representation of complex data structures that helps us understand relationships between different entities. Graph theory provides us with tools to model & analyze a vast array of real-world problems, e.g. transportation systems, social networks, & internet connectivity.

– Lý thuyết đồ thị là 1 nhánh cơ bản của toán học, chuyên nghiên cứu về đồ thị & mạng. Đồ thị là 1 biểu diễn trực quan của các cấu trúc dữ liệu phức tạp, giúp chúng ta hiểu được mối quan hệ giữa các thực thể khác nhau. Lý thuyết đồ thị cung cấp cho chúng ta các công cụ để mô hình hóa & phân tích 1 loạt các vấn đề thực tế, ví dụ như hệ thống giao thông, mạng xã hội, & kết nối internet.

In this chap, delve into essentials of graph theory, covering 3 main topics: graph properties, graph concepts, & graph algorithms. Begin by defining graphs & their components. Then introduce different types of graphs & explain their properties & applications. Next, cover fundamental graph concepts, objects, & measures, including adjacency matrix. Finally, dive into graph algorithms, focusing on 2 fundamental algorithms, BFS & DFS. By end of this chap, have a solid foundation in graph theory, allowing you to tackle more advanced topics & design GNNs. In this chap, cover 3 main topics: introducing graph properties, discovering graph concepts, exploring graph algorithms.

– Trong chương này, chúng ta sẽ đi sâu vào những kiến thức cốt lõi của lý thuyết đồ thị, bao gồm 3 chủ đề chính: tính chất đồ thị, khái niệm đồ thị và thuật toán đồ thị. Bắt đầu bằng việc định nghĩa đồ thị và các thành phần của chúng. Sau đó, giới

thiệu các loại đồ thị khác nhau và giải thích tính chất cũng như ứng dụng của chúng. Tiếp theo, chúng ta sẽ tìm hiểu các khái niệm cơ bản về đồ thị, đối tượng, độ đo, bao gồm cả ma trận kề. Cuối cùng, chúng ta sẽ đi sâu vào các thuật toán đồ thị, tập trung vào 2 thuật toán cơ bản: BFS và DFS. Kết thúc chương này, bạn sẽ có nền tảng vững chắc về lý thuyết đồ thị, cho phép bạn giải quyết các chủ đề nâng cao hơn và thiết kế mạng nơ-ron nhân tạo (GNN). Trong chương này, chúng ta sẽ tìm hiểu 3 chủ đề chính: giới thiệu tính chất đồ thị, khám phá các khái niệm đồ thị và tìm hiểu thuật toán đồ thị.

- o **2.1. Introducing graph properties.** In graph theory, a graph is a mathematical structure consisting of a set of objects, called *vertices* or *nodes*, & a set of connections, called *edges*, which link pairs of vertices. Notation $G = (V, E)$ is used to represent a graph, where G is graph, V : set of vertices, E : set of edges. Nodes of a graph can represent any objects, e.g. cities, people, web pages, or molecules, & edges represent relationships or connections between them, e.g. physical roads, social relationships, hyperlinks, or chemical bonds. This sect provides an overview of fundamental graph properties used extensively in later chaps.

– Giới thiệu các tính chất của đồ thị. Trong lý thuyết đồ thị, đồ thị là 1 cấu trúc toán học bao gồm 1 tập hợp các đối tượng, được gọi là *đỉnh* hoặc *nút*, & 1 tập hợp các kết nối, được gọi là *cạnh*, nối các cặp đỉnh. Ký hiệu $G = (V, E)$ được sử dụng để biểu diễn 1 đồ thị, trong đó G là đồ thị, V : tập hợp các đỉnh, E : tập hợp các cạnh. Các nút của đồ thị có thể biểu diễn bất kỳ đối tượng nào, ví dụ: thành phố, con người, trang web hoặc phân tử, & các cạnh biểu diễn các mối quan hệ hoặc kết nối giữa chúng, ví dụ: đường xá thực tế, mối quan hệ xã hội, siêu liên kết hoặc liên kết hóa học. Phần này cung cấp tổng quan về các tính chất cơ bản của đồ thị được sử dụng rộng rãi trong các chương sau.

- o **Directed graphs.** 1 of most basic properties of a graph is whether it is directed or undirected. In a *directed graph*, also called a *digraph*, each edge has a direction or orientation, i.e., edge connects 2 nodes in a particular direction, where 1 node is source & other is destination. In contrast, an undirected graph has undirected edges, where edges have no direction, i.e., edge between 2 vertices can be traversed in either direction, & order in which we visit nodes does not matter. In Python, can use **networkx** library to define an undirected graph as follows with **nx.Graph()**:

– 1 trong những đặc điểm cơ bản nhất của đồ thị là nó có hướng hay vô hướng. Trong 1 đồ thị có hướng, còn được gọi là đồ thị có hướng, mỗi cạnh có 1 hướng hoặc hướng, tức là cạnh nối 2 nút theo 1 hướng cụ thể, trong đó 1 nút là nguồn & nút còn lại là đích. Ngược lại, 1 đồ thị vô hướng có các cạnh vô hướng, trong đó các cạnh không có hướng, tức là cạnh giữa 2 đỉnh có thể được duyệt theo cả hai hướng, & thứ tự chúng ta duyệt các nút không quan trọng. Trong Python, có thể sử dụng thư viện **networkx** để định nghĩa 1 đồ thị vô hướng như sau với **nx.Graph()**:

```
import networkx as nx
# define an undirected graph
G = nx.Graph()
G.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'F'), ('C', 'G')])
```

Code to create a directed graph is similar; simply replace **nx.Graph()** with **nx.DiGraph()**:

```
# define a directed graph
DG = nx.DiGraph()
DG.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'F'), ('C', 'G')])
```

In directed graphs, edges are typically represented using arrows to denote their orientation.

- o **Weighted graphs.** Another important property of graphs is whether edges are weighted or unweighted. In a *weighted graph*, each edge has a weight or cost associated with it. These weights can represent various factors, e.g. distance, travel time, or cost.

– 1 thuộc tính quan trọng khác của đồ thị là các cạnh có trọng số hay không có trọng số. Trong 1 đồ thị có trọng số, mỗi cạnh đều có trọng số hoặc chi phí liên quan. Các trọng số này có thể biểu thị nhiều yếu tố khác nhau, ví dụ: khoảng cách, thời gian di chuyển hoặc chi phí.

E.g., in a transportation network, weights of edges might represent distances between different cities or time it takes to travel between them. In contrast, unweighted graphs have no weight associated with their edges. These types of graphs are commonly used in situations where relationships between nodes are binary, & edges simply indicate presence or absence of a connection between them.

– E.g., trong mạng lưới giao thông, trọng số của các cạnh có thể biểu thị khoảng cách giữa các thành phố khác nhau hoặc thời gian di chuyển giữa chúng. Ngược lại, đồ thị không trọng số không gắn trọng số với các cạnh của chúng. Các loại đồ thị này thường được sử dụng trong các trường hợp mối quan hệ giữa các nút là nhị phân, & cạnh chỉ đơn giản biểu thị sự có hoặc không có kết nối giữa chúng.

Can modify previous undirected graph to add weights to our edges. In **networkx**, edges of graph are defined with a tuple containing start & end nodes & a dictionary specifying edge's weight:

```
# define an weighted graph
WG = nx.Graph()
WG.add_edges_from([('A', 'B', {"weight": 10}), ('A', 'C', {"weight": 20}), ('B', 'D', {"weight": 30}), ('B', 'E', {"weight": 40})])
labels = nx.get_edge_attributes(WG, "weight")
```

- **Connected graphs.** Graph connectivity is a fundamental concept in graph theory that is closely related to graph's structure & function.

– Kết nối đồ thị là 1 khái niệm cơ bản trong lý thuyết đồ thị có liên quan chặt chẽ đến cấu trúc & chức năng của đồ thị.

In a connected graph, there is a path between any 2 vertices in graph. Formally, a graph G is connected iff for every pair of vertices u, v in G , there exists a path from u to v . In contrast, a graph is disconnected if it is not connected, i.e., at least 2 vertices are not connected by a path.

– Trong 1 đồ thị liên thông, luôn có 1 đường đi giữa hai đỉnh bất kỳ trong đồ thị. Về mặt hình thức, 1 đồ thị G được coi là liên thông nếu và chỉ nếu (iff) với mọi cặp đỉnh u, v trong G , tồn tại 1 đường đi từ u đến v . Ngược lại, 1 đồ thị được coi là không liên thông nếu nó không liên thông, tức là có ít nhất 2 đỉnh không được nối với nhau bằng 1 đường đi.

networkx library provides a built-in function for verifying whether a graph is connected or not. In following example, 1st graph contains isolated nodes (4 & 5), unlike 2nd graph.

– Thư viện **networkx** cung cấp 1 hàm tích hợp để kiểm tra xem đồ thị có kết nối hay không. Trong ví dụ sau, đồ thị thứ nhất chứa các nút riêng biệt (4 & 5), không giống như đồ thị thứ 2.

```
# verify whether a graph is connected or disconnected
G1 = nx.Graph()
G1.add_edges_from([(1, 2), (2, 3), (3, 1), (4, 5)])
print(f"Is graph 1 connected? {nx.is_connected(G1)}")
```

```
G2 = nx.Graph()
G2.add_edges_from([(1, 2), (2, 3), (3, 1), (1, 4)])
print(f"Is graph 2 connected? {nx.is_connected(G2)}")
```

This property is easy to visualize with small graphs. Connected graphs have several interesting properties & applications. E.g., in a communication network, a connected graph ensures that any 2 nodes can communicate with each other through a path. In contrast, disconnected graphs can have isolated nodes that cannot communicate with other nodes in network, making it challenging to design efficient routing algorithms.

– Tính chất này dễ hình dung bằng các đồ thị nhỏ. Đồ thị liên thông có 1 số tính chất & ứng dụng thú vị. Ví dụ, trong mạng truyền thông, 1 đồ thị liên thông đảm bảo rằng bất kỳ 2 nút nào cũng có thể giao tiếp với nhau thông qua 1 đường dẫn. Ngược lại, đồ thị liên thông có thể có các nút bị cô lập không thể giao tiếp với các nút khác trong mạng, khiến việc thiết kế các thuật toán định tuyến hiệu quả trở nên khó khăn.

There are different ways to measure connectivity of a graph. 1 of most common measures is minimum number of edges that need to be removed to disconnect graph, which is known as graph's minimum cut. Minimum cut problem has several applications in network flow optimization, clustering, & community detection.

– Có nhiều cách khác nhau để đo lường khả năng kết nối của 1 đồ thị. 1 trong những phép đo phổ biến nhất là số cạnh tối thiểu cần loại bỏ để ngắt kết nối đồ thị, được gọi là đường cắt tối thiểu của đồ thị. Bài toán đường cắt tối thiểu có nhiều ứng dụng trong tối ưu hóa luồng mạng, phân cụm, & phát hiện cộng đồng.

- **Types of graphs.** In addition to commonly used graph types, there are some special types of graphs that have unique properties & characteristics:

1. A tree is a connected, undirected graph with no cycles. Since there is only 1 path between any 2 nodes in a tree, a tree is a special case of a graph. Trees are often used to model hierarchical structures, e.g. family trees, organizational structures, or classification trees.
2. A rooted tree is a tree in which 1 node is designated as root, & all other vertices are connected to it by a unique path. Rooted trees are often used in CS to represent hierarchical data structures, e.g. filesystems or structures of XML documents.
3. A directed acyclic graph (DAG) is a directed graph that has no cycles, i.e., edges can only be traversed in a particular direction, & there are no loops or cycles. DAGs are often used to model dependencies between tasks or events – e.g., in project management or in computing critical path of a job.
4. A bipartite graph is a graph in which vertices can be divided into 2 disjoint sets, s.t. all edges connect vertices in different sets. Bipartite graphs are often used in mathematics & CS to model relationships between 2 different types of objects, e.g. buyers & sellers, or employees & projects.
5. A complete graph is a graph in which every pair of vertices is connected by an edge. Complete graphs are often used in combinatorics to model problems involving all possible pairwise connections, & in computer networks to model fully connected networks.

Now have reviewed essential types of graphs, explore some of most important graph objects. Understanding these concepts will help analyze & manipulate graphs effectively.

– Các loại đồ thị. Ngoài các loại đồ thị thường dùng, còn có 1 số loại đồ thị đặc biệt với các thuộc tính & đặc điểm riêng:

1. Cây là đồ thị liên thông, vô hướng và không có chu trình. Vì chỉ có 1 đường đi giữa 2 nút bất kỳ trong cây, nên cây là 1 trường hợp đặc biệt của đồ thị. Cây thường được sử dụng để mô hình hóa các cấu trúc phân cấp, ví dụ: cây phả hệ, cấu trúc tổ chức hoặc cây phân loại.

2. Cây có gốc là cây trong đó 1 nút được chỉ định là gốc, & tất cả các đỉnh khác được kết nối với nó bằng 1 đường đi duy nhất. Cây có gốc thường được sử dụng trong CS để biểu diễn các cấu trúc dữ liệu phân cấp, ví dụ: hệ thống tệp hoặc cấu trúc của tài liệu XML.
3. Đồ thị có hướng phi chu trình (DAG) là đồ thị có hướng không có chu trình, tức là các cạnh chỉ có thể được duyệt theo 1 hướng cụ thể, & không có vòng lặp hoặc chu trình. DAG thường được sử dụng để mô hình hóa sự phụ thuộc giữa các tác vụ hoặc sự kiện – ví dụ: trong quản lý dự án hoặc tính toán đường dẫn tới hạn của 1 công việc.
4. Đồ thị hai phía là đồ thị trong đó các đỉnh có thể được chia thành 2 tập rời rạc, nghĩa là tất cả các cạnh đều nối các đỉnh thuộc các tập khác nhau. Đồ thị hai phía thường được sử dụng trong toán học & Khoa học Máy tính để mô hình hóa mối quan hệ giữa 2 loại đối tượng khác nhau, ví dụ: người mua & người bán, hoặc nhân viên & dự án.
5. Đồ thị đầy đủ là đồ thị trong đó mỗi cặp đỉnh được nối bởi 1 cạnh. Đồ thị đầy đủ thường được sử dụng trong tổ hợp để mô hình hóa các bài toán liên quan đến tất cả các kết nối từng cặp có thể có, & trong mạng máy tính để mô hình hóa các mạng được kết nối đầy đủ.

Bây giờ chúng ta đã xem xét các loại đồ thị thiết yếu, khám phá 1 số đối tượng đồ thị quan trọng nhất. Việc hiểu các khái niệm này sẽ giúp phân tích & thao tác đồ thị 1 cách hiệu quả.

- **Discovering graph concepts.** In this sect, explore some of essential concepts in graph theory, including graph objects (e.g. degree & neighbors), graph measures (e.g., centrality & density), & adjacency matrix representation.
 - **Khám phá các khái niệm về đồ thị.** Trong phần này, khám phá 1 số khái niệm thiết yếu trong lý thuyết đồ thị, bao gồm các đối tượng đồ thị (ví dụ: bậc & lân cận), các phép đo đồ thị (ví dụ: độ trung tâm & mật độ), & biểu diễn ma trận kề.

- * **Fundamental objects.** 1 of key concepts in graph theory is degree of a node, which is number of edges incident to this node. An edge is said to be incident on a node if that node is 1 of edge's endpoints. Degree of a node v is often denoted by $\deg v$. It can be defined for both directed & undirected graphs:

- In an undirected graph, degree of a vertex is number of edges connected to it. If node is connected to itself (called a loop, or self-loop), it adds 2 to degree.
- In a directed graph, degree is divided into 2 types: indegree & outdegree. Indegree (denoted by $\deg_-(v)$) of a node represents number of edges that point towards that node, while outdegree (denoted by $\deg^+(v)$) represents number of edges that start from that node. In this case, a self-loop adds 1 to indegree & to outdegree.

Indegree & outdegree are essential for analyzing & understanding directed graphs, as they provide insight into how information or resources are distributed within graph. E.g., nodes with high indegree are likely to be important sources of information or resources. In contrast, nodes with high outdegree are likely to be important destinations or consumers of information or resources.

- **Các đối tượng cơ bản.** 1 trong những khái niệm chính trong lý thuyết đồ thị là bậc của 1 nút, tức là số cạnh liên thuộc nút này. 1 cạnh được gọi là liên thuộc nút nếu nút đó là 1 trong các điểm cuối của cạnh đó. Bậc của 1 nút v thường được ký hiệu là $\deg v$. Nó có thể được định nghĩa cho cả đồ thị có hướng & vô hướng:

- Trong đồ thị vô hướng, bậc của 1 đỉnh là số cạnh được nối với nó. Nếu nút được nối với chính nó (gọi là vòng lặp, hoặc tự vòng lặp), nó cộng thêm 2 vào bậc.
- Trong đồ thị có hướng, bậc được chia thành 2 loại: bậc vào & bậc ra. Bậc vào (ký hiệu là $\deg_-(v)$) của 1 nút biểu diễn số cạnh hướng đến nút đó, trong khi bậc ra (ký hiệu là $\deg^+(v)$) biểu diễn số cạnh bắt đầu từ nút đó. Trong trường hợp này, 1 vòng lặp tự thêm 1 vào bậc vào & vào bậc ra.

Bậc vào & bậc ra rất cần thiết để phân tích & hiểu đồ thị có hướng, vì chúng cung cấp cái nhìn sâu sắc về cách thông tin hoặc tài nguyên được phân bổ trong đồ thị. Ví dụ: các nút có bậc vào cao có thể là nguồn thông tin hoặc tài nguyên quan trọng. Ngược lại, các nút có bậc ra cao có thể là đích đến hoặc người tiêu thụ thông tin hoặc tài nguyên quan trọng. In **networkx**, can simply calculate node degree, indegree, or outdegree using built-in methods:

- Trong **networkx**, có thể dễ dàng tính toán bậc nút, bậc vào hoặc bậc ra bằng các phương pháp tích hợp:

```
# calculate node degree, indegree, or outdegree
G = nx.Graph()
G.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'F'), ('C', 'G')])
print(f"deg(A) = {G.degree['A']}")

DG = nx.DiGraph()
DG.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'F'), ('C', 'G')])
print(f"deg^-(A) = {DG.in_degree['A']}")
print(f"deg^+(A) = {DG.out_degree['A']}")
```

Concept of node degree is related to that of neighbors. Neighbors refer to nodes directly connected to a particular node through an edge. Moreover, 2 nodes are said to be adjacent if they share at least 1 common neighbor. Concepts of neighbors & adjacency are fundamental to many graph algorithms & applications, e.g. searching for a path between 2 nodes or identifying clusters in a network.

- **Khái niệm bậc nút liên quan đến khái niệm lân cận.** Lân cận là các nút được kết nối trực tiếp với 1 nút cụ thể thông qua 1 cạnh. Hơn nữa, 2 nút được gọi là lân cận nếu chúng có chung ít nhất 1 lân cận. Khái niệm lân cận & kề là nền tảng cho nhiều thuật toán đồ thị & ứng dụng, ví dụ: tìm kiếm đường đi giữa 2 nút hoặc xác định các cụm trong mạng.

In graph theory, a path is a sequence of edges that connects 2 nodes (or more) in a graph. Length of a path is number of edges that are traversed along path. There are different types of paths, but 2 of them are particularly important:

1. A simple path is a path that does not visit any node more than once, except for start & end vertices.
2. A cycle is a path in which 1st & last vertices are same. A graph is said to be acyclic if it contains no cycles, e.g. trees & DAGs.

Degrees & paths can be used to determine importance of a node in a network. This measure is referred to as *centrality*.

– Trong lý thuyết đồ thị, đường đi là 1 chuỗi các cạnh nối 2 nút (hoặc nhiều hơn) trong 1 đồ thị. Độ dài của đường đi là số cạnh được đi qua dọc theo đường đi. Có nhiều loại đường đi khác nhau, nhưng có 2 loại đặc biệt quan trọng:

1. Đường đi đơn là đường đi không đi qua bất kỳ nút nào quá 1 lần, ngoại trừ đỉnh đầu & đỉnh cuối.
2. Chu trình là đường đi mà đỉnh đầu & đỉnh cuối giống nhau. 1 đồ thị được gọi là chu trình nếu nó không chứa chu trình nào, ví dụ: cây & DAG.

Bậc & đường đi có thể được sử dụng để xác định tầm quan trọng của 1 nút trong mạng. Thước đo này được gọi là độ trung tâm.

* **Graph measures.** Centrality quantifies importance of a vertex or node in a network. It helps us to identify key node in a graph based on their connectivity & influence on flow of information or interactions within network. There are several measures of centrality, each providing a different perspective on importance of a node:

1. **Degree centrality** is 1 of simplest & most commonly used measures of centrality. It is simply defined as degree of node. A high degree centrality indicates a vertex is highly connected to other vertices in graph, & thus significantly influences network.
2. **Closeness centrality** measures how close a node is to all other nodes in graph. It corresponds to average length of shortest path between target node & all other nodes in graph. A node with high closeness centrality can quickly reach all other vertices in network.
3. **Betweenness centrality** measures number of times a node lies on shortest path between pairs of other nodes in graph. A node with high betweenness centrality acts as a bottleneck or bridge between different parts of graph.

Calculate these measure on previous graphs using built-in functions of **networkx** & analyze result:

– Đo lường đồ thị. Độ tập trung định lượng tầm quan trọng của 1 đỉnh hoặc nút trong mạng. Nó giúp chúng ta xác định nút chính trong đồ thị dựa trên khả năng kết nối & ảnh hưởng của chúng đến luồng thông tin hoặc tương tác trong mạng. Có 1 số thước đo độ tập trung, mỗi thước đo cung cấp 1 góc nhìn khác nhau về tầm quan trọng của 1 nút:

1. **Độ tập trung bậc** là 1 trong những thước đo độ tập trung đơn giản nhất & được sử dụng phổ biến nhất. Nó được định nghĩa đơn giản là bậc của nút. Độ tập trung bậc cao cho thấy 1 đỉnh có kết nối cao với các đỉnh khác trong đồ thị, & do đó ảnh hưởng đáng kể đến mạng.
2. **Độ tập trung gần** đo lường mức độ gần của 1 nút với tất cả các nút khác trong đồ thị. Nó tương ứng với độ dài trung bình của đường đi ngắn nhất giữa nút mục tiêu & tất cả các nút khác trong đồ thị. 1 nút có độ tập trung gần cao có thể nhanh chóng tiếp cận tất cả các đỉnh khác trong mạng.
3. **Độ tập trung giữa** đo lường số lần 1 nút nằm trên đường đi ngắn nhất giữa các cặp nút khác trong đồ thị. 1 nút có tính trung tâm cao đóng vai trò như 1 nút thắt cổ chai hoặc cầu nối giữa các phần khác nhau của đồ thị.

Tính toán các phép đo này trên các đồ thị trước đó bằng cách sử dụng các hàm tích hợp của **networkx** & phân tích kết quả:

Graph measures

```
print(f"Degree centrality = {nx.degree_centrality(G)}")
print(f"Closeness centrality = {nx.closeness_centrality(G)}")
print(f"Betweenness centrality = {nx.betweenness_centrality(G)}")
```

Importance of nodes A, B, C in a graph depends on type of centrality used. Degree centrality considers nodes B & C to be more important because they have more neighbors than node A. However, in closeness centrality, node A is most important as it can reach any other node in graph in shortest possible path. On other hand, nodes A, B, & C have equal betweenness centrality, as they all lie on a large number of shortest paths between other nodes.

– Tầm quan trọng của các nút A, B, C trong đồ thị phụ thuộc vào loại trung tâm được sử dụng. Trung tâm bậc coi các nút B & C quan trọng hơn vì chúng có nhiều hàng xóm hơn nút A. Tuy nhiên, về trung tâm gần, nút A quan trọng nhất vì nó có thể đến bất kỳ nút nào khác trong đồ thị theo đường đi ngắn nhất có thể. Mặt khác, các nút A, B, & C có trung tâm nằm giữa bằng nhau, vì tất cả chúng đều nằm trên 1 số lượng lớn các đường đi ngắn nhất giữa các nút khác.

In addition to these measures, see how to calculate importance of a node using ML techniques in next chaps. However, it is not only measure covered. Indeed, *density* is another important measure, indicating how connected graph is. It is a ratio between actual number of edges & maximum possible number of edges in graph. A graph with high density is considered more connected & has more information flow compared to a graph with low density.

– Ngoài các phép đo này, xem cách tính tầm quan trọng của 1 nút bằng kỹ thuật ML trong các chương tiếp theo. Tuy nhiên, đây không chỉ là phép đo được đề cập. Thật vậy, *density* là 1 phép đo quan trọng khác, cho biết mức độ kết nối của đồ thị. Nó là tỷ lệ giữa số cạnh thực tế & số cạnh tối đa có thể có trong đồ thị. 1 đồ thị có mật độ cao được coi là kết nối hơn & có luồng thông tin lớn hơn so với đồ thị có mật độ thấp.

Formula to calculate density depends on whether graph is directed or undirected. For an undirected graph with n nodes, maximum possible number of edges is $\frac{n(n-1)}{2}$. For a directed graph with n nodes, maximum number of edges is $n(n-1)$. Density of a graph is calculated as number of edges divided by maximum number of edges.

– Công thức tính mật độ phụ thuộc vào việc đồ thị có hướng hay vô hướng. Đối với đồ thị vô hướng có n nút, số cạnh tối đa có thể là $\frac{n(n-1)}{2}$. Đối với đồ thị có hướng có n nút, số cạnh tối đa là $n(n-1)$. Mật độ của đồ thị được tính bằng số cạnh chia cho số cạnh tối đa.

A dense graph has a density closer to 1, while a sparse graph has a density closer to 0. There is no strict rule for what constitutes a dense or sparse graph, but generally, a graph is considered dense if its density is > 0.5 & sparse if its density is < 0.1 . This measure is directly connected to a fundamental problem with graphs: how to represent adjacency matrix.

– Đồ thị dày đặc có mật độ gần bằng 1, trong khi đồ thị thưa có mật độ gần bằng 0. Không có quy tắc nghiêm ngặt nào về việc thế nào là đồ thị dày đặc hay thưa thớt, nhưng nhìn chung, 1 đồ thị được coi là dày đặc nếu mật độ của nó $> 0,5$ & thưa nếu mật độ của nó $< 0,1$. Phép đo này liên quan trực tiếp đến 1 vấn đề cơ bản với đồ thị: cách biểu diễn ma trận kề.

* **Adjacency matrix representation.** An adjacency matrix is a matrix that represents edges in a graph, where each cell indicates whether there is an edge between 2 nodes. Adjacency matrix is a square matrix of size $n \times n$, where n : number of nodes in graph. A value of 1 in cell (i, j) indicates that there is an edge between node i & node j , while a value of 0 indicates that there is no edge. For an undirected graph, adjacency matrix is symmetric, while for a directed graph, adjacency matrix is not necessarily symmetric. In Python, it can be implemented as a list of lists:

– **Biểu diễn ma trận kề.** Ma trận kề là 1 ma trận biểu diễn các cạnh trong đồ thị, trong đó mỗi ô biểu thị liệu có cạnh giữa 2 nút hay không. Ma trận kề là 1 ma trận vuông có kích thước $n \times n$, trong đó n : số nút trong đồ thị. Giá trị 1 trong ô (i, j) biểu thị rằng có 1 cạnh giữa nút i & nút j , trong khi giá trị 0 biểu thị rằng không có cạnh nào. Đối với đồ thị vô hướng, ma trận kề là đối xứng, trong khi đối với đồ thị có hướng, ma trận kề không nhất thiết phải đối xứng. Trong Python, ma trận kề có thể được triển khai dưới dạng danh sách các danh sách.

Adjacency matrix is a straightforward representation that can be easily visualized as a 2D array. 1 of key advantages of using an adjacency matrix: checking whether 2 nodes are connected is a constant time operation $O(1)$. This makes it an efficient way to test existence of an edge in graph. Moreover, it is used to perform matrix operations, which are useful for certain graph algorithms, e.g. calculating shortest path between 2 nodes.

– Ma trận kề là 1 biểu diễn đơn giản có thể dễ dàng hình dung dưới dạng 1 mảng 2D. 1 trong những lợi thế chính của việc sử dụng ma trận kề: việc kiểm tra xem 2 nút có được kết nối hay không là 1 phép toán thời gian hằng số $O(1)$. Điều này khiến nó trở thành 1 cách hiệu quả để kiểm tra sự tồn tại của 1 cạnh trong đồ thị. Hơn nữa, nó được sử dụng để thực hiện các phép toán ma trận, hữu ích cho 1 số thuật toán đồ thị, ví dụ: tính toán đường đi ngắn nhất giữa 2 nút.

However, adding or removing nodes can be costly, as matrix needs to be resized or shifted. 1 of main drawbacks of using an adjacency matrix is its space complexity: as number of nodes in graph grows, space required to store adjacency matrix increases exponentially. Formally, say: adjacency matrix has a space complexity of $O(|V|^2)$, where $|V|$ represents number of nodes in graph.

– Tuy nhiên, việc thêm hoặc xóa các nút có thể tốn kém, vì ma trận cần được thay đổi kích thước hoặc dịch chuyển. 1 trong những nhược điểm chính của việc sử dụng ma trận kề là độ phức tạp về không gian: khi số lượng nút trong đồ thị tăng lên, không gian cần thiết để lưu trữ ma trận kề cũng tăng theo cấp số nhân. Ví dụ: ma trận kề có độ phức tạp về không gian là $O(|V|^2)$, trong đó $|V|$ biểu thị số lượng nút trong đồ thị.

Overall, while adjacency matrix is a useful data structure for representing small graphs, it may not be practical for larger ones due to its space complexity. Additionally, overhead of adding or removing nodes can make it inefficient for dynamically changing graphs.

– Nhìn chung, mặc dù ma trận kề là 1 cấu trúc dữ liệu hữu ích để biểu diễn các đồ thị nhỏ, nhưng nó có thể không thực tế đối với các đồ thị lớn hơn do độ phức tạp về không gian. Ngoài ra, việc thêm hoặc bớt các nút có thể khiến ma trận kề không hiệu quả đối với các đồ thị thay đổi động.

This is why other representation can be helpful. E.g., another popular way to store graphs is *edge list*. An edge list is a list of all edges in a graph. Each edge is represented by a tuple or a pair of vertices. Edge list can also include weight or cost of each edge. This is data structure used to create our graphs with **networkx**:

– Đây là lý do tại sao các biểu diễn khác có thể hữu ích. Ví dụ, 1 cách phổ biến khác để lưu trữ đồ thị là **edge list**. Danh sách cạnh là danh sách tất cả các cạnh trong 1 đồ thị. Mỗi cạnh được biểu diễn bằng 1 bộ hoặc 1 cặp đỉnh. Danh sách cạnh cũng có thể bao gồm trọng số hoặc chi phí của mỗi cạnh. Đây là cấu trúc dữ liệu được sử dụng để tạo đồ thị của chúng ta với **networkx**:

```
edge_list = [(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)]
```

When compare both data structures applied to our graph, clear: edge list is less verbose. This is case because our graph is fairly sparse. On other hand, if our graph was complete, would require 21 tuples instead of 6. This is explained by a space complexity of $O(|E|)$, where $|E|$ is number of edges. Edge lists are more efficient for storing sparse graphs, where number of edges is much smaller than number of nodes.

– Khi so sánh cả hai cấu trúc dữ liệu được áp dụng cho đồ thị của chúng ta, rõ ràng: danh sách cạnh ít chi tiết hơn. Điều này là do đồ thị của chúng ta khá thưa thớt. Mặt khác, nếu đồ thị của chúng ta đầy đủ, sẽ cần 21 bộ thay vì 6. Điều này được giải thích bởi độ phức tạp không gian là $O(|E|)$, trong đó $|E|$ là số cạnh. Danh sách cạnh hiệu quả hơn trong việc lưu trữ đồ thị thưa thớt, trong đó số cạnh nhỏ hơn nhiều so với số nút.

However, checking whether 2 vertices are connected in an edge list requires iterating through entire list, which can be time-consuming for large graphs with many edges. Therefore, edge lists are commonly used in applications where space is a concern.

– Tuy nhiên, việc kiểm tra xem 2 đỉnh có được kết nối trong danh sách cạnh hay không đòi hỏi phải lặp lại toàn bộ danh

sách, điều này có thể tốn thời gian đối với các đồ thị lớn có nhiều cạnh. Do đó, danh sách cạnh thường được sử dụng trong các ứng dụng đòi hỏi không gian.

A 3rd & popular representation is *adjacency list*. It consists of a list of pairs, where each pair represents a node in graph & its adjacent nodes. The pairs can be stored in a linked list, dictionary, or other data structures, depending on implementation. An adjacency list has several advantages over an adjacency matrix or an edge list. 1st, space complexity is $O(|V| + |E|)$. This is more efficient than $O(|V|^2)$ space complexity of an adjacency matrix for sparse graphs. 2nd, it allows for efficient iteration through adjacent vertices of a node, which is useful in many graph algorithms. Finally, adding a node or an edge can be done in constant time.

– Biểu diễn phổ biến thứ 3 là *adjacency list*. Nó bao gồm 1 danh sách các cặp, trong đó mỗi cặp biểu diễn 1 nút trong đồ thị & các nút liền kề của nó. Các cặp có thể được lưu trữ trong danh sách liên kết, từ điển hoặc các cấu trúc dữ liệu khác, tùy thuộc vào cách triển khai. Danh sách kề có 1 số ưu điểm so với ma trận kề hoặc danh sách cạnh. Thứ nhất, độ phức tạp không gian là $O(|V| + |E|)$. Điều này hiệu quả hơn độ phức tạp không gian $O(|V|^2)$ của ma trận kề đối với đồ thị thưa. Thứ hai, nó cho phép lặp hiệu quả qua các đỉnh liền kề của 1 nút, điều này hữu ích trong nhiều thuật toán đồ thị. Cuối cùng, việc thêm 1 nút hoặc 1 cạnh có thể được thực hiện trong thời gian không đổi.

However, checking whether 2 vertices are connected can be slower than with an adjacency matrix. This is because it requires iterating through adjacency list of 1 of vertices, which can be time-consuming for large graphs.

– Tuy nhiên, việc kiểm tra xem 2 đỉnh có được kết nối hay không có thể chậm hơn so với ma trận kề. Điều này là do ma trận này yêu cầu phải lặp qua danh sách kề của 1 đỉnh, điều này có thể tốn thời gian đối với các đồ thị lớn.

Each data structure has its own advantages & disadvantages that depend on specific application & requirements. In next sect, process graphs & introduce 2 most fundamental graph algorithms.

– Mỗi cấu trúc dữ liệu đều có ưu điểm & nhược điểm riêng, tùy thuộc vào yêu cầu & ứng dụng cụ thể. Trong phần tiếp theo, đồ thị quy trình & giới thiệu 2 thuật toán đồ thị cơ bản nhất.

- Exploring graph algorithms. Graph algorithms are critical in solving problems related to graphs, e.g. finding shortest path between 2 nodes or detecting cycles. This sect will discuss 2 graph traversal algorithms: BFS & DFS.

– Khám phá thuật toán đồ thị. Thuật toán đồ thị đóng vai trò quan trọng trong việc giải quyết các bài toán liên quan đến đồ thị, ví dụ: tìm đường đi ngắn nhất giữa 2 nút hoặc phát hiện chu trình. Phần này sẽ thảo luận về 2 thuật toán duyệt đồ thị: BFS & DFS.

* Breadth-first search. BFS is a graph traversal algorithm that starts at root node & explores all neighboring nodes at a particular level before moving to next level of nodes. It works by maintaining a queue of nodes to visit & marking each visited node as it is added to queue. Algorithm then dequeues next node in queue & explores all its neighbors, adding them to queue if they have not been visited yet. How can implement BFS in Python:

- Create an empty graph & add edges with `add_edges_from()` method.
- Define a function called `bfs()` that implements BF's algorithm on a graph. Function takes 2 arguments: `graph` object & starting node for search.
- Initialize 2 lists `visited`, `queue` & add starting node. `visited` list keeps track of nodes that have been visited during search, while `queue` list stores nodes that need to be visited.
- Enter a `while` loop that continues until `queue` list is empty. Inside loop, remove 1st node in `queue` list using `pop(0)` method & store result in `node` variable.
- Iterate through neighbors of node using a `for` loop. For each neighbor that has not been visited yet, add it to `visited` list & to end of `queue` list using `append()` method. When it is complete, return `visited` list.
- Call `bfs()` function with `G` argument & 'A' starting node: `bfs(G, 'A')`.
- Function returns list of visited nodes in order in which they were visited.

BFS is particularly useful in finding shortest path between 2 nodes in an unweighted graph. This is because algorithm visits nodes in order of their distance from starting node, so 1st time target node is visited, it must be along shortest path from starting node.

– BFS đặc biệt hữu ích trong việc tìm đường đi ngắn nhất giữa 2 nút trong đồ thị không trọng số. Điều này là do thuật toán sẽ duyệt các nút theo thứ tự khoảng cách từ nút bắt đầu, do đó, lần đầu tiên nút đích được duyệt, nó phải nằm trên đường đi ngắn nhất từ nút bắt đầu.

In addition to finding shortest path, BFS can also be used to check whether a graph is connected or to find all connected components of a graph. It is also used in applications e.g. web crawlers, social network analysis, & shortest path routing in networks.

– Ngoài việc tìm đường đi ngắn nhất, BFS còn có thể được sử dụng để kiểm tra xem 1 đồ thị có liên thông hay không hoặc để tìm tất cả các thành phần liên thông của đồ thị. Nó cũng được sử dụng trong các ứng dụng như trình thu thập dữ liệu web, phân tích mạng xã hội và định tuyến đường đi ngắn nhất trong mạng.

Time complexity of BFS is $O(|V| + |E|)$. This can be a significant issue for graphs with a high degree of connectivity or for graphs that are sparse. Several variants of BFS have been developed to mitigate this issue, e.g. bidirectional BFS & A* search, which use heuristics to reduce number of nodes that need to be explored.

– Độ phức tạp thời gian của BFS là $O(|V| + |E|)$. Điều này có thể là 1 vấn đề đáng kể đối với các đồ thị có mức độ kết nối cao hoặc các đồ thị thưa thớt. 1 số biến thể của BFS đã được phát triển để giảm thiểu vấn đề này, ví dụ: tìm kiếm BFS song hướng & A*, sử dụng các thuật toán tìm kiếm để giảm số lượng nút cần khám phá.

- * Depth-first search. DFS is a recursive algorithm that starts at root node & explores as far as possible along each branch before backtracking. It chooses a node & explores all of its unvisited neighbors, visiting 1st neighbor that has not been

explored & backtracking only when all neighbors have been visited. By doing so, it explores graph by following as deep a path from starting node as possible before backtracking to explore other branches. This continues until all nodes have been explored. Implement DFS in Python:

1. 1st initialize an empty list called **visited**.
2. Define a function called **dfs()** that takes in **visited**, **graph**, **node** as arguments: `def dfs(visited, graph, node):`
3. If current **node** is not in **visited** list, append it to list.
4. Then iterate through each neighbor of current **node**. For each neighbor, recursively call **dfs()** function passing in **visited**, **graph**, **neighbor** as arguments.
5. **dfs()** function continues to explore graph depth-1st, visiting all neighbors of each node until there are no more unvisited neighbors. Finally, **visited** list is returned.

– Tìm kiếm theo chiều sâu. DFS là 1 thuật toán đệ quy bắt đầu từ nút gốc & khám phá càng xa càng tốt dọc theo mỗi nhánh trước khi quay lui. Nó chọn 1 nút & khám phá tất cả các hàng xóm chưa được thăm của nó, thăm hàng xóm thứ nhất chưa được thăm & chỉ quay lui khi tất cả các hàng xóm đã được thăm. Bằng cách này, nó khám phá đồ thị bằng cách đi theo đường dẫn sâu nhất có thể từ nút bắt đầu trước khi quay lui để khám phá các nhánh khác. Quá trình này tiếp tục cho đến khi tất cả các nút đã được khám phá. Triển khai DFS trong Python:

1. Đầu tiên, khởi tạo 1 danh sách rỗng có tên là **visited**.
2. Định nghĩa 1 hàm có tên là **dfs()** nhận **visited**, **graph**, **node** làm đối số: `def dfs(visited, graph, node):`
3. Nếu **node** hiện tại không có trong danh sách **visited**, thêm nó vào danh sách.
4. Sau đó, lặp qua từng hàng xóm của nút hiện tại. Với mỗi hàng xóm, gọi đệ quy hàm **dfs()** truyền vào **visited**, **graph**, **neighbor** làm đối số.
5. Hàm **dfs()** tiếp tục khám phá đồ thị theo chiều sâu 1, duyệt tất cả các hàng xóm của mỗi nút cho đến khi không còn hàng xóm nào chưa được duyệt. Cuối cùng, danh sách **visited** được trả về.

DFS is useful in solving various problems, e.g. finding connected components, topological sorting, & solving maze problems. It is particularly useful in finding cycles in a graph since it traverses graph in a depth-1st order, & a cycle exists iff a node is visited twice during traversal.

– DFS hữu ích trong việc giải quyết nhiều bài toán khác nhau, ví dụ như tìm các thành phần liên thông, sắp xếp tôpô, & giải các bài toán mê cung. Nó đặc biệt hữu ích trong việc tìm chu trình trong đồ thị vì nó duyệt đồ thị theo thứ tự độ sâu 1, & tồn tại 1 chu trình chỉ khi và chỉ khi 1 nút được thăm hai lần trong quá trình duyệt.

Like BFS, it has a time complexity of $O(|V| + |E|)$. It requires less memory but does not guarantee shallowest path solution. Finally, unlike BFS, you can be trapped in infinite loops using DFS.

– Giống như BFS, DFS có độ phức tạp thời gian là $O(|V| + |E|)$. Nó đòi hỏi ít bộ nhớ hơn nhưng không đảm bảo giải pháp đường đi nông nhất. Cuối cùng, không giống như BFS, bạn có thể bị mắc kẹt trong vòng lặp vô hạn khi sử dụng DFS.

Additionally, many other algorithms in graph theory build upon BFS & DFS, e.g. Dijkstra's shortest path algorithm, Kruskal's minimum spanning tree algorithm, & Tarjan's strongly connected components algorithm. Therefore, a solid understanding of BFS & DFS is essential for anyone who wants to work with graphs & develop more advanced graph algorithms.

– Ngoài ra, nhiều thuật toán khác trong lý thuyết đồ thị được xây dựng dựa trên BFS & DFS, ví dụ như thuật toán đường đi ngắn nhất Dijkstra, thuật toán cây bao trùm nhỏ nhất Kruskal, và thuật toán thành phần liên thông mạnh Tarjan. Do đó, việc hiểu rõ về BFS & DFS là điều cần thiết cho bất kỳ ai muốn làm việc với đồ thị & phát triển các thuật toán đồ thị nâng cao hơn.

- o **Summary.** In this chap, covered essentials of graph theory, a branch of mathematics that studies graphs & networks. Began by defining what a graph is & explained different types of graphs, e.g., directed, weighted, & connected graphs. Then introduced fundamental graph objects (including neighbors) & measures (e.g. centrality & density), which are used to understand & analyze graph structures.

– Chương này trình bày những kiến thức cơ bản về lý thuyết đồ thị, 1 nhánh của toán học nghiên cứu đồ thị & mạng lưới. Bắt đầu bằng việc định nghĩa đồ thị & giải thích các loại đồ thị khác nhau, ví dụ: đồ thị có hướng, có trọng số, & liên thông. Sau đó, giới thiệu các đối tượng đồ thị cơ bản (bao gồm cả các lân cận) & các phép đo (ví dụ: độ tập trung & mật độ), được sử dụng để hiểu & phân tích cấu trúc đồ thị.

Additionally, discussed adjacency matrix & its different representations. Finally, explored 2 fundamental graph algorithms, BFS & DFS, which form foundation for developing more complex graph algorithms.

– Ngoài ra, chúng tôi còn thảo luận về ma trận kề & các biểu diễn khác nhau của nó. Cuối cùng, chúng tôi đã khám phá 2 thuật toán đồ thị cơ bản, BFS & DFS, tạo thành nền tảng cho việc phát triển các thuật toán đồ thị phức tạp hơn.

In Chap. 3: Creating Node Representation with DeepWalk, explore DeepWalk architecture & its 2 components: **Word2Vec** & random walks. Start by understanding **Word2Vec** architecture & then implement it using a specialized library. Then, delve into DeepWalk algorithm & implement random walks on a graph.

– Trong Chương 3: Tạo Biểu diễn Nút với DeepWalk, khám phá kiến trúc DeepWalk & 2 thành phần của nó: **Word2Vec** & bước ngẫu nhiên. Bắt đầu bằng cách tìm hiểu kiến trúc **Word2Vec** & sau đó triển khai nó bằng 1 thư viện chuyên dụng. Sau đó, đi sâu vào thuật toán DeepWalk & triển khai bước ngẫu nhiên trên đồ thị.

- 3. **Creating Node Representations with DeepWalk.** DeepWalk is 1 of 1st major successful applications of ML techniques to graph data. It introduces important concepts e.g. embeddings that are at core of GNNs. Unlike traditional neural networks, goal

of this architecture: produce representations that are then fed to other models, which perform downstream tasks (e.g., node classification).

– 3. **Tạo Biểu diễn Nút với DeepWalk.** DeepWalk là 1 trong những ứng dụng thành công đầu tiên của kỹ thuật ML vào đồ thị dữ liệu. Nó giới thiệu các khái niệm quan trọng, ví dụ như nhúng, vốn là cốt lõi của mạng nơ-ron nhân tạo (GNN). Không giống như mạng nơ-ron truyền thống, mục tiêu của kiến trúc này là tạo ra các biểu diễn sau đó được đưa vào các mô hình khác, thực hiện các tác vụ tiếp theo (ví dụ: phân loại nút).

In this chap, learn about DeepWalk architecture & its 2 major components: Word2Vec & random walks. Explain how Word2Vec architecture works, with a particular focus on skip-gram model. Implement this model with popular **gensim** library on a NLP example to understand how it is supposed to be used.

– Trong chương này, chúng ta sẽ tìm hiểu về kiến trúc DeepWalk & 2 thành phần chính của nó: Word2Vec & random walks. Giải thích cách thức hoạt động của kiến trúc Word2Vec, đặc biệt tập trung vào mô hình skip-gram. Triển khai mô hình này với thư viện **gensim** phổ biến trên 1 ví dụ NLP để hiểu cách sử dụng.

Then focus on DeepWalk algorithm & see how performance can be improved using hierarchical softmax (H-Softmax). This powerful optimization of softmax function can be found in many fields: it is incredibly useful when you have a lot of possible classes in your classification task. Also implement random walks on a graph before wrapping things up with an end-to-end supervised classification exercise on Zachary’s Karate Club.

– Sau đó, tập trung vào thuật toán DeepWalk & xem cách cải thiện hiệu suất bằng cách sử dụng softmax phân cấp (H-Softmax). Việc tối ưu hóa mạnh mẽ hàm softmax này có thể được tìm thấy trong nhiều lĩnh vực: nó cực kỳ hữu ích khi bạn có nhiều lớp khả thi trong tác vụ phân loại của mình. Hãy triển khai các bước ngẫu nhiên trên đồ thị trước khi kết thúc bằng bài tập phân loại có giám sát đầu cuối trên Zachary’s Karate Club.

By end of this chap, master Word2Vec in context of NLP & beyond. Able to create node embeddings using topological information of graphs & solve classification tasks on graph data. In this chap, cover main topics: introducing Word2Vec, DeepWalk & random walks, implementing DeepWalk.

– Đến cuối chương này, bạn sẽ nắm vững Word2Vec trong ngữ cảnh NLP & hơn thế nữa. Có khả năng tạo nhúng nút bằng thông tin tôpô của đồ thị & giải quyết các bài toán phân loại trên dữ liệu đồ thị. Trong chương này, bạn sẽ tìm hiểu các chủ đề chính: giới thiệu Word2Vec, DeepWalk & bước ngẫu nhiên, và triển khai DeepWalk.

◦ **Introducing Word2Vec.** 1st step to comprehending DeepWalk algorithm: understand its major component: Word2Vec. Word2Vec has been 1 of most influential DL techniques in NLP. Published in 2013 by TOMAS MIKOLOV et al. (Google) in 2 different papers, it proposed a new technique to translate words into vectors (also known as *embeddings*) using large datasets of text. These representations can then be used in downstream tasks, e.g. sentiment classification. It is also 1 of rare examples of patented & popular ML architecture.

– **Giới thiệu Word2Vec.** Bước đầu tiên để hiểu thuật toán DeepWalk: hiểu thành phần chính của nó: Word2Vec. Word2Vec là 1 trong những kỹ thuật DL có ảnh hưởng nhất trong NLP. Được công bố vào năm 2013 bởi TOMAS MIKOLOV và cộng sự (Google) trong 2 bài báo khác nhau, kỹ thuật này đã đề xuất 1 kỹ thuật mới để dịch từ thành các vectơ (còn được gọi là NHÚNG) bằng cách sử dụng các tập dữ liệu văn bản lớn. Các biểu diễn này sau đó có thể được sử dụng trong các tác vụ tiếp theo, ví dụ như phân loại cảm xúc. Đây cũng là 1 trong những ví dụ hiếm hoi về kiến trúc ML phổ biến và đã được cấp bằng sáng chế.

A few examples of how Word2Vec can transform words into vectors: $vec(king) = [-2.1, 4.1, 0.6]$, $vec(queen) = [-1.9, 2.6, 1.5]$, $vec(man) = [3.0, -1.1, -2]$, $vec(woman) = [2.8, -2.6, -1.1]$. Can see in this example: in terms of Euclidean distance, word vectors for *king* & *queen* are closer than ones for *king* & *woman* (4.37 vs. 8.47). In general, other metrics, e.g. popular *cosine similarity*, are used to measure likeness of these words. Cosine similarity focuses on angle between vectors & does not consider their magnitude (length), which is more helpful in comparing them. Here is how it is defined:

$$\text{cosine similarity}(\vec{A}, \vec{B}) = \cos \theta = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}.$$

1 of most surprising results of Word2Vec is its ability to solve analogies. A popular example is how it can answer question “man is to woman, what kind is to ...?” It can be calculated as follows: $vec(king) - vec(man) + vec(woman) \approx vec(queen)$. This is not true with any analogy, but this property can bring interesting applications to perform arithmetic operations with embeddings.

– 1 vài ví dụ về cách Word2Vec có thể chuyển đổi từ thành vectơ: $vec(king) = [-2.1, 4.1, 0.6]$, $vec(queen) = [-1.9, 2.6, 1.5]$, $vec(man) = [3.0, -1.1, -2]$, $vec(woman) = [2.8, -2.6, -1.1]$. Có thể thấy trong ví dụ này: về khoảng cách Euclidean, vectơ từ cho *king* & *queen* gần hơn vectơ cho *king* & *woman* (4,37 so với 8,47). Nhìn chung, các số liệu khác, ví dụ như *cosine similarity* phổ biến, được sử dụng để đo mức độ giống nhau của các từ này. Độ giống nhau cosine tập trung vào góc giữa các vectơ & không xem xét độ lớn (chiều dài) của chúng, điều này hữu ích hơn khi so sánh chúng. Đây là cách định nghĩa của nó:

$$\text{cosine similarity}(\vec{A}, \vec{B}) = \cos \theta = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}.$$

1 trong những kết quả đáng ngạc nhiên nhất của Word2Vec là khả năng giải quyết các phép loại suy. 1 ví dụ phổ biến là cách nó có thể trả lời câu hỏi “đàn ông là gì với phụ nữ, vậy loại nào là ...?”. Nó có thể được tính như sau: $vec(king) -$

$vec(man) + vec(woman) \approx vec(queen)$. Điều này không đúng với bất kỳ phép loại suy nào, nhưng tính chất này có thể mang lại những ứng dụng thú vị để thực hiện các phép toán số học với phép nhúng.

- * **CBOW vs. skip-gram.** A model must be trained on a pretext task to produce these vectors. Task itself does not need to be meaningful: its only goal is to produce high-quality embeddings. In practice, this task is always related to predicting words given a certain context.

- **CBOW so với skip-gram.** 1 mô hình phải được huấn luyện trên 1 tác vụ giả định để tạo ra các vectơ này. Bản thân tác vụ không cần phải có ý nghĩa: mục tiêu duy nhất của nó là tạo ra các nhúng chất lượng cao. Trên thực tế, tác vụ này luôn liên quan đến việc dự đoán các từ trong 1 ngữ cảnh nhất định.

Authors proposed 2 architectures with similar tasks:

- **Continuous bag-of-words (CBOW) model.** This is trained to predict a word its surrounding context (words coming before & after target word). Order of context words does not matter since their embeddings are summed in model. Authors claim to obtain better results using 4 words before & after the one that is predicted.
- **Continuous skip-gram model.** Here we feed a single word to model & try to predict words around it. Increasing range of context words lead to better embeddings but also increases training time.

In summary, here are inputs & outputs of both models: Fig. 3.1: CBOW & skip-gram architectures.

- Các tác giả đã đề xuất 2 kiến trúc với các nhiệm vụ tương tự:

- **Mô hình túi từ liên tục (CBOW).** Mô hình này được huấn luyện để dự đoán 1 từ trong ngữ cảnh xung quanh nó (các từ đứng trước & sau từ mục tiêu). Thứ tự của các từ ngữ cảnh không quan trọng vì các nhúng của chúng được tổng hợp trong mô hình. Các tác giả tuyên bố thu được kết quả tốt hơn khi sử dụng 4 từ đứng trước & sau từ được dự đoán.
- **Mô hình bỏ qua liên tục.** Ở đây, chúng tôi đưa 1 từ duy nhất vào mô hình & cố gắng dự đoán các từ xung quanh nó. Việc tăng phạm vi từ ngữ cảnh dẫn đến các nhúng tốt hơn nhưng cũng làm tăng thời gian huấn luyện.

Tóm lại, đây là đầu vào & đầu ra của cả hai mô hình: Hình 3.1: Kiến trúc CBOW & bỏ qua.

In general, CBOW model is considered faster to train, but skip-gram model is more accurate thanks to its ability to learn infrequent words. This topic is still debated in NLP community: a different implementation could fix issues related to CBOW in some contexts.

- Nhìn chung, mô hình CBOW được đánh giá là nhanh hơn khi huấn luyện, nhưng mô hình skip-gram lại chính xác hơn nhờ khả năng học các từ ít phổ biến. Chủ đề này vẫn đang được tranh luận trong cộng đồng NLP: 1 triển khai khác có thể khắc phục các vấn đề liên quan đến CBOW trong 1 số bối cảnh.

- * **Creating skip-grams.** For now, focus on skip-gram model since it is architecture used by DeepWalk. Skip-grams are implemented as pairs of words with following structure (target word, context word), where *target word* is input & *context word* is word to predict. Number of skip grams for same target word depends on a parameter called *context size*, as shown in Fig. 3.2: Text to skip-grams. Same idea can be applied to a corpus of text instead of a single sentence.

- **Tạo skip-gram.** Trước mắt, hãy tập trung vào mô hình skip-gram vì đây là kiến trúc được DeepWalk sử dụng. Skip-gram được triển khai theo cặp từ với cấu trúc sau (từ đích, từ ngữ cảnh), trong đó từ đích là đầu vào & từ ngữ cảnh là từ cần dự đoán. Số lượng skip-gram cho cùng 1 từ đích phụ thuộc vào 1 tham số gọi là kích thước ngữ cảnh, như thể hiện trong Hình 3.2: Văn bản thành skip-gram. Ý tưởng tương tự có thể được áp dụng cho 1 tập hợp văn bản thay vì 1 câu đơn lẻ.

In practice, store all context words for same target word in a list to save memory. See how it is done with an example on an entire paragraph. In following example, create skip-grams for an entire paragraph stored in `text` variable. Set `CONTEXT_SIZE` variable to 2, i.e., look at 2 words before & after our target word:

1. Start by importing necessary libraries: `numpy`.
2. Then need to set `CONTEXT_SIZE` variable to 2 & bring in text we want to analyze:
3. Next create skip-grams thanks to a simple `for` loop to consider every word in `text`. A list comprehension generates context words, stored in `skipgrams` list:

```
skipgrams = []
for i in range(CONTEXT_SIZE, len(text) - CONTEXT_SIZE):
    array = [text[j] for j in np.arange(i - CONTEXT_SIZE, i + CONTEXT_SIZE + 1) if j != i]
    skipgrams.append((text[i], array))
```

4. Finally, use `print()` function to see skip-grams we generated:

```
print(skipgrams[0:2])
```

These 2 target words, with their corresponding context, work to show what inputs to Word2Vec look like.

- Trong thực tế, hãy lưu trữ tất cả các từ ngữ cảnh cho cùng 1 từ mục tiêu trong 1 danh sách để tiết kiệm bộ nhớ. Xem cách thực hiện với ví dụ trên toàn bộ 1 đoạn văn. Trong ví dụ sau, hãy tạo skip-gram cho toàn bộ 1 đoạn văn được lưu trữ trong biến `text`. Đặt biến `CONTEXT_SIZE` thành 2, tức là xem xét 2 từ trước & sau từ mục tiêu của chúng ta:

1. Bắt đầu bằng cách nhập các thư viện cần thiết: `numpy`.
2. Sau đó, cần đặt biến `CONTEXT_SIZE` thành 2 & nhập văn bản chúng ta muốn phân tích:
3. Tiếp theo, hãy tạo skip-gram nhờ vòng lặp `for` đơn giản để xem xét mọi từ trong `text`. 1 danh sách hiểu biết tạo ra các từ ngữ cảnh, được lưu trữ trong danh sách `skipgrams`:

```

skipgrams = []
for i in range(CONTEXT_SIZE, len(text) - CONTEXT_SIZE):
    array = [text[j] for j in np.arange(i - CONTEXT_SIZE, i + CONTEXT_SIZE + 1) if j != i]
    skipgrams.append((text[i], array))

```

4. Cuối cùng, sử dụng hàm `print()` để xem các skip-gram mà chúng ta đã tạo ra:

```
print(skipgrams[0:2])
```

Hai từ mục tiêu này, cùng với ngữ cảnh tương ứng, sẽ hiển thị các dữ liệu đầu vào của Word2Vec trông như thế nào.

* **skip-gram model.** Goal of Word2Vec: produce high-quality word embeddings. To learn these embeddings, training task of skip-gram model consists of predicting correct context words given a target word.

– mô hình skip-gram. Mục tiêu của Word2Vec: tạo ra các nhúng từ chất lượng cao. Để học các nhúng này, nhiệm vụ huấn luyện của mô hình skip-gram bao gồm dự đoán các từ ngữ cảnh chính xác dựa trên 1 từ mục tiêu.

Imagine we have a sequence of N words w_1, w_2, \dots, w_N . Probability of seeing word w_2 given word w_1 is written $p(w_2|w_1)$. Goal: maximize sum of every probability of seeing a context word given a target word in an entire text:

$$\frac{1}{N} \sum_{n=1}^N \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{n+j}|w_n),$$

where c is size of context vector.

– Tưởng tượng chúng ta có 1 chuỗi N từ w_1, w_2, \dots, w_N . Xác suất nhìn thấy từ w_2 cho từ w_1 được viết là $p(w_2|w_1)$. Mục tiêu: tối đa hóa tổng của mọi xác suất nhìn thấy 1 từ ngữ cảnh cho 1 từ đích trong toàn bộ văn bản:

$$\frac{1}{N} \sum_{n=1}^N \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{n+j}|w_n),$$

trong đó c là kích thước của vectơ ngữ cảnh.

Remark 1. Why do we use a log probability in prev equation? Transforming probabilities into log probabilities is a common technique in ML (& CS in general) for 2 main reasons. Products become additions (& divisions become subtractions). Multiplications are more computationally expensive than additions, so it is faster to compute log probability: $\log(ab) = \log a + \log b$. Way computers store very small numbers, e.g., $3.14e-128$ is not perfectly accurate, unlike log of same numbers -127.5 in this case. These small errors can add up & bias final results when events are extremely unlikely. On whole, this simple transformation allows us to gain speed & accuracy without changing our initial objective.

– Tại sao chúng ta sử dụng xác suất logarit trong phương trình prev? Việc chuyển đổi xác suất thành xác suất logarit là 1 kỹ thuật phổ biến trong ML (& CS nói chung) vì 2 lý do chính. Tích trở thành phép cộng (& phép chia trở thành phép trừ). Phép nhân tốn kém về mặt tính toán hơn phép cộng, do đó, tính xác suất logarit nhanh hơn: $\log(ab) = \log a + \log b$. Cách máy tính lưu trữ những số rất nhỏ, ví dụ: $3.14e-128$ không hoàn toàn chính xác, không giống như logarit của cùng 1 số -127.5 trong trường hợp này. Những sai số nhỏ này có thể cộng lại & làm sai lệch kết quả cuối cùng khi các sự kiện cực kỳ khó xảy ra. Nhìn chung, phép biến đổi đơn giản này cho phép chúng ta tăng tốc & độ chính xác mà không làm thay đổi mục tiêu ban đầu.

Basic skip-gram models uses softmax function to calculate probability of a context word embedding h_c given a target word embedding h_t :

$$p(w_c|w_t) = \frac{\exp(h_c h_t^\top)}{\sum_{i=1}^{|V|} \exp(h_i h_t^\top)},$$

where V is vocabulary of size $|V|$. This vocabulary corresponds to list of unique words the model tries to predict. Can obtain this list using `set` data structure to remove duplicate words:

– Các mô hình skip-gram cơ bản sử dụng hàm softmax để tính xác suất nhúng từ ngữ cảnh h_c cho 1 từ mục tiêu nhúng h_t :

$$p(w_c|w_t) = \frac{\exp(h_c h_t^\top)}{\sum_{i=1}^{|V|} \exp(h_i h_t^\top)},$$

trong đó V là từ vựng có kích thước $|V|$. Từ vựng này tương ứng với danh sách các từ duy nhất mà mô hình cố gắng dự đoán. Có thể lấy danh sách này bằng cách sử dụng cấu trúc dữ liệu `set` để loại bỏ các từ trùng lặp:

```

# skip-gram model
vocab = set(text)
VOCAB_SIZE = len(vocab)
print(f"Length of vocabulary = {VOCAB_SIZE}")

```

Now have size of our vocabulary, there is 1 more parameter we need to define: N , dimensionality of word vectors. Typically, this value is set between 100 & 1000. In this example, set it to 10 because of limited size of our dataset.

– Bây giờ, với kích thước của vốn từ vựng, chúng ta cần định nghĩa thêm 1 tham số nữa: N , số chiều của vectơ từ. Thông thường, giá trị này được đặt trong khoảng từ 100 đến 1000. Trong ví dụ này, hãy đặt thành 10 vì kích thước tập dữ liệu của chúng ta có hạn.

Skip-gram model is composed of only 2 layers:

- A *projection layer* with a weight matrix W_{embed} , which takes a 1-hot encoded-word vector as an input & returns corresponding N -dim word embedding. It acts as a simple lookup table that stores embeddings of a predefined dimensionality.
- A *fully connected layer* with a weight matrix W_{output} , which takes a word embedding as input & outputs $|V|$ -dim logits. A softmax function is applied to these predictions to transform logits into probabilities.
- Mô hình Skip-gram chỉ bao gồm 2 lớp:
 - A *projection layer* với ma trận trọng số W_{embed} , lấy 1 vectơ từ được mã hóa 1-hot làm đầu vào & trả về nhúng từ N -dim tương ứng. Nó hoạt động như 1 bảng tra cứu đơn giản lưu trữ các nhúng có chiều được xác định trước.
 - A *fully connected layer* với ma trận trọng số W_{output} , lấy 1 nhúng từ làm đầu vào & trả về logit $|V|$ -dim. 1 hàm softmax được áp dụng cho các dự đoán này để chuyển đổi logit thành xác suất.

Remark 2. *There is no activation function: Word2Vec is a linear classifier that models a linear relationship between words.*

– Không có hàm kích hoạt: Word2Vec là 1 bộ phân loại tuyến tính mô hình hóa mối quan hệ tuyến tính giữa các từ.

Call \mathbf{x} 1-hot encoded-word vector *input*. Corresponding word embedding can be calculated as a simple projection $h = W_{\text{embed}}^T \cdot \mathbf{x}$. Using skip-gram model, can rewrite previous probability as follows:

$$p(w_c|w_t) = \frac{\exp(W_{\text{output}} \cdot h)}{\sum_{i=1}^{|V|} \exp(W_{\text{output}(i)} \cdot h)}.$$

Skip-gram model outputs a $|V|$ -dim vector, which is conditional probability of every word in vocabulary:

$$\text{word2vec}(w_t) = \begin{bmatrix} p(w_1|w_t) \\ p(w_2|w_t) \\ \vdots \\ p(w_{|V|}|w_t) \end{bmatrix}.$$

During training, these probabilities are compared to correct 1-hot encoded-target word vectors. Difference between these values (calculated by a loss function e.g. cross-entropy loss) is backpropagated through network to update weights & obtain better predictions.

– Gọi \mathbf{x} là vectơ từ mã hóa 1-hot *input*. Nhúng từ tương ứng có thể được tính toán như 1 phép chiếu đơn giản $h = W_{\text{embed}}^T \cdot \mathbf{x}$. Sử dụng mô hình skip-gram, có thể viết lại xác suất trước đó như sau:

$$p(w_c|w_t) = \frac{\exp(W_{\text{output}} \cdot h)}{\sum_{i=1}^{|V|} \exp(W_{\text{output}(i)} \cdot h)}.$$

Mô hình Skip-gram đưa ra 1 vectơ $|V|$ -dim, là xác suất có điều kiện của mọi từ trong từ vựng:

$$\text{word2vec}(w_t) = \begin{bmatrix} p(w_1|w_t) \\ p(w_2|w_t) \\ \vdots \\ p(w_{|V|}|w_t) \end{bmatrix}.$$

Trong quá trình huấn luyện, các xác suất này được so sánh với các vectơ từ mục tiêu được mã hóa 1-hot chính xác. Sự khác biệt giữa các giá trị này (được tính bằng hàm mất mát, ví dụ: mất mát entropy chéo) được truyền ngược qua mạng để cập nhật trọng số & thu được dự đoán tốt hơn.

Entire Word2Vec architecture is summarized in following diagram, with both matrices & final softmax layer: Fig. 3.3: Word2Vec architecture. Can implement this model using **gensim** library, which is also used in official implementation of DeepWalk. Can then build vocabulary & train our model based on previous text:

1. Begin by installing **gensim** & importing **Word2Vec** class:
2. Initialize a skip-gram model with a **Word2Vec** object & an **sg = 1** parameter (skip-gram = 1).
3. A good idea to check shape of our 1st weight matrix. It should correspond to vocabulary size & word embeddings' dimensionality.
4. Train model for 10 epochs.
5. Can print a word embedding to see what result of this training looks like.

While this approach works well with small vocabularies, computational cost of applying a full softmax function to millions of words (vocabulary size) is too costly in most cases. This has been a limiting factor in developing accurate language models for a long time. Fortunately for us, other approaches have been designed to solve this issue.

– Toàn bộ kiến trúc Word2Vec được tóm tắt trong sơ đồ sau, với cả ma trận & lớp softmax cuối cùng: Hình 3.3: Kiến trúc Word2Vec. Có thể triển khai mô hình này bằng thư viện **gensim**, cũng được sử dụng trong triển khai chính thức của DeepWalk. Sau đó, có thể xây dựng từ vựng & huấn luyện mô hình của chúng ta dựa trên văn bản trước đó:

1. Bắt đầu bằng cách cài đặt **gensim** & nhập lớp **Word2Vec**:
2. Khởi tạo mô hình skip-gram với đối tượng **Word2Vec** & tham số **sg = 1** (skip-gram = 1).
3. Nên kiểm tra hình dạng của ma trận trọng số thứ nhất. Nó phải tương ứng với kích thước từ vựng & chiều của nhúng từ.
4. Huấn luyện mô hình trong 10 kỷ nguyên.
5. Có thể in 1 nhúng từ để xem kết quả của quá trình huấn luyện này trông như thế nào.

Mặc dù phương pháp này hiệu quả với các từ vựng nhỏ, nhưng chi phí tính toán khi áp dụng hàm softmax đầy đủ cho hàng triệu từ (kích thước từ vựng) thường quá tốn kém. Đây là 1 yếu tố hạn chế trong việc phát triển các mô hình ngôn ngữ chính xác trong 1 thời gian dài. May mắn thay, các phương pháp khác đã được thiết kế để giải quyết vấn đề này.

Word2Vec (& **DeepWalk**) implements 1 of these techniques, called **H-Softmax**. Instead of a flat softmax that directly calculates probability of every word, this technique uses a binary tree structure where leaves are words. Even more interestingly, a Huffman tree can be used, where infrequent words are stored at deeper levels than common words. In most cases, this dramatically speeds up word prediction by a factor ≥ 50 .

– **Word2Vec** (& **DeepWalk**) triển khai 1 trong những kỹ thuật này, được gọi là **H-Softmax**. Thay vì 1 softmax phẳng tính toán trực tiếp xác suất của từng từ, kỹ thuật này sử dụng cấu trúc cây nhị phân với các lá là các từ. Thú vị hơn nữa, có thể sử dụng cây Huffman, trong đó các từ ít phổ biến được lưu trữ ở các cấp độ sâu hơn so với các từ phổ biến. Trong hầu hết các trường hợp, điều này giúp tăng tốc đáng kể việc dự đoán từ lên gấp ≥ 50 .

H-Softmax can be activated in **gensim** using **hs = 1**. This was most difficult part of **DeepWalk** architecture. But before can implement it, need 1 more component: how to create our training data.

– **H-Softmax** có thể được kích hoạt trong **gensim** bằng cách sử dụng **hs = 1**. Đây là phần khó nhất của kiến trúc **DeepWalk**. Nhưng trước khi có thể triển khai nó, cần thêm 1 thành phần nữa: cách tạo dữ liệu huấn luyện.

- o **DeepWalk & random walks**. Proposed in 2014 by **PEROZZI** et al., **DeepWalk** quickly became extremely popular among graph researchers. Inspired by recent advances in NLP, it consistently outperformed other methods on several datasets. While more performant architectures have been proposed since then, **DeepWalk** is a simple & reliable baseline that can be quickly implemented to solve a lot of problems.

– **DeepWalk & random walks**. Được đề xuất vào năm 2014 bởi **PEROZZI** và cộng sự, **DeepWalk** nhanh chóng trở nên cực kỳ phổ biến trong giới nghiên cứu đồ thị. Lấy cảm hứng từ những tiến bộ gần đây trong NLP, nó luôn vượt trội hơn các phương pháp khác trên 1 số tập dữ liệu. Mặc dù các kiến trúc hiệu suất cao hơn đã được đề xuất kể từ đó, **DeepWalk** là 1 đường cơ sở đơn giản & đáng tin cậy, có thể được triển khai nhanh chóng để giải quyết nhiều vấn đề.

Goal of **DeepWalk**: produce high-quality feature representations of nodes in an unsupervised way. This architecture is heavily inspired by **Word2Vec** in NLP. However, instead of words, our dataset is composed of nodes. This is why we use random walks to generate meaningful sequence of nodes that act like sentences. Following diagram illustrates connection between sentences & graphs: Fig. 3.4: Sentences can be represented as graphs. Random walks are sequences of nodes produced by randomly choosing a neighboring node at every step. Thus, nodes can appear several times in same sequence.

– Mục tiêu của **DeepWalk**: tạo ra các biểu diễn đặc trưng chất lượng cao của các nút theo cách không giám sát. Kiến trúc này lấy cảm hứng rất nhiều từ **Word2Vec** trong NLP. Tuy nhiên, thay vì các từ, tập dữ liệu của chúng tôi được tạo thành từ các nút. Đây là lý do tại sao chúng tôi sử dụng các bước ngẫu nhiên để tạo ra chuỗi các nút có ý nghĩa hoạt động như các câu. Sơ đồ sau minh họa mối liên hệ giữa các câu & đồ thị: Hình 3.4: Các câu có thể được biểu diễn dưới dạng đồ thị. Các bước ngẫu nhiên là các chuỗi nút được tạo ra bằng cách chọn ngẫu nhiên 1 nút lân cận ở mỗi bước. Do đó, các nút có thể xuất hiện nhiều lần trong cùng 1 chuỗi.

Why are random walks important? Even if nodes are randomly selected, fact they often appear together in a sequence means: they are close to each other. Under *network homophily* hypothesis, notes that are close to each other are similar. This is particularly case in social networks, where people are connected to friends & family.

– Tại sao các bước ngẫu nhiên lại quan trọng? Ngay cả khi các nút được chọn ngẫu nhiên, việc chúng thường xuất hiện cùng nhau theo 1 trình tự có nghĩa là: chúng gần nhau. Theo giả thuyết *network homophily*, các nốt gần nhau thì tương tự nhau. Điều này đặc biệt đúng trong các mạng xã hội, nơi mọi người kết nối với bạn bè & gia đình.

This idea is at core of **DeepWalk** algorithm: when nodes are close to each other, want to obtain high similarity scores. On contrary, want low scores when they are farther apart. Implement a random walk function using a **networkx** graph:

– Ý tưởng này là cốt lõi của thuật toán **DeepWalk**: khi các nút gần nhau, chúng ta muốn đạt được điểm tương đồng cao. Ngược lại, chúng ta muốn đạt điểm tương đồng thấp khi chúng ở xa nhau. Triển khai hàm bước ngẫu nhiên bằng đồ thị **networkx**:

1. Import required libraries & initialize random number generator for reproducibility:

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import random
random.seed(0) # initialize random number generator for reproducibility
```

2. Generate a random graph thanks to **erdos_renyi_graph** function with a fixed number of nodes 10 & a predefined probability of creating an edge between 2 nodes 0.3:

```
G = nx.erdos_renyi_graph(10, 0.3, seed = 1, directed = False)
```

3. Plot this random graph to see what it looks like.

4. Implement random walks with a simple function. This function takes 2 parameters: starting node **start** & length of walk **length**. At every step, randomly select a neighboring node (using `np.random.choice`) until walk is complete:

```
# implement random walks
def random_walk(start, length):
    walk = [str(start)] # starting node
    for i in range(length):
        neighbors = [node for node in G.neighbors(start)]
        next_node = np.random.choice(neighbors, 1)[0]
        walk.append(str(next_node))
        start = next_node
    return walk
```

5. Print result of this function with starting node as 0 & a length of 10:

```
print(random_walk(0, 10))
```

Can see: certain nodes, e.g. 0 & 9, are often found together. Considering that it is a homophilic graph, i.e., they are similar. It is precisely type of relationship we are trying to capture with DeepWalk. Now have implemented Word2Vec & random walks separately, combine them to create DeepWalk.

– Có thể thấy: 1 số nút nhất định, ví dụ: 0 & 9, thường được tìm thấy cùng nhau. Xét đến việc đây là 1 đồ thị đồng dạng, tức là chúng tương tự nhau. Đây chính xác là loại mối quan hệ mà chúng ta đang cố gắng nắm bắt bằng DeepWalk. Bây giờ, chúng ta đã triển khai Word2Vec & random walks riêng biệt, hãy kết hợp chúng để tạo ra DeepWalk.

○ **Implementing DeepWalk.** Now have a good understanding of every component in this architecture, use it to solve an ML problem. Dataset we use is Zachary's Karate Club. It simply represents relationships within a karate club studied by WAYNE W. ZACHARY in 1970s. It is a kind of social network where every node is a member, & members who interact outside club are connected.

– **Triển khai DeepWalk.** Bây giờ, hãy hiểu rõ từng thành phần trong kiến trúc này, hãy sử dụng nó để giải quyết 1 bài toán ML. Tập dữ liệu chúng tôi sử dụng là Câu lạc bộ Karate của Zachary. Nó đơn giản biểu diễn các mối quan hệ trong 1 câu lạc bộ karate được WAYNE W. ZACHARY nghiên cứu vào những năm 1970. Đây là 1 dạng mạng xã hội, trong đó mỗi nút là 1 thành viên, & các thành viên tương tác bên ngoài câu lạc bộ được kết nối.

In this example, club is divided into 2 groups: could like to assign right group to every member (node classification) just by looking at their connections:

1. Import dataset using `nx.karate_club_graph()`:

```
G = nx.karate_club_graph()
```

2. Need to convert string class labels into numerical values Mr. Hi = 0, Officer = 1:

3. Plot this graph using our new labels.

4. Next step: generate our dataset, random walks. Want to be as exhaustive as possible, which is why we will create 80 random walks of a length of 10 for every node in graph.

5. Print a walk to verify that it is correct: 1st walk that was generated.

6. Final step consists of implementing Word2Vec. Here use skip-gram model previously seen with H-Softmax. Can play with other parameters to improve quality of embeddings.

7. Model is then simply trained on random walks generated:

```
model.train(walks, total_examples = model.corpus_count, epochs = 30, report_delay = 1)
```

8. Now our model is trained, see its different applications. 1st one allows us to find most similar nodes to a given one (in terms of cosine similarity):

```
print('Nodes that are the most similar to node 0:')
for similarity in model.wv.most_similar(positive = ['0']):
    print(f' {similarity}')
```

This produces following output for Nodes most similar to node 0. Another important application is calculating similarity score between 2 nodes. It can be performed as follows:

```
# similarity between 2 nodes
print(f"Similarity between node 0 & 4: {model.wv.similarity('0', '4')}")
```

– Trong ví dụ này, câu lạc bộ được chia thành 2 nhóm: có thể muốn gán đúng nhóm cho từng thành viên (phân loại nút) chỉ bằng cách xem xét các kết nối của họ:

1. Nhập tập dữ liệu bằng `nx.karate_club_graph()`:

```
G = nx.karate_club_graph()
```

2. Cần chuyển đổi nhãn lớp chuỗi thành giá trị số `Mr. Hi = 0, Officer = 1`:

3. Vẽ đồ thị này bằng các nhãn mới của chúng ta.

4. Bước tiếp theo: tạo tập dữ liệu của chúng ta, các bước đi ngẫu nhiên. Muốn càng đầy đủ càng tốt, đó là lý do tại sao chúng ta sẽ tạo 80 bước đi ngẫu nhiên có độ dài 10 cho mỗi nút trong đồ thị.

5. In 1 bước đi để xác minh rằng nó chính xác: Bước đi đầu tiên đã được tạo.

6. Bước cuối cùng bao gồm việc triển khai Word2Vec. Ở đây sử dụng mô hình skip-gram đã thấy trước đó với H-Softmax. Có thể sử dụng các tham số khác để cải thiện chất lượng nhúng.

7. Mô hình sau đó được huấn luyện đơn giản dựa trên các bước đi ngẫu nhiên được tạo ra:

```
model.train(walks, total_examples = model.corpus_count, epochs = 30, report_delay = 1)
```

8. Bây giờ mô hình của chúng ta đã được huấn luyện, hãy xem các ứng dụng khác nhau của nó. Ứng dụng đầu tiên cho phép chúng ta tìm các nút giống nhau nhất với 1 nút cho trước (theo độ tương đồng cosin):

```
print('Các nút giống nhau nhất với nút 0:')
để biết độ tương đồng trong model.wv.most_similar(positive = ['0']):
print(f' {similarity}')
```

Điều này tạo ra kết quả sau cho Nút giống nhau nhất với nút 0. 1 ứng dụng quan trọng khác là tính điểm tương đồng giữa 2 nút. Có thể thực hiện như sau:

```
# độ tương đồng giữa 2 nút
print(f"Độ tương đồng giữa nút 0 và 4: {model.wv.similarity('0', '4')}")
```

Can plot resulting embeddings using *t-distributed stochastic neighbor embedding* (t-SNE) to visualize these high-dimensional vectors in 2D:

1. Import TSNE class from `sklearn`:

```
from sklearn.manifold import TSNE
```

2. Create 2 arrays: 1 to store word embeddings & the other 1 to store labels.

3. Next train t-SNE model with 2 dimension `n_components = 2` on embeddings.

4. Plot 2D vectors produced by trained t-SNE model with corresponding labels.

This plot is quite encouraging since we can see a clear line the separates 2 classes. It should be possible for a simple ML algorithm to classify these nodes with enough examples (training data). Implement a classifier & train it on our node embeddings:

1. Import a Random Forest model from `sklearn`, which is a popular choice when it comes to classification. Accuracy score is metric we will use to evaluate this model:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

2. Need to split embeddings into 2 groups: training & test data. A simple way of doing it: create masks as follows.

3. Next train Random Forest classifier on training data with appropriate labels.

4. Finally evaluate trained model on test data based on its accuracy score to give final result of our classifier.

– Có thể vẽ đồ thị các phép nhúng kết quả bằng cách sử dụng nhúng lân cận ngẫu nhiên phân phối t (t-SNE) để trực quan hóa các vectơ nhiều chiều này trong 2D:

1. Nhập lớp TSNE từ `sklearn`:

```
from sklearn.manifold import TSNE
```

2. Tạo 2 mảng: 1 để lưu trữ các phép nhúng từ & 1 để lưu trữ các nhãn.

3. Huấn luyện mô hình t-SNE tiếp theo với `n_components = 2` 2 chiều trên các phép nhúng.

4. Vẽ đồ thị các vectơ 2D được tạo ra bởi mô hình t-SNE đã huấn luyện với các nhãn tương ứng.

Đồ thị này khá đáng khích lệ vì chúng ta có thể thấy 1 đường rõ ràng phân tách 2 lớp. 1 thuật toán ML đơn giản có thể phân loại các nút này với đủ ví dụ (dữ liệu huấn luyện). Triển khai bộ phân loại & huấn luyện nó trên các nhúng nút của chúng ta:

1. Nhập 1 mô hình Rừng Ngẫu nhiên từ `sklearn`, đây là 1 lựa chọn phổ biến khi phân loại. Điểm chính xác là thước đo chúng ta sẽ sử dụng để đánh giá mô hình này:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

2. Cần chia các nhúng thành 2 nhóm: dữ liệu huấn luyện & dữ liệu kiểm tra. 1 cách đơn giản để thực hiện: tạo mặt nạ như sau.
3. Tiếp theo, huấn luyện bộ phân loại Rừng Ngẫu nhiên trên dữ liệu huấn luyện với các nhãn phù hợp.
4. Cuối cùng, đánh giá mô hình đã huấn luyện trên dữ liệu kiểm tra dựa trên điểm chính xác của nó để đưa ra kết quả cuối cùng cho bộ phân loại của chúng ta.

Our model obtains an accuracy score of 95.45%, which is pretty good considering unfavorable train/test split we gave it. There is still room for improvement, but this example showed 2 useful applications of DeepWalk:

- * *Discovering similarities between nodes* using embeddings & cosine similarity (unsupervised learning)
- * *Using these embeddings as a dataset* for a supervised task e.g. node classification.

See in following chaps, ability to learn node representations offers a lot of flexibility to design deeper & more complex architectures.

– Mô hình của chúng tôi đạt điểm chính xác là 95,45%, khá tốt khi xét đến việc phân tách bài kiểm tra train/bất lợi mà chúng tôi đã đưa ra. Vẫn còn nhiều điểm cần cải thiện, nhưng ví dụ này đã cho thấy 2 ứng dụng hữu ích của DeepWalk:

- * *Khám phá điểm tương đồng giữa các nút* bằng cách sử dụng nhúng & độ tương đồng cosine (học không giám sát)
- * *Sử dụng các nhúng này làm tập dữ liệu* cho 1 tác vụ có giám sát, ví dụ: phân loại nút.

Xem trong các chương tiếp theo, khả năng học biểu diễn nút mang lại rất nhiều tính linh hoạt để thiết kế các kiến trúc sâu hơn & phức tạp hơn.

- o **Summary.** In this chap, learned about DeepWalk architecture & its major components. Then transformed graph data into sequences using random walks to apply powerful Word2Vec algorithm. Resulting embeddings can be used to find similarities between nodes or as input to other algorithms. In particular, solved a node classification problem using a supervised approach.

– Trong chương này, chúng ta đã tìm hiểu về kiến trúc DeepWalk & các thành phần chính của nó. Sau đó, dữ liệu đồ thị được chuyển đổi thành chuỗi bằng cách sử dụng các bước ngẫu nhiên để áp dụng thuật toán Word2Vec mạnh mẽ. Kết quả nhúng có thể được sử dụng để tìm điểm tương đồng giữa các nút hoặc làm đầu vào cho các thuật toán khác. Cụ thể, chúng ta đã giải quyết được bài toán phân loại nút bằng phương pháp tiếp cận có giám sát.

In Chap. 4: Improving Embeddings with Biased Random Walks in Node2Vec, introduce a 2nd algorithm based on Word2Vec. Difference with DeepWalk: random walks can be biased towards more or less exploration, which directly impacts embeddings that are produced. Implement this algorithm on a new example & compare its representations with those obtained using DeepWalk.

– Trong Chương 4: Cải thiện nhúng với bước ngẫu nhiên có thiên vị trong Node2Vec, giới thiệu thuật toán thứ 2 dựa trên Word2Vec. Điểm khác biệt với DeepWalk: bước ngẫu nhiên có thể thiên vị theo hướng khám phá nhiều hơn hoặc ít hơn, điều này ảnh hưởng trực tiếp đến các nhúng được tạo ra. Triển khai thuật toán này trên 1 ví dụ mới & so sánh các biểu diễn của nó với các biểu diễn thu được bằng DeepWalk.

PART 2: FUNDAMENTALS. In this 2nd part of book, delve into process of constructing node representations using graph learning. Start by exploring traditional graph learning techniques, drawing on advancements made in NLP. Our aim: understand how these techniques can be applied to graphs & how they can be used to build node representations.

– Trong phần 2 của cuốn sách này, chúng ta sẽ đi sâu vào quá trình xây dựng biểu diễn nút bằng phương pháp học đồ thị. Bắt đầu bằng việc khám phá các kỹ thuật học đồ thị truyền thống, dựa trên những tiến bộ trong NLP. Mục tiêu của chúng ta: tìm hiểu cách áp dụng các kỹ thuật này vào đồ thị & cách sử dụng chúng để xây dựng biểu diễn nút.

Then move on to incorporating node features into our models & explore how they can be used to build even more accurate representations. Finally, introduce 2 of most fundamental GNN architectures, Graph Convolutional Network (GCN) & Graph Attention Network (GAT). These 2 architectures are building blocks of many state-of-art graph learning methods & will provide a solid foundation for next part.

– Sau đó, chuyển sang việc tích hợp các đặc trưng nút vào mô hình của chúng ta & khám phá cách sử dụng chúng để xây dựng các biểu diễn chính xác hơn nữa. Cuối cùng, hãy giới thiệu 2 kiến trúc GNN cơ bản nhất: Mạng Tích chập Đồ thị (GCN) & Mạng Chú ý Đồ thị (GAT). Hai kiến trúc này là nền tảng của nhiều phương pháp học đồ thị tiên tiến & sẽ cung cấp nền tảng vững chắc cho phần tiếp theo.

By end of this part, have a deeper understanding of how traditional graph learning techniques, e.g. random walks, can be used to crate node representations & develops graph applications. Additionally, learn how to build even more powerful representations using GNNs. Introduce 2 key GNN architectures & learn how they can be used to tackle various graph-based tasks.

– Đến cuối phần này, bạn sẽ hiểu sâu hơn về cách sử dụng các kỹ thuật học đồ thị truyền thống, ví dụ như bước đi ngẫu nhiên, để tạo ra các biểu diễn nút & phát triển các ứng dụng đồ thị. Ngoài ra, hãy tìm hiểu cách xây dựng các biểu diễn mạnh mẽ hơn bằng GNN. Giới thiệu 2 kiến trúc GNN chính & tìm hiểu cách sử dụng chúng để giải quyết các tác vụ dựa trên đồ thị khác nhau.

- 4. Improving Embeddings with Biased Random Walks in Node2Vec. Node2Vec is an architecture largely based on DeepWalk. In prev chap., saw 2 main components of this architecture: random walks & Word2Vec. How can improve quality of our embeddings? Interestingly enough, not with more ML. Instead, Node2Vec brings critical modifications to way random walks themselves are generated.

– Cải thiện nhúng với bước ngẫu nhiên có thiên vị trong Node2Vec. Node2Vec là 1 kiến trúc phần lớn dựa trên DeepWalk. Trong chương trước, chúng ta đã thấy hai thành phần chính của kiến trúc này: bước ngẫu nhiên & Word2Vec. Làm thế nào để cải thiện chất lượng nhúng của chúng ta? Điều thú vị là, không phải bằng cách tăng cường ML. Thay vào đó, Node2Vec mang đến những thay đổi quan trọng đối với cách tạo ra các bước ngẫu nhiên.

In this chap, talk about these modifications & how to find best parameters for a given graph. Implement Node2Vec architecture & compare it to using DeepWalk on Zachary's Karate Club. This give you a good understanding of differences between 2 architectures. Finally, use this technology to build a real application: a move *recommender system (RecSys)* powered by Node2Vec.

– Trong chương này, chúng ta sẽ thảo luận về những thay đổi này & cách tìm tham số tốt nhất cho 1 đồ thị nhất định. Triển khai kiến trúc Node2Vec & so sánh với việc sử dụng DeepWalk trên Zachary's Karate Club. Điều này giúp bạn hiểu rõ sự khác biệt giữa hai kiến trúc. Cuối cùng, hãy sử dụng công nghệ này để xây dựng 1 ứng dụng thực tế: 1 hệ thống đề xuất di chuyển (RecSys) được hỗ trợ bởi Node2Vec.

By end of this chap, know how to implement Node2Vec on any graph dataset & how to select good parameters. Understand why this architecture works better than DeepWalk in general, & how to apply it to build creative applications.

– Đến cuối chương này, bạn sẽ biết cách triển khai Node2Vec trên bất kỳ tập dữ liệu đồ thị nào & cách chọn tham số phù hợp. Hiểu lý do tại sao kiến trúc này hoạt động tốt hơn DeepWalk nói chung, & cách áp dụng nó để xây dựng các ứng dụng sáng tạo.

- Introducing Node2Vec. Node2Vec was introduced in 2016 by GROVER & LESKOVEC from Stanford University [1]. It keeps same 2 main components from DeepWalk: random walks & Word2Vec. Difference: instead of obtaining sequences of nodes with a uniform distribution, random walks are carefully biased in Node2Vec. See why these *biased random walks* perform better & how to implement them in 2 following sects: defining a *neighborhood*, introducing biases in random walks. Start by questioning our intuitive concept of neighborhoods.

– Giới thiệu Node2Vec. Node2Vec được giới thiệu vào năm 2016 bởi GROVER & LESKOVEC từ Đại học Stanford [1]. Nó giữ nguyên 2 thành phần chính từ DeepWalk: các bước ngẫu nhiên & Word2Vec. Điểm khác biệt: thay vì thu được chuỗi các nút có phân phối đều, các bước ngẫu nhiên được phân bổ cẩn thận trong Node2Vec. Xem lý do tại sao các bước ngẫu nhiên *biased random walks* này hoạt động tốt hơn & cách triển khai chúng trong 2 phần sau: định nghĩa 1 *neighborhood*, giới thiệu các bước ngẫu nhiên trong các bước ngẫu nhiên. Bắt đầu bằng cách đặt câu hỏi về khái niệm trực quan của chúng ta về các vùng lân cận.

* Defining a neighborhood. How do you define neighborhood of a node? Key concept introduced in Node2Vec is flexible notion of a neighborhood. Intuitively, we think of it as something close to initial node, but what does “close” mean in context of a graph? Want to explore 3 nodes in neighborhood of node A. This exploration process is also called a *sampling strategy*.

- A possible solution would be to consider 3 closest nodes in terms of connections. In this case, neighborhood of A, noted $N(A)$, would be $\{B, C, D\}$.
- Another possible sampling strategy consists of selecting nodes that are not adjacent to previous nodes 1st. In our example, $N(A) = \{D, E, F\}$.

I.e., want to implement a BFS in 1st case & a DFS in 2nd one.

– Định nghĩa vùng lân cận. Bạn định nghĩa vùng lân cận của 1 nút như thế nào? Khái niệm chính được giới thiệu trong Node2Vec là khái niệm linh hoạt về vùng lân cận. Theo trực giác, chúng ta nghĩ về nó như 1 thứ gì đó gần với nút ban đầu, nhưng “đóng” có nghĩa là gì trong ngữ cảnh của đồ thị? Bạn muốn khám phá 3 nút trong vùng lân cận của nút A. Quá trình khám phá này cũng được gọi là *chiến lược lấy mẫu*:

- 1 giải pháp khả thi là xem xét 3 nút gần nhất về mặt kết nối. Trong trường hợp này, vùng lân cận của A, ký hiệu là $N(A)$, sẽ là $\{B, C, D\}$.
- 1 chiến lược lấy mẫu khả thi khác bao gồm việc chọn các nút không liền kề với các nút trước đó trước. Trong ví dụ của chúng ta, $N(A) = \{D, E, F\}$.

Ví dụ, bạn muốn triển khai BFS trong trường hợp thứ nhất & DFS trong trường hợp thứ 2.

What is important to notice here: these sampling strategies have opposite behaviors: BFS focuses on local network around a node while DFS establishes a more macro view of graph. Considering our intuitive definition of a neighborhood, it is tempting to simply discard DFS. However, Node2vec's authors argue: this would be a mistake: each approach captures a different but valuable representation of network.

– Điều quan trọng cần lưu ý ở đây: các chiến lược lấy mẫu này có hành vi trái ngược nhau: BFS tập trung vào mạng cục bộ xung quanh 1 nút trong khi DFS thiết lập 1 góc nhìn vĩ mô hơn về đồ thị. Xét theo định nghĩa trực quan của chúng ta về vùng lân cận, việc loại bỏ DFS là điều dễ hiểu. Tuy nhiên, các tác giả của Node2vec lập luận: đây sẽ là 1 sai lầm: mỗi phương pháp nắm bắt 1 biểu diễn mạng khác nhau nhưng có giá trị.

They argue: BFS is ideal to emphasize structural equivalence since this strategy only looks at neighboring nodes. In these random walks, nodes are often repeated & stay close to each other. DFS, on other hand, emphasizes opposite of homophily

by creating sequences of distant nodes. These random walks can sample nodes that are far from source & thus become less representative. This is why we are looking for a trade-off between these 2 properties: homophily may be more helpful for understanding certain graphs & vice versa.

– Họ lập luận: BFS là lý tưởng để nhấn mạnh tính tương đương về mặt cấu trúc vì chiến lược này chỉ xem xét các nút lân cận. Trong các bước đi ngẫu nhiên này, các nút thường được lặp lại & ở gần nhau. Mặt khác, DFS nhấn mạnh tính đối lập của tính đồng dạng bằng cách tạo ra các chuỗi nút ở xa. Các bước đi ngẫu nhiên này có thể lấy mẫu các nút ở xa nguồn & do đó trở nên kém đại diện hơn. Đây là lý do tại sao chúng ta đang tìm kiếm sự đánh đổi giữa 2 tính chất này: tính đồng dạng có thể hữu ích hơn trong việc hiểu 1 số đồ thị & ngược lại.

If you are confused about this connection, you are not alone: several papers & blogs wrongly assume: BFS emphasizes homophily & DFS is connected to structural equivalence. In any case, consider graphs that combine homophily & structural equivalence to be desired solution. This is why, regardless of these connections, want to use both sampling strategies to create our dataset. See how can implement them to generate random walks.

– Nếu bạn đang bối rối về mối liên hệ này, bạn không phải là người duy nhất: 1 số bài báo & blog đã sai lầm khi cho rằng: BFS nhấn mạnh tính đồng dạng & DFS có liên quan đến tính tương đương về mặt cấu trúc. Trong mọi trường hợp, hãy coi các đồ thị kết hợp tính đồng dạng & tính tương đương về mặt cấu trúc là giải pháp mong muốn. Đây là lý do tại sao, bất kể những mối liên hệ này, chúng ta nên sử dụng cả hai chiến lược lấy mẫu để tạo tập dữ liệu. Hãy xem cách triển khai chúng để tạo các bước ngẫu nhiên.

* **Introducing biases in random walks.** As a reminder, random walks are sequences of nodes that are randomly selected in a graph. They have a starting point, which can also be random, & a predefined length. Nodes that often appear together in these walks are like words that appear together in sentences: under homophily hypothesis, they share a similar meaning, hence a similar representation.

– Giới thiệu về độ lệch trong các bước ngẫu nhiên. Xin nhắc lại, các bước ngẫu nhiên là chuỗi các nút được chọn ngẫu nhiên trong đồ thị. Chúng có điểm khởi đầu, cũng có thể là ngẫu nhiên, & độ dài được xác định trước. Các nút thường xuất hiện cùng nhau trong các bước này giống như các từ xuất hiện cùng nhau trong câu: theo giả thuyết đồng dạng, chúng có ý nghĩa tương tự nhau, do đó có biểu diễn tương tự nhau.

In Node2Vec, goal: bias randomness of these walks to either 1 of following:

1. Promoting nodes that are not connected to previous one (similar to DFS)
2. Promoting nodes that are close to previous one (similar to BFS)/

Current node is called j , previous node is i , & future node is k . Note π_{jk} unnormalized transition probability from node j to node k . This probability can be decomposed as $\pi_{jk} = \alpha(i, k) \cdot \omega_{jk}$, where $\alpha(i, k)$ is *search bias* between nodes i, k & ω_{jk} is weight of edge from j to k .

– Trong Node2Vec, mục tiêu: điều chỉnh độ ngẫu nhiên của các bước này thành 1 trong các bước sau:

1. Thăng hạng các nút không được kết nối với nút trước đó (tương tự như DFS)
2. Thăng hạng các nút gần nút trước đó (tương tự như BFS)/

Nút hiện tại được gọi là j , nút trước đó là i , & nút tương lai là k . Lưu ý π_{jk} xác suất chuyển tiếp không chuẩn hóa từ nút j đến nút k . Xác suất này có thể được phân tích thành $\pi_{jk} = \alpha(i, k) \cdot \omega_{jk}$, trong đó $\alpha(i, k)$ là *độ lệch tìm kiếm* giữa các nút i, k & ω_{jk} là trọng số của cạnh từ j đến k .

In DeepWalk, have $\alpha(a, b) = 1$ for any pair of nodes a, b . In Node2Vec, value of $\alpha(a, b)$ is defined based on distance between nodes & 2 additional parameters: p : return parameter & q : in-out parameter. Their role: approximate DFS & BFS, resp. Here is how value of $\alpha(a, b)$ is defined:

$$\alpha(a, b) = \begin{cases} \frac{1}{p} & \text{if } d_{ab} = 0, \\ 1 & \text{if } d_{ab} = 1, \\ \frac{1}{q} & \text{if } d_{ab} = 2. \end{cases}$$

Here d_{ab} is shortest path distance between nodes a, b . Can update unnormalized transition probability from previous graphs as follows: Fig. 4.3: Graph with transition probabilities. Decrypt these probabilities:

- Walk starts from node i & now arrive at node j . Probability of going back to previous node i is controlled by parameter p . Higher it is, more random walk will explore new nodes instead of repeating same ones & looking like DFS.
- Unnormalized probability of going to k_1 is 1 because this node is in immediate neighborhood of our previous node i .
- Finally, probability of going to node k_2 is controlled by parameter q . Higher it is, more random walk will focus on nodes that are close to previous one & look like BFS.

Best way to understand this is to actually implement this architecture & play with parameters. Do it step by step on Zachary's Karate Club.

– Ở đây d_{ab} là khoảng cách đường đi ngắn nhất giữa các nút a, b . Có thể cập nhật xác suất chuyển tiếp chưa chuẩn hóa từ các đồ thị trước đó như sau: Hình 4.3: Đồ thị với xác suất chuyển tiếp. Giải mã các xác suất này:

- Đường đi bắt đầu từ nút i & bây giờ đến nút j . Xác suất quay lại nút i trước đó được kiểm soát bởi tham số p . Giá trị p càng cao, bước đi ngẫu nhiên càng khám phá các nút mới thay vì lặp lại các nút cũ & trông giống như DFS.
- Xác suất chưa chuẩn hóa để đi đến k_1 là 1 vì nút này nằm trong vùng lân cận trực tiếp của nút i trước đó.
- Cuối cùng, xác suất đi đến nút k_2 được kiểm soát bởi tham số q . Giá trị p càng cao, bước đi ngẫu nhiên càng tập trung vào các nút gần nút trước đó & trông giống như BFS.

Cách tốt nhất để hiểu điều này là thực sự triển khai kiến trúc này & thử nghiệm với các tham số. Thực hiện từng bước trên Câu lạc bộ Karate của Zachary.

Note: it is an unweighted network, which is why transition probability is only determined by search bias. 1st, want to create a function that will randomly select next node in a graph based on previous node, current node, & 2 parameters p, q .

1. Start by importing required libraries: `networkx`, `random`, `numpy`.
2. Define `next_node` function with list of our parameters.
3. Retrieve list of neighboring nodes from current node & initialize a list of alpha values.
4. For each neighbor, want to calculate appropriate alpha value: $\frac{1}{p}$ if this neighbor is previous node, 1 if this neighbor is connected to previous node, & $\frac{1}{q}$ otherwise.
5. Normalize these values to create probabilities.
6. Randomly select next node based on transition probabilities calculated in previous step using `np.random.choice()` & return it.

Before this function can be tested, need code to generate entire random walk.

– Lưu ý: đây là 1 mạng không trọng số, đó là lý do tại sao xác suất chuyển tiếp chỉ được xác định bởi độ lệch tìm kiếm. Đầu tiên, muốn tạo 1 hàm sẽ chọn ngẫu nhiên nút tiếp theo trong đồ thị dựa trên nút trước đó, nút hiện tại, & 2 tham số p, q .

1. Bắt đầu bằng cách nhập các thư viện cần thiết: `networkx`, `random`, `numpy`.
2. Định nghĩa hàm `next_node` với danh sách các tham số của chúng ta.
3. Truy xuất danh sách các nút lân cận từ nút hiện tại & khởi tạo danh sách các giá trị alpha.
4. Đối với mỗi nút lân cận, muốn tính giá trị alpha phù hợp: $\frac{1}{p}$ nếu nút lân cận này là nút trước đó, 1 nếu nút lân cận này được kết nối với nút trước đó, & $\frac{1}{q}$ nếu không.
5. Chuẩn hóa các giá trị này để tạo xác suất.
6. Chọn ngẫu nhiên nút tiếp theo dựa trên xác suất chuyển tiếp được tính toán ở bước trước bằng cách sử dụng `np.random.choice()` & trả về kết quả.

Trước khi có thể kiểm tra hàm này, cần mã để tạo toàn bộ bước ngẫu nhiên.

Way we generate these random walks is similar to what we saw in previous chap. Difference: next node is chosen by `next_node()` function, which requires additional parameters: p, q , but also previous & current nodes. These nodes can easily be obtained by looking at 2 last elements added to `walk` variable. Also return strings instead of integers for compatibility reasons. Here is new version of `random_walk()` function.

– Cách chúng ta tạo ra các bước ngẫu nhiên này tương tự như những gì chúng ta đã thấy trong chương trước. Điểm khác biệt: nút tiếp theo được chọn bởi hàm `next_node()`, hàm này yêu cầu các tham số bổ sung: p, q , nhưng cũng bao gồm các nút trước đó & nút hiện tại. Các nút này có thể dễ dàng được lấy bằng cách xem xét 2 phần tử cuối cùng được thêm vào biến `walk`. Đồng thời, trả về chuỗi thay vì số nguyên vì lý do tương thích. Dưới đây là phiên bản mới của hàm `random_walk()`.

Now have every element to generate our random walks. Try one with a length of 5, $p = q = 1$: `random_walk(0, 8, p = 1, q = 1)`

This should be random since every neighboring node has same transition probability. With these parameters, reproduce exact DeepWalk algorithm. Now bias them toward going back to previous node with $q = 10$: `random_walk(0, 8, p = 1, q = 10)`.

This time, random walk explores more node in graph, can see that it never goes back to previous node because probability is low with $p = 10$: `random_walk(0, 8, p = 10, q = 1)`. See how to use these properties in a real example & compare it to DeepWalk.

– Bây giờ, hãy tạo ra mọi phần tử để tạo ra các bước đi ngẫu nhiên. Hãy thử một bước đi có độ dài 5, $p = q = 1$: `random_walk(0, 8, p = 1, q = 1)`. Bước đi này phải ngẫu nhiên vì mọi nút lân cận đều có cùng xác suất chuyển tiếp. Với các tham số này, hãy tái tạo chính xác thuật toán DeepWalk. Bây giờ, hãy hướng chúng về phía nút trước đó với $q = 10$: `random_walk(0, 8, p = 1, q = 10)`. Lần này, bước đi ngẫu nhiên khám phá nhiều nút hơn trong đồ thị, có thể thấy rằng nó không bao giờ quay lại nút trước đó vì xác suất thấp với $p = 10$: `random_walk(0, 8, p = 10, q = 1)`. Xem cách sử dụng các thuộc tính này trong một ví dụ thực tế & so sánh nó với DeepWalk.

- **Implementing Node2Vec.** Now have function to generate biased random walks, implementation of Node2Vec is very similar to implementing DeepWalk. It is so similar that we can reuse same code & create sequences with $p = q = 1$ to implement DeepWalk as a special case of Node2Vec. Reuse Zachary's Karate Club for this task. As in prev chap, goal: correctly classify each member of club as part of 1 of 2 groups ("Mr. Hi" & "Officer"). Use node embeddings provided by Node2Vec as input to a ML classifier (Random Forest in this case). See how to implement it step by step:

1. 1st, want to install `gensim` library to use Word2Vec. This time, use version 3.8.0 for compatibility reasons: `!pip install -qI gensim`
2. Import required libraries.
3. Load dataset (Zachary's Karate Club).
4. Transform nodes' labels into numerical values: 0 & 1.
5. Generate a list of random walks as seen previously using `random_walk()` function 80 times for each node in graph. Parameters p, q as specified here (2 & 1, resp)

- Building a move RecSys.

- 5. Including Node Features with Vanilla Neural Networks.
- 6. Introducing Graph Convolutional Networks.
- 7. Graph Attention Networks.

PART 3: ADVANCED TECHNIQUES.

- 8. Scaling Up Graph Neural Networks with GraphSAGE.
- 9. Defining Expressiveness for Graph Classification.
- 10. Predicting Links with Graph Neural Networks.
- 11. Generating Graphs Using Graph Neural Networks.
- 12. Learning from Heterogeneous Graphs.
- 13. Temporal Graph Neural Networks.
- 14. Explaining Graph Neural Networks.

PART 4: APPLICATIONS.

- 15. Forecasting Traffic Using A3T-GCN.
- 16. Detecting Anomalies Using Heterogeneous GNNs.
- 17. Building a Recommender System Using LightGCN.
- 18. Unlocking the Potential of Graph Neural Networks for Real-World Applications.

2 Miscellaneous