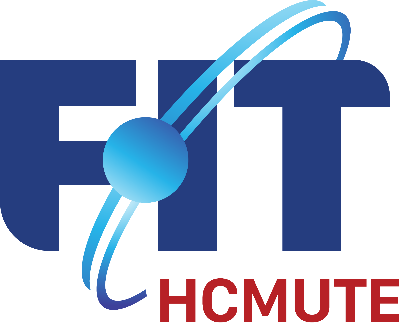
**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN KỸ THUẬT DỮ LIỆU**



**NGUYỄN THANH BÌNH - 20133025**

**NGUYỄN NHẬT TRIỀU - 20133102**

ĐỀ TÀI

**TÌM HIỂU GRAPH NEURAL NETWORK VÀ ỨNG DỤNG**

**TIỂU LUẬN CHUYÊN NGÀNH**

**KỸ SƯ KỸ THUẬT DỮ LIỆU**

**GIÁO VIÊN HƯỚNG DẪN**

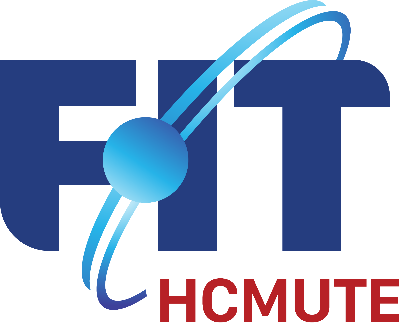
**ThS. QUÁCH ĐÌNH HOÀNG**

**KHÓA 2020 - 2024**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN KỸ THUẬT DỮ LIỆU**



**NGUYỄN THANH BÌNH - 20133025**

**NGUYỄN NHẬT TRIỀU - 20133102**

ĐỀ TÀI

**TÌM HIỂU GRAPH NEURAL NETWORK VÀ ỨNG DỤNG**

**TIỂU LUẬN CHUYÊN NGÀNH**

**KỸ SƯ KỸ THUẬT DỮ LIỆU**

**GIÁO VIÊN HƯỚNG DẪN**

**ThS. QUÁCH ĐÌNH HOÀNG**

**KHÓA 2020 - 2024**

|  |  |
| --- | --- |
| **ĐH SƯ PHẠM KỸ THUẬT TP.HCM**  **KHOA CNTT**  \*\*\*\*\*\*\* | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc Độc lập – Tự do – Hạnh Phúc**  \*\*\*\*\*\*\* |

**PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

Họ và tên Sinh viên 1: Nguyễn Thanh Bình MSSV 1: 20133025

Họ và tên Sinh viên 2: Nguyễn Nhật Triều MSSV 2: 20133102

Ngành: Kỹ thuật dữ liệu

Tên đề tài: Tìm hiểu về Graph Neural Network và ứng dụng

Họ và tên Giảng viên hướng dẫn: ThS. Quách Đình Hoàng

**NHẬN XÉT**

1. Về nội dung đề tài & khối lượng thực hiện:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Ưu điểm:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Khuyết điểm:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Đề nghị cho bảo vệ hay không: ……………………………………………………….
2. Đánh giá loại: ………………………………………………………………………….
3. Điểm: ………………………………………………………………………………….

*Tp. Hồ Chí Minh, ngày tháng năm 202*

Giảng viên hướng dẫn

*(Ký & ghi rõ họ tên)*

|  |  |
| --- | --- |
| **ĐH SƯ PHẠM KỸ THUẬT TP.HCM**  **KHOA CNTT**  \*\*\*\*\*\*\* | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc Độc lập – Tự do – Hạnh Phúc**  \*\*\*\*\*\*\* |

**PHIẾU NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN**

Họ và tên Sinh viên 1: Nguyễn Thanh Bình MSSV 1: 20133025

Họ và tên Sinh viên 2: Nguyễn Nhật Triều MSSV 2: 20133102

Ngành: Kỹ thuật dữ liệu

Tên đề tài: Tìm hiểu về Graph Neural Network và ứng dụng

Họ và tên Giảng viên phản biện: TS. Hoàng Văn Dũng

**NHẬN XÉT**

1. Về nội dung đề tài & khối lượng thực hiện:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Ưu điểm:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Khuyết điểm:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Đề nghị cho bảo vệ hay không: ……………………………………………………….
2. Đánh giá loại: ………………………………………………………………………….
3. Điểm: ………………………………………………………………………………….

*Tp. Hồ Chí Minh, ngày tháng năm 202*

Giáo viên phản biện

*(Ký & ghi rõ họ tên)*

**LỜI CẢM ƠN**

Lời đầu tiên, nhóm chúng em xin được bày tỏ lòng biết ơn sâu sắc đến tất cả thầy cô trong trường đại học Sư Phạm Kỹ Thuật Thành phố Hồ Chí Minh đã tạo điều kiện và hỗ trợ tốt nhất cho nhóm chúng em trong suốt quá trình học tập vừa qua.

Đồng thời, nhóm chúng em xin chân thành gởi lời cảm ơn sâu sắc đến các thầy cô trong khoa Công nghệ Thông tin - trường Đại học Sư Phạm Kỹ Thuật Thành Phố Hồ Chí Minh đã cung cấp nhiều kiến thức bổ ích để nhóm chúng em có thể thuận lợi nâng cao sự hiểu biết của bản thân.

Đặc biệt, nhóm chúng em xin chân thành gửi lời cảm ơn tới thầy Quách Đình Hoàng - giáo viên hướng dẫn, người đã trực tiếp giúp đỡ, chỉ dẫn tận tình và hướng dẫn nhóm chúng em hoàn thành bài tiểu luận chuyên ngành trong thời gian qua.

Trong quá trình hoàn thành, nhóm chúng em đã có những cố gắng nhất định, song do kinh nghiệm và kiến thức còn nhiều thiếu sót, dẫn đến bài báo cáo còn nhiều hạn chế nhất định. Vì vậy, nhóm chúng em kính mong nhận được những phản hồi, đóng góp ý kiến và chỉ bảo thêm từ quý thầy cô nói để nhóm chúng em có thể rút ra được nhiều bài học quý giá từ đó để áp dụng vào cuộc sống sau này.

Nhóm chúng em xin chân thành cảm ơn!

|  |  |
| --- | --- |
| **ĐH SƯ PHẠM KỸ THUẬT TP.HCM**  **KHOA CNTT**  \*\*\*\*\*\*\* | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc Độc lập – Tự do – Hạnh Phúc**  \*\*\*\*\*\*\* |

**ĐỀ CƯƠNG TIỂU LUẬN CHUYÊN NGÀNH**

Họ và tên Sinh viên 1: Nguyễn Thanh Bình MSSV 1: 20133025

Họ và tên Sinh viên 2: Nguyễn Nhật Triều MSSV 2: 20133102

Thời gian làm luận văn từ 09/2023 đến 12/2023

Chuyên ngành: Kỹ thuật dữ liệu

Tên đề tài: Tìm hiểu về Graph Neural Network và ứng dụng

Họ và tên Giảng viên hướng dẫn: ThS. Quách Đình Hoàng

**Nhiệm vụ thực hiện:**

* + - 1. Tìm hiểu về lý thuyến đồ thị
      2. Tìm hiểu về Neural Network
      3. Tìm hiểu về học biểu diễn
      4. Tìm hiểu về Graph Neural Network
      5. Tìm hiểu về các mô hình cụ thể của GNN: GCN, GraphSAGE, GAT
      6. Tìm hiểu về bài toán Link Prediction
      7. Ứng dụng bài toán Link Prediction cho các mô hình của GNN
      8. Nhận xét kết quả và định hướng phát triển

**Đề cương viết tiểu luận:**

CHƯƠNG 1: MỞ ĐẦU

1. LÝ DO CHỌN ĐỀ TÀI

2. MỤC TIÊU CỦA ĐỀ TÀI

2.1. Mục tiêu của đề tài

2.2. Mục tiêu thực hiện

3. PHƯƠNG PHÁP NGHIÊN CỨU

3.1. Về mặt lý thuyết

3.2. Đối tượng nghiên cứu

3.3. Phạm vi nghiên cứu

4. KẾT QUẢ DỰ KIẾN ĐẠT ĐƯỢC

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

1. MỘT SỐ KIẾN THỨC TỔNG QUAN VỀ LÝ THUYẾT ĐỒ THỊ

1.1. KHÁI NIỆM

1.2. CÁC LOẠI ĐỒ THỊ

1.3. MỘT SỐ THUẬT NGỮ KHÁC CỦA ĐỒ THỊ

1.4. BIỂU DIỄN ĐỒ THỊ

1.5. CÁC THAO TÁC VỚI ĐỒ THỊ

2. MẠNG NEURAL (NEURAL NETWORK)

2.1. KHÁI NIỆM

2.2. CẤU TRÚC CỦA NEURAL NETWORK

2.3. MỘT SỐ HÀM KÍCH HOẠT VÀ HÀM LỖI

2.3.1. Hàm Sigmoid

2.3.2. Hàm Tanh

2.3.3. Hàm Relu

2.3.4. Hàm Leaky ReLU

2.3.5. Hàm Softmax

2.3.6. Hàm lỗi Cross-entropy

2.4. LAN TRUYỀN XUÔI (FEEDFORWARD)

2.5. LAN TRUYỀN NGƯỢC (BACK PROPAGATION)

2.6. ỨNG DỤNG CỦA NEURAL NETWORK

3. TỔNG QUÁT VỀ HỌC BIỂU DIỄN

3.1. HỌC BIỂU DIỄN

3.1.1. Khái niệm

3.1.2. Cách phương pháp của học biểu diễn

3.2. HỌC BIỂU DIỄN ĐỒ THỊ

3.3. GRAPH EMBEDDING

3.3.1. Định nghĩa

3.3.2. Một số phương pháp của Graph Embedding

4. GRAPH NEURAL NETWORK

4.1. GIỚI THIỆU VỀ GRAPH NEURAL NETWORK

4.1.1. Lịch sử hình thành và phát triển của Graph Neural Network

4.1.2. Định nghĩa

4.1.3. Kiến trúc cơ bản trong Graph Neural Network

4.2. MỘT SỐ MÔ HÌNH CỤ THỂ CỦA GRAPH NEURAL NETWORK

4.2.1. Graph Convolution Network

4.2.2. GraphSAGE

4.2.3. Graph Attention Network

4.3. NHƯỢC ĐIỂM CỦA GRAPH NEURAL NETWORK

4.4. ỨNG DỤNG CỦA GRAPH NEURAL NETWORK

4.5. ĐỊNH HƯỚNG TƯƠNG LAI

CHƯƠNG 3: THỰC NGHIỆM VÀ ĐÁNH GIÁ

1. BÀI TOÁN PHÂN LOẠI NÚT (NODE CLASSIFICATION)

1.1. GIỚI THIỆU VỀ BÀI TOÁN

1.2. CÁC VẤN ĐỀ GẶP PHẢI CỦA BÀI TOÁN

1.3. CÁC ỨNG DỤNG CỦA BÀI TOÁN NODE CLASSIFICATION

2. GIỚI THIỆU TẬP DỮ LIỆU

2.1. TẬP DỮ LIỆU CITESEER

2.2. TẬP DỮ LIỆU CORA

2.3. TẬP DỮ LIỆU PUBMED

2.4. TẬP DỮ LIỆU TWITCH

3. CÁC THƯ VIỆN CẦN THIẾT

3.1. MÔI TRƯỜNG

3.2. THƯ VIỆN CÀI ĐẶT

3.2.1. Thư viện Streamlit

3.2.2. Thư viện Pytorch

3.2.3. Thư viện Matplotlib

3.2.4. Thư viện Plotly

3.2.5. Thư viện numpy

3.2.6. Thư viện Pandas

3.2.7. Thư viện StellarGraph

3.2.8. Thư viện Seaborn

3.2.9. Thư viện networkx

4. THỰC NGHIỆM

4.1. CHIA DỮ LIỆU

4.1.1. Tập dữ liệu CiteSeer

4.1.2. Tập dữ liệu Cora

4.1.3. Tập dữ liệu PubMed

4.1.4. Tập dữ liệu Twitch

4.2. THỰC HIỆN EXPLORATORY DATA ANALYSIS (EDA)

4.2.1. Biểu đồ phân bố số lượng nút theo từng lớp

4.2.2. Biểu đồ thể hiện mức độ kết nối giữa các lớp

4.2.3. Biểu đồ phân bố số lượng nút theo bậc

4.2.4. Biểu đồ phân số số lượng đặc trưng trên tập dữ liệu

4.3. HUẤN LUYỆN MÔ HÌNH

4.3.1. Mô hình GCN

4.3.2. Mô hình GraphSAGE

4.3.3. Mô hình GAT

4.4. ĐÁNH GIÁ VÀ NHẬN XÉT KẾT QUẢ

4.4.1. Hidden Channels (kích thước của lớp ẩn)

4.4.2. Learning Rate (tốc độ học)

4.4.3. Epochs (số lượng vòng lặp)

CHƯƠNG 4: KẾT LUẬN

1. KẾT LUẬN

2. HẠN CHẾ

3. HƯỚNG PHÁT TRIỂN

**KẾT HOẠCH THỰC HIỆN**

|  |  |  |
| --- | --- | --- |
| STT | Các nội dung, công việc thực hiện chủ yếu | Khoảng thời gian |
| 1 | Nghiên cứu các vấn đề lý thuyết nền tảng của mạng nơ-ron đồ thị | Từ ngày 04/09/2023 đến ngày 25/09/2023 |
| 2 | Nghiên cứu các phương pháp của mạng nơ-ron đồ thị | Từ ngày 26/09/2023 đến ngày 17/10/2023 |
| 3 | Tìm hiểu các biến thể của gnn, và sự phát triển của từng loại biến thể | Từ ngày 18/10/2023 đến ngày 1/11/2023 |
| 4 | Nghiên cứu các ứng dụng của mạng nơ-ron đồ thị | Từ ngày 2/11/2023 đến ngày 10/11/2023 |
| 5 | Thực nghiệm một ứng dụng cụ thể của mạng nơ-ron đồ thị | Từ ngày 11/11/2023 đến ngày 3/12/2023 |
| 6 | Viết báo cáo tổng kết | Từ ngày 4/12/2023 đến ngày 15/12/2023 |

|  |  |
| --- | --- |
| Ý kiến của giáo viên hướng dẫn  *(Ký và ghi rõ họ tên)* | Ngày tháng năm  Người viết đề cương |
| ThS. QUÁCH ĐÌNH HOÀNG | Nguyễn Thanh Bình  Nguyễn Nhật Triều |

MỤC LỤC

[CHƯƠNG 1: MỞ ĐẦU 15](#_Toc166545942)

[1. LÝ DO CHỌN ĐỀ TÀI 15](#_Toc166545943)

[2. MỤC TIÊU CỦA ĐỀ TÀI 16](#_Toc166545944)

[2.1. Mục tiêu của đề tài 16](#_Toc166545945)

[2.2. Mục tiêu thực hiện 16](#_Toc166545946)

[3. PHƯƠNG PHÁP NGHIÊN CỨU 16](#_Toc166545947)

[3.1. Về mặt lý thuyết 16](#_Toc166545948)

[3.2. Đối tượng nghiên cứu 17](#_Toc166545949)

[3.3. Phạm vi nghiên cứu 17](#_Toc166545950)

[4. KẾT QUẢ DỰ KIẾN ĐẠT ĐƯỢC 17](#_Toc166545951)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT 18](#_Toc166545952)

[1. TỔNG QUAN VỀ HỌC MÁY (MACHINE LEARNING) 18](#_Toc166545953)

[1.1. Học máy 18](#_Toc166545954)

[1.1.1. Giới thiệu về học máy 18](#_Toc166545955)

[1.1.2. Các phương pháp của học máy 18](#_Toc166545956)

[1.2. Học sâu (Deep learning) 19](#_Toc166545957)

[1.2.1. Giới thiệu về học sâu 19](#_Toc166545958)

[1.2.2. Một số đặc điểm của học sâu 19](#_Toc166545959)

[2. LÝ THUYẾT CƠ BẢN LIÊN QUAN ĐẾN ĐỒ THỊ 20](#_Toc166545960)

[2.1. Lý thuyết đồ thị cơ bản 20](#_Toc166545963)

[2.2. Mạng Neural (Neural Network) cơ bản 22](#_Toc166545964)

[3. TỔNG QUÁT VỀ HỌC BIỂU DIỄN (REPRESENTATION LEARNING) 23](#_Toc166545965)

[3.1. Học biểu diễn 24](#_Toc166545966)

[3.1.1. Khái niệm 24](#_Toc166545967)

[3.1.2. Ứng dụng của học biểu diễn 24](#_Toc166545968)

[3.1.3. Thách thức của học biểu diễn 24](#_Toc166545969)

[3.2. Học biểu diễn đồ thị (Graph representation learning) 24](#_Toc166545970)

[3.2.1. Khái niệm 24](#_Toc166545971)

[3.2.2. Một số phương pháp của học biểu diễn đồ thị 25](#_Toc166545972)

[4. GRAPH NEURAL NETWORKS (GNNs) 26](#_Toc166545973)

[4.1. Giới thiệu về Graph Neural Networks 26](#_Toc166545974)

[4.1.1. Lịch sử hình thành của Graph Neural Networks 26](#_Toc166545975)

[4.1.2. Định nghĩa về Graph Neural Networks 27](#_Toc166545976)

[4.2. Các kiến trúc phổ biến của Graph Neural Networks 27](#_Toc166545977)

[4.2.1. Graph Convolutional Networks (GCNs) 27](#_Toc166545978)

[4.2.2. Graph Sample and Aggregation (GraphSAGE) 29](#_Toc166545979)

[4.2.3. Graph Attention Networks (GATs) 31](#_Toc166545980)

[4.3. Ứng dụng của Graph Neural Networks 34](#_Toc166545981)

[4.4. Hạn chế của Graph Neural Networks 34](#_Toc166545982)

[4.5. Tương lai của Graph Neural Networks 35](#_Toc166545983)

[CHƯƠNG 3: ỨNG DỤNG 35](#_Toc166545984)

[1. GIỚI THIỆU VỀ BÀI TOÁN LINK PREDICTION 35](#_Toc166545985)

[1.1. Định nghĩa bài toán Link Prediction trong mạng đồ thị 35](#_Toc166545986)

[1.2. Cách tiếp cận và phương pháp 36](#_Toc166545987)

[1.2.1. Các phương pháp truyền thống 36](#_Toc166545988)

[1.2.2. Áp dụng Graph Neural Networks cho bài toán Link prediction 38](#_Toc166545989)

[1.3. Ứng dụng của bài toán Link prediction 42](#_Toc166545990)

[1.4. Các thách thức của bài toán Link prediction 42](#_Toc166545991)

[2. THỰC NGHIỆM VÀ ĐÁNH GIÁ 43](#_Toc166545992)

[2.1. Môi trường cần thiết 43](#_Toc166545993)

[2.2. Mô tả dữ liệu 44](#_Toc166545994)

[2.3. Xử lý dữ liệu 45](#_Toc166545995)

[2.3.1. SEAL Framework 45](#_Toc166545996)

[2.3.2. TwoWL 48](#_Toc166545997)

[2.4. Xây dựng mô hình 50](#_Toc166545998)

[2.4.1. SEAL Framework 50](#_Toc166545999)

[2.4.2. Mô hình 2WL 54](#_Toc166546000)

[2.5. Huấn luyện mô hình 55](#_Toc166546001)

[2.5.1. SEAL Framework 55](#_Toc166546002)

[2.5.2. TwoWL 56](#_Toc166546003)

[2.6. Dự đoán kết quả 58](#_Toc166546004)

[2.6.1. SEAL Framework 58](#_Toc166546005)

[2.6.2. Two-WL 59](#_Toc166546006)

[2.7. Đánh giá và nhận xét kết quả 60](#_Toc166546007)

[2.7.1. SEAL Framework 60](#_Toc166546008)

[2.7.2. TwoWL 61](#_Toc166546009)

[2.7.3. So sánh 61](#_Toc166546010)

[CHƯƠNG 4: KẾT LUẬN 62](#_Toc166546011)

[1. KẾT LUẬN 62](#_Toc166546012)

[2. HẠN CHẾ 63](#_Toc166546013)

[3. HƯỚNG PHÁT TRIỂN 63](#_Toc166546014)

[CHƯƠNG 5: TÀI LIỆU THAM KHẢO 63](#_Toc166546015)

**PHỤ LỤC HÌNH ẢNH**

[Hình 1. Hình ảnh mình họa về một đồ thị [1] **Error! Bookmark not defined.**](#_Toc153657230)

[Hình 2. Hình ảnh minh họa về đồ thị vô hướng [1] **Error! Bookmark not defined.**](#_Toc153657231)

[Hình 3. Ví dụ về đồ thị có hướng [1] **Error! Bookmark not defined.**](#_Toc153657232)

[Hình 4. Hình ảnh minh họa cách biểu diễn đồ thị sang ma trận kề [3] **Error! Bookmark not defined.**](#_Toc153657233)

[Hình 5. Hình ảnh minh họa cách biểu diễn đồ thị sang danh sách kề [3] **Error! Bookmark not defined.**](#_Toc153657234)

[Hình 6. Hình ảnh minh họa về cấu trúc của một Neural Network **Error! Bookmark not defined.**](#_Toc153657235)

[Hình 7. Hình ảnh minh họa về giá trị đầu ra của hàm Sigmoid [5] **Error! Bookmark not defined.**](#_Toc153657236)

[Hình 8. Hình ảnh minh họa về giá trị đầu ra của hàm Tanh [5] **Error! Bookmark not defined.**](#_Toc153657237)

[Hình 9. Hình ảnh minh họa về giá trị đầu ra của hàm Relu [5] **Error! Bookmark not defined.**](#_Toc153657238)

[Hình 10. Hình ảnh minh họa về giá trị đầu ra của hàm Leaky ReLU [5] **Error! Bookmark not defined.**](#_Toc153657239)

[Hình 11. Hình ảnh minh họa về giá trị đầu ra của hàm Softmax [5] **Error! Bookmark not defined.**](#_Toc153657240)

[Hình 12. Hình ảnh minh họa về các hệ số của một mạng Neural Network [6] **Error! Bookmark not defined.**](#_Toc153657241)

[Hình 13. Hình ảnh minh họa quá trình tính toán của lan truyền xuôi [6] **Error! Bookmark not defined.**](#_Toc153657242)

[Hình 14. Hình ảnh minh họa về các lớp trong Graph Neural Network [11] **Error! Bookmark not defined.**](#_Toc153657243)

[Hình 15. Hình ảnh minh họa chi tiết mô hình lan truyền DGCN **Error! Bookmark not defined.**](#_Toc153657244)

[Hình 16. Hình ảnh minh họa cách lan truyền giữa các lớp trong Graph Convolutions Network **Error! Bookmark not defined.**](#_Toc153657245)

[Hình 17. Hình ảnh minh họa cách hoạt động của thuật toán tạo nhúng [14] 2](#_Toc153657246)

[Hình 18. Hình ảnh thể hiện cách hoạt động của GAT [13] **Error! Bookmark not defined.**](#_Toc153657247)

[Hình 19. Biểu đồ của đồ thị dựa trên dữ liệu CiteSeer **Error! Bookmark not defined.**](#_Toc153657248)

[Hình 20. Biểu đồ của đồ thị dựa trên dữ liệu Cora **Error! Bookmark not defined.**](#_Toc153657249)

[Hình 21. Biểu đồ của đồ thị dựa trên dữ liệu PubMed **Error! Bookmark not defined.**](#_Toc153657250)

[Hình 22. Hình ảnh thể hiện thông tin của một số cạnh trong tập dữ liệu Twitch ở Gremary **Error! Bookmark not defined.**](#_Toc153657251)

[Hình 23. Hình ảnh thể hiện các tập dữ liệu của CiteSeer **Error! Bookmark not defined.**](#_Toc153657252)

[Hình 24. Hình ảnh thể hiện các tập dữ liệu của Cora **Error! Bookmark not defined.**](#_Toc153657253)

[Hình 25. Hình ảnh thể hiện các tập dữ liệu trong tập PubMed **Error! Bookmark not defined.**](#_Toc153657254)

[Hình 26. Hình ảnh thể hiện đoạn mã chia dữ liệu **Error! Bookmark not defined.**](#_Toc153657255)

[Hình 27. Biểu đồ thể hiện sự phân bố số lượng nút trong tập dữ liệu CiteSeer **Error! Bookmark not defined.**](#_Toc153657256)

[Hình 28. Biểu đồ thể hiện mức độ kết nối giữa các lớp trong tập dữ liệu CiteSeer **Error! Bookmark not defined.**](#_Toc153657257)

[Hình 29. Biểu đồ thể hiện sự phân bố số lượng nút theo bậc của tập dữ liệu CiteSeer **Error! Bookmark not defined.**](#_Toc153657258)

[Hình 30. Biểu đồ thể hiện sự phân bố số lượng đặc trưng trên tập dữ liệu Twitch - ENGB **Error! Bookmark not defined.**](#_Toc153657259)

[Hình 31.Hình ảnh thể hiện đoạn mã khởi tạo mô hình GCN **Error! Bookmark not defined.**](#_Toc153657260)

[Hình 32. Hình ảnh thẻ hiện đoạn mã của quá trình lan truyền tiến của mô hình GCN **Error! Bookmark not defined.**](#_Toc153657261)

[Hình 33. Hình ảnh đoạn mã thể hiện quá trình huấn luyện của mô hình GCN **Error! Bookmark not defined.**](#_Toc153657262)

[Hình 34. Hình ảnh đoạn mã thể hiện quá trình đánh giá và in thông tin của mô hình GCN **Error! Bookmark not defined.**](#_Toc153657263)

[Hình 35. Hình ảnh đoạn mã thể hiện quá trình khởi tạo mô hình GraphSage **Error! Bookmark not defined.**](#_Toc153657264)

[Hình 36. Hình ảnh đoạn mã thể hiện quá trình lan truyền tiến của mô hình GraphSage **Error! Bookmark not defined.**](#_Toc153657265)

[Hình 37. Hình ảnh đoạn mã quá trình huấn luyện của mô hình GraphSage **Error! Bookmark not defined.**](#_Toc153657266)

[Hình 38. Hình ảnh đoạn mã thể hiện quá trình đánh giá và in thông tin của mô hình GraphSage **Error! Bookmark not defined.**](#_Toc153657267)

[Hình 39. Hình ảnh đoạn mã quá trình khởi tạo mô hình GAT **Error! Bookmark not defined.**](#_Toc153657268)

[Hình 40. Hình ảnh đoạn mã thể hiện quá trình lan truyền tiến của mô hình GAT **Error! Bookmark not defined.**](#_Toc153657269)

[Hình 41.Hình ảnh đoạn mã quá trình huấn luyện của mô hình GAT **Error! Bookmark not defined.**](#_Toc153657270)

[Hình 42. Hình ảnh đoạn mã thể hiện quá trình đánh giá và in thông tin của mô hình GAT **Error! Bookmark not defined.**](#_Toc153657271)

**DANH SÁCH BẢNG**

[Bảng 1. Bảng so sánh sự khác nhau của từng mô hình khi thay đổi giá trị hidden channels **Error! Bookmark not defined.**](#_Toc153654361)

[Bảng 2. Bảng so sánh sự khác nhau của từng mô hình khi thay đổi giá trị learning rate **Error! Bookmark not defined.**](#_Toc153654362)

[Bảng 3. Bảng só sánh sự khác nhau của từng mô hình khi thay đổi giá trị epochs **Error! Bookmark not defined.**](#_Toc153654363)

**DANH MỤC TỪ VIẾT TẮT**

|  |  |
| --- | --- |
| GNN | Graph Neural Network |
| GCN | Graph Convolutional Network |
| CNN | Convolution Neural Network |
| GraphSAGE | Graph Sample Aggregate |
| GAT | Graph Attention Network |
| Leaky ReLU | Leaky Rectified Linear Units |
| ReLU | Rectified Linear Units |

**CHƯƠNG 1: MỞ ĐẦU**

1. **LÝ DO CHỌN ĐỀ TÀI**

Với sự phát triển mạnh mẽ của công nghệ số hiện nay, đặc biệt là sự bùng nổ của trí tuệ nhân tạo đã đặt ra nhiều thách thức và cơ hội mới cho con người, trong đó có việc xử lý và phân tích các loại dữ liệu phức tạp. Một trong những mô hình nổi bật trong lĩnh vực xử lý và phân tích dữ liệu này là Graph Neural Networks (GNNs).

Graph Neural Networks là một mô hình học máy mạnh mẽ áp dụng các phương pháp tiên tiến để phân tích và xử lý dữ liệu dưới dạng đồ thị. Graph Neural Network cho phép chúng ta tìm hiểu cấu trúc và mối quan hệ phức tạp giữa các yếu tố trong dữ liệu đồ thị và từ đó tạo ra các ứng dụng thông minh để áp dụng vào cuộc sống sau này.

Trong phạm vi của chủ đề này, chúng ta sẽ khám phá Graph Neural Networks từ các khía cạnh cơ bản khác nhau như tìm hiểu về các kiến thức cơ sở liên quan, lịch sử ra đời, khái niệm, cũng như kiến trúc và các hoạt động của từng mô hình cụ thể trong Graph Neural Networks, đồng thời áp dụng các kiến thức đã tìm hiểu về Graph Neural Networks để thực hiện xây dựng các mô hình cụ thể của Graph Neural Network trên các tập dữ liệu khác nhau.

Qua việc tìm hiểu về Graph Neural Networks và ứng dụng nó với các tập dữ liệu khác nhau, chúng ta có thể đánh giá được khả năng và tiềm năng của Graph Neural Network trong việc hiểu và tận dụng thông tin từ dữ liệu đồ thị, từ đó hỗ trợ chúng ta trong nhiều công việc hiện nay. Chính vì lý do đó, việc chọn đề tài: *“Tìm hiểu về Graph Neural Networks và ứng dụng”* là một đề tài thú vị và mang tính ứng dụng cao.

1. **MỤC TIÊU CỦA ĐỀ TÀI**
   1. **Mục tiêu của đề tài**

Dựa trên những tài liệu đã tìm hiểu và nghiên cứu, nhóm đã có cái nhìn tổng quan về Graph Neural Networks. Đồng thời, nhóm cũng thực hiện việc xây dựng chương trình áp dụng các mô hình cụ thể của Graph Neural Networks với các tập dữ liệu khác nhau cho bài toán phân loại nút, sau đó đưa ra nhận xét.

* 1. **Mục tiêu thực hiện**
* Mục tiêu 1: Nắm được kiến thức cơ bản về dữ liệu đồ thị cũng như Graph Neural Network dựa trên các bài báo, bài viết ở các nguồn đáng tin cậy.
* Mục tiêu 2: Xây dựng được một chương trình cụ thể áp dụng được các mô hình cụ thể của Graph Neural Network với các tập dữ liệu khác nhau từ các kiến thức đã tìm hiểu và nghiên cứu về Graph Neural Network. Qua đó, nhóm đưa ra các nhận xét và đánh giá.

1. **PHƯƠNG PHÁP NGHIÊN CỨU**
   1. **Về mặt lý thuyết**

* Nghiên cứu, tổng hợp các tài liệu liên quan đến Graph Neural Network và dữ liệu đồ thị.
* Nghiên cứu, tổng hợp các các tài liệu liên quan đến việc xây dựng một chương trình thực nghiệm áp dụng các mô hình cụ thể của Graph Neural Network.
  1. **Đối tượng nghiên cứu**
* Dữ liệu đồ thị.
* Mô hình Neural Network.
* Học biểu diễn.
* Mô hình Graph Neural Network.
  1. **Phạm vi nghiên cứu**
* Nghiên cứu các tài liệu xoay quanh Graph Neural Network, cũng như các mô hình cụ thể của Graph Neural Network (Graph Convolutional Network, GraphSAGE, Graph Attention Network).

1. **KẾT QUẢ DỰ KIẾN ĐẠT ĐƯỢC**

* Về mặt lý thuyết:
* Nắm được các kiến thức cơ bản về dữ liệu đồ thị, cách phân tích và xử lý.
* Nắm được các kiến thức cơ bản xoay quanh Graph Neural Network.
* Về mặt ứng dụng:
* Xây dựng được một chương trình áp dụng được các mô hình cụ thể của Graph Neural Network vào các tập dữ liệu khác nhau.

# **CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**

**TỔNG QUAN VỀ HỌC MÁY (MACHINE LEARNING)**

* 1. **Học máy**
     1. **Giới thiệu về học máy**

Học máy là một lĩnh vực của trí tuệ nhân tạo và khoa học máy tính, nó tập trung vào việc nghiên cứu và xây dựng các kĩ thuật cho phép các hệ thống “học” tự động từ dữ liệu để giải quyết những vấn đề cụ thể.

Các thuật toán trong học máy sẽ dựa trên dữ liệu mẫu (dữ liệu mẫu đó được gọi là dữ liệu huấn luyện) để xây dựng một mô hình cụ thể, rồi từ đó mô hình đưa ra các dự đoán hoặc quyết định mà không cần được lập trình chi tiết về vấn đề đó.

Ví dụ, từ việc phân loại rác thải thành rác có thể tái chế hoặc không, ta có thể xây dựng một mô hình học máy phân loại các loại rác đó khi ta dùng các loại rác thải đã được phân loại thành các nhóm như có thể tái chế hoặc không thể tái chế để tiến hành huấn luyện mô hình, rồi từ đó mô hình có thể học được đặc điểm của các loại rác này và tiến hành tự động phân loại. (\*)

* + 1. **Các phương pháp của học máy**
       1. **Học có giám sát (Supervised learning)**

- Học có giám sát đã có thành công lớn trong các ứng dụng trong thực tế. Ở phương pháp này, ta sẽ sử dụng các tập dữ liệu được gán nhãn để huấn luyện các thuật toán phân loại dữ liệu hoặc dự đoán kết quả một cách chính xác.

- Dữ liệu có gán nhãn sẽ được đưa vào mô hình, mô hình sẽ tự động điều chỉnh đến khi kết quả từ dữ liệu huấn luyện khi đưa vào mô hình giống với dữ liệu thực tế nhất có thể.

- Một số mô hình được sử dụng trong học có giám sát có thể kể đến như: logistic regression, neural networks, linear regression, naive bayes, random forest, support vector machine (SVM), …

- Ví dụ như mô hình phân loại rác thải ở mục 1.1.1 (\*) thì đối với sử dụng các loại rác thải đã phân loại để huấn luyện mô hình thì ta cần gán nhãn cho các loại rác ấy để mô hình có thể phân biệt trong quá trình “học”.

* + - 1. **Học không giám sát (Unsupervised machine learning)**

- Cũng tương tự như quá trình học có giám sát thì học không giám sát cũng đạt nhiều thành tựu trong thực tế. Nhưng khác so với học có có giám sát thì ở phương pháp này, ta sẽ sử dụng các tập dữ liệu không được gán nhãn để huấn luyện các thuật toán, giúp các thuật toán này có thể phát hiện ra các mẫu hoặc nhóm dữ liệu ẩn.

- Một số thuật toán được sử dụng trong học không giám sát có thể kể đến như: k-means clustering, neural networks, probabilistic clustering methods, …

* + - 1. **Học bán giám sát (Semi-supervised learning)**

- Đây là phương pháp có sự kết hợp giữa học có giám sát và học không giám sát. Trong quá trình huấn luyện các mô hình của phương pháp này thì nó sẽ sử dụng các tập dữ liệu có gán nhãn để huấn luyện việc phân loại và trích xuất các đặc điểm và tính năng của tập dữ liệu không gán nhãn có kích thước lớn hơn.

<https://en.wikipedia.org/wiki/Machine_learning>

* 1. **Học sâu (Deep learning)**
     1. **Giới thiệu về học sâu**

- Học sâu là một phần trong một nhánh rộng hơn các phương pháp học máy dựa trên mạng thần kinh nhân tạo kết hợp với việc học biểu diễn đặc trưng. Mạng thần kinh nhân tạo này lấy cảm hứng từ cách hoạt động của não con người trong thực tế, cho phép học sâu có thể học hỏi và xử lý thông tin từ một lượng lớn dữ liệu.

Các công nghệ deep learning đã được ứng dụng trong nhiều dự án thực tế mang lại nhiều lợi ích to lớn như hệ thống nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên, nhận dạng giọng nói, khuyến nghị, …

* + 1. **Một số đặc điểm của học sâu**

- Có khả năng tự động học: Các mô hình học sâu có khả năng tự động thay đổi các trọng số để cải thiện khả năng học tập và hiệu suất của mô hình khi tra liên tục cung cấp thêm dữ liệu trong quá trình huấn luyện.

- Tương thích với nhiều dữ liệu phi cấu trúc: Các mô hình học máy truyền thống bị hạn chế bởi khả năng học và phân tích đối với nhiều loại dữ liệu phi cấu trúc. Nhưng học sâu lại có thể tự học biểu diễn nhiều loại dữ liệu phi cấu trúc đó như hình ảnh, văn bản, giọng nói, … để tiến hành quá trình đào tạo rồi từ đó phân tích và giải quyết nhiều bài toán khác nhau.

- Nhiều lớp neural trong các mô hình học sâu: Trong các mô hình học sâu thường chứa nhiều lớp mạng neural xếp chồng lên nhau cho phép mô hình học sâu có thể học được nhiều đặc trưng và các mối quan hệ phức tạp trong dữ liệu.

<https://en.wikipedia.org/wiki/Deep_learning>

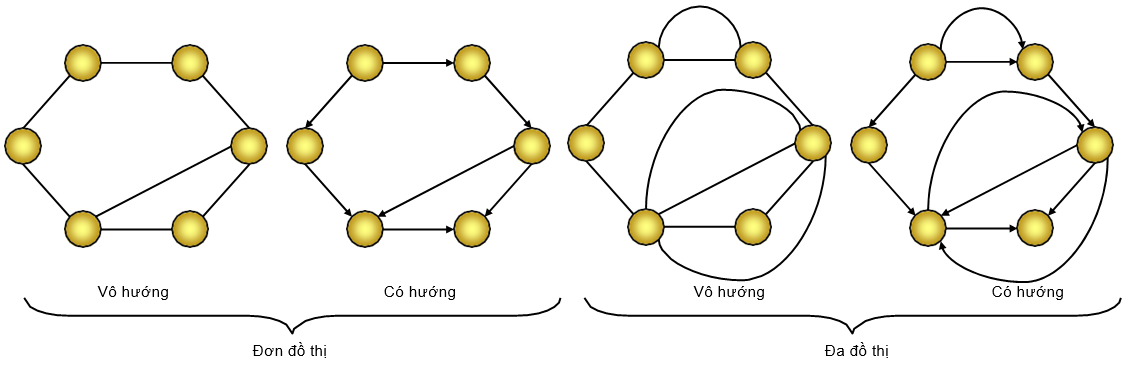
**LÝ THUYẾT CƠ BẢN LIÊN QUAN ĐẾN ĐỒ THỊ**

2. 1. **Lý thuyết đồ thị cơ bản**
      1. Giới thiệu

Đồ thị G = (V, E) là một cấu trúc rời rạc, trong đó gồm một tập các đối tượng được gọi là các đỉnh V (hoặc nút) nối với nhau bởi các cạnh E (hoặc cung). Một cạnh nối một cặp nút u và v, đồng thời chỉ ra rằng có một mối quan hệ 2 nút.

Đồ thị được gọi là có hướng hay vô hướng tùy thuộc vào các cạnh trong đồ thị có hướng hay không. Trong đồ thị nếu như giữa đỉnh bất kỳ có nhiều hơn một cạnh nối hai đỉnh đó thì được gọi là đa đồ thị, ngược lại thì gọi là đơn đồ thị.

Ngoài các khái niệm trên, trong đồ thị còn có một số các khái niệm khác có thể kể đến như: tính kề nhau, bậc, khuyên đường dẫn, …



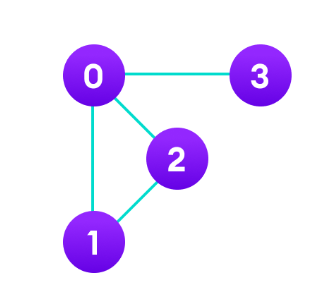
Hình 1. Hình minh họa về các loại đồ thị. Nguồn: viblo.asia

Lý thuyết đồ thị (graph theory) là một nhánh trong lĩnh vực của toán học, nó nghiên cứu về các khái niệm, tính chất và cấu trúc của một đồ thị.

<https://en.wikipedia.org/wiki/Graph_theory>

* + 1. **Các biểu diễn đồ thị**

Có hai cách lưu trữ và biểu diễn đồ thị phổ biến:



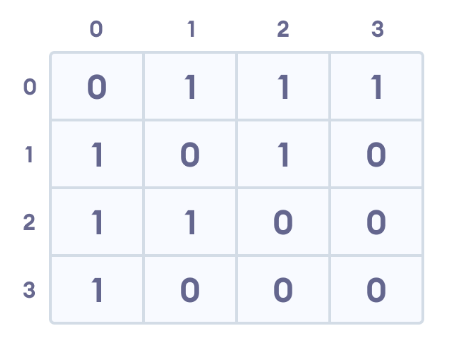
Hình. Hình ảnh minh họa một đồ thị và các cạnh nối giữa các nút. Nguồn: sinhnx.dev

* + - 1. **Ma trận kề**

- Ma trận kề làm một ma trận NxN, trong đó mỗi hàng và cột tương ứng với một đỉnh của đồ thị và N thể hiện cho số đỉnh trong đồ thị. Nếu trong đồ thị có một cạnh nối giữa đỉnh i và j thì giá trị a[i][j] trong ma trận là 1, ngược lại thì giá trị a[i][j] là 0. Trong trường hợp đồ thị vô hướng, ta cũng cần thêm cả giá trị a[j][i] và a[i][j] trong ma trận đều là 1 nếu như có cạnh nối từ đỉnh i đến j trong đồ thị.

- Phương pháp này sẽ dễ dàng thực hiện các phép toán đồng nhất trên ma trận. Nhưng đối với các đồ thị lớn thì phương pháp này sẽ dễ gây tốn kém không gian lưu trữ.

- Dựa trên hình mình họa thì ta có ma trận kề như sau:



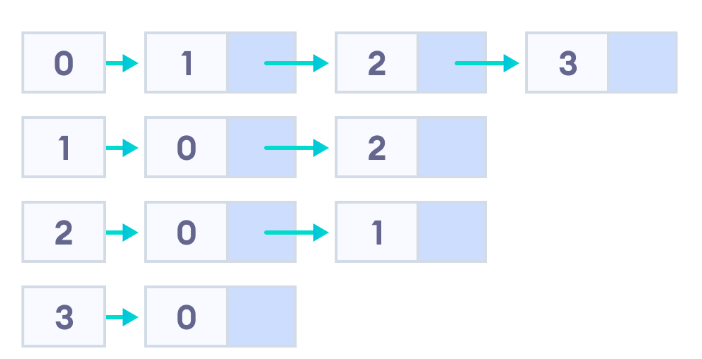
Hình. Hình ảnh minh họa ma trận kề của đồ thị. Nguồn: sinhnx.dev

* + - 1. **Danh sách kề**

- Danh sách kề sẽ biểu thị một biểu đồ dưới dạng một mảng các danh sách được liên kết. Với mỗi đỉnh, ta có một danh sách lưu trữ tất cả các đỉnh kề với nó.

- Phương pháp này sẽ tối ưu được không gian lưu trữ đối với các đồ thị lớn và dễ dàng thực hiện các thao tác như tìm kiếm đỉnh kề, kiểm tra tính liên thông, …. Nhưng trong đồ thị vô hướng, cấu trúc này có thể gây trùng lặp.

- Dựa trên hình mình họa thì ta có danh sách kề như sau:



Hình. Hình ảnh minh họa danh sách kề của đồ thị. Nguồn: sinhnx.dev

<https://en.wikipedia.org/wiki/Graph_theory>

* 1. **Mạng Neural (Neural Network) cơ bản**

Neural network (hay còn gọi là mạng nơ-ron) là mô hình toán học hay mô hình tính toán phức tạp được xây dựng để xử lý thông tin. Mạng nơ-ron được xây dựng dựa trên cấu trúc và cách hoạt động của các tế bào thần kinh trong não con người. Tương tự như bộ não con người, các nút trong mạng nơ-ron cũng liên kết với nhau tạo thành một mạng lưới các nút.

<https://en.wikipedia.org/wiki/Neural_network_(machine_learning)>

<https://bizflycloud.vn/tin-tuc/neural-network-la-gi-20220512174329187.htm>

Mỗi một mạng nơ-ron gồm có ba thành phần chính:

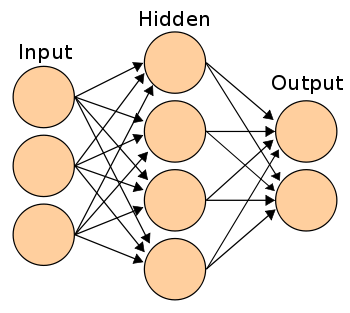
* Tầng vào (input layer): Tầng này gồm các nút nhận dữ liệu đầu vào cho mô hình.
* Tầng ẩn (hidden layer): Là tầng nằm giữa của mô hình, thể hiện quá trình suy luận logic của mô hình. Tùy vào từng trường hợp cụ thể mà tầng này có thể có nhiều lớp neural.
* Tầng ra (output layer): Tầng này gồm các nút xuất ra kết quả của mô hình sau khi nhận kết quả từ tầng ẩn của mô hình.

Các hàm kích hoạt có thể áp dụng trong mô hình có thể là hàm sigmoid, hàm Tanh, hàm Relu, hàm Leaky Relu, …. Tùy theo yêu cầu của người xây dựng mô hình mà có thể áp dụng các hàm kích hoạt khác nhau, nhưng đều áp dụng thuật toán đồng nhất để có thể dễ dàng hoạt động.

Ngoài các hàm kích hoạt thì trong mô hình có các trọng số *w* riêng đối với mỗi liên kết giữa các neural khác nhau và có 1 hệ số bias *b* riêng.

Trong mạng nơ-ron gồm có hai quá trình:

* Quá trình lan truyền xuôi (feedforward): Là quá trình truyền dữ liệu từ tầng, vào qua tầng ẩn, rồi đến tầng ra để trả ra kết quả. Quá trình truyền dữ liệu sẽ diễn ra 2 bước: tính tổng linear và áp dụng activation function trên mỗi nút.
* Quá trình lan truyền ngược (back propagation): Là quá trình dựa trên kết quả của quá trình lan truyền xuôi so với kết quả đầu ra mong muốn để tính hành cập nhật các trọng số liên kết cho mô hình, cải thiện hiệu suất và kết quả của mô hình.



Hình 2. Hình ảnh minh họa về cấu trúc của một Neural Network

1. **TỔNG QUÁT VỀ HỌC BIỂU DIỄN (REPRESENTATION LEARNING)**
   1. **Học biểu diễn**
      1. **Khái niệm**

Học biểu diễn (còn được gọi là học đặc trưng – feature learning) là một quá trình trong học máy, nơi đó phép các thuật toán trích xuất các mẫu có ý nghĩa từ dữ liệu thô để tạo ra các biểu diễn dễ hiểu và xử lý hơn. Nghĩa là quá trình tạo ra các biểu diễn số học cho dữ liệu đầu vào làm sao cho các đặc trưng của dữ liệu được tái hiện lại trong mô hình một cách có hiệu quả nhất.

* + 1. **Ứng dụng của học biểu diễn**

Mục tiêu của học biểu diễn là biến đổi dữ liệu ban đầu từ dạng không có cấu trúc, không có số học (như hình ảnh, …) thành dữ liệu có cấu trúc (như vector, …) để dễ dàng đưa vào các mô hình mà xử lý các bài toán phức tạp.

Vì mục tiêu đó nên học biểu diễn có nhiều ứng dụng trong lĩnh vựa khoa học dữ liệu. Nó thường được sử dụng trong nhận dạng hình ảnh, giọng nói, … thông qua việc tự động học các tính năng quan trọng từ dữ liệu âm thanh, pixel, … thô. Nó cũng được sử dụng trong xử lý ngôn ngữ tự nhiên bằng việc học cách biểu diễn các từ hoặc câu theo cách nắm bắt được ý nghĩa ngữ nghĩa của chúng.

* + 1. **Thách thức của học biểu diễn**

Mặc dù, học biểu diễn mang lại nhiều lợi ích to lớn, nhưng nó cũng gặp nhiều thách thức có thể kể đến như:

- Khó khăn trong việc lựa chọn kiến trúc, mô hình và các tham số cần thiết để có thể biểu diễn dữ liệu tốt nhất, mang lại hiệu quả lớn nhất cho quá trình xử lý sau này. Điều này đòi hỏi kiến thức chuyên môn cao và có sự hiểu biết đối với dữ liệu

- Quá trình này cần đỏi hỏi nhiều tài nguyên và chi phí khi gặp các tập dữ liệu lớn.

<https://saturncloud.io/glossary/representation-learning/>

* 1. **Học biểu diễn đồ thị (Graph representation learning)**
     1. **Khái niệm**

Học biểu diễn đồ thị (hay còn gọi là nhúng đồ thị - Graph Embedding) là một lĩnh vực nghiên cứu rất tích cực và quan trọng trong lĩnh vực học máy và khai phá dữ liệu trong những năm gần đây, mục tiêu của việc học biểu diễn đồ thị là nó tìm cách biểu diễn đồ thị dưới dạng các vector nhưng cấu trúc, quan hệ và các tính năng của đồ thị vẫn được tái hiện một cách có hiệu quả. Nó được sử dụng để phân tích, xử lý và giải quyết các bài toán phức tạp như phân loại đồ thị, dự đoán liên kết giữa các đỉnh, hay tìm kiếm cộng đồng trong đồ thị.

Trong các ứng dụng thực tế, nhiều loại dữ liệu được biểu diễn dưới dạng đồ thị, chẳng hạn như mạng xã hội, mạng ngôn ngữ (liên kết từ), mạng sinh học và các loại dữ liệu đa phương tiện khác.

* + 1. **Một số phương pháp của học biểu diễn đồ thị**

Việc biểu diễn đồ thị có thể có nhiều phương pháp khác nhau, mỗi phương pháp sẽ được ứng dụng tùy vào tường trường hợp. Ở đây, ta có thể kể đến một số phương pháp biểu diễn đồ thị sau:

- Phương pháp dựa trên giảm kích thước (Dimension-reduction-based methods): Các phương pháp nhúng biểu đồ cổ điển nhằm mục đích giảm kích thước của dữ liệu biểu đồ nhiều chiều thành biểu diễn có chiều thấp hơn trong khi vẫn giữ được các thuộc tính mong muốn của dữ liệu gốc.

- Phương pháp dựa trên bước đi ngẫu nhiên (Random-walk-based methods): Random-walk-based methods là các phương pháp dựa trên bước đi ngẫu nhiên trong đồ thị. Cụ thể, cho một đồ thị và một nút khởi đầu, các phương pháp dựa trên bước đi ngẫu nhiên trước tiên chọn một trong các lân cận của nút ngẫu nhiên và sau đó di chuyển đến lân cận này. Thủ tục này được lặp lại để thu được các chuỗi nút. Tính ngẫu nhiên của các bước đi mang lại khả năng khám phá biểu đồ và nắm bắt cả thông tin cấu trúc toàn cầu và cục bộ bằng cách đi qua các đỉnh lân cận. Một số thuật toán của phương pháp này: DeepWalk, Node2vec, …

- Phương pháp dựa trên hệ số ma trận (Matrix-factorization-based methods): là một loại thuật toán lọc cộng tác được sử dụng trong các hệ thống đề xuất. Phương pháp này dựa trên ý tưởng rằng ma trận có thể được phân tách thành hai ma trận nhỏ hơn, mỗi ma trận đại diện cho một yếu tố của dữ liệu, hay nói cách khác là phân tách ma trận tương tác giữa người dùng và mục thành tích của hai ma trận hình chữ nhật có số chiều thấp hơn. Ví dụ, một ma trận đánh giá phim có thể được phân tách thành hai ma trận, một ma trận đại diện cho sở thích của người dùng và một ma trận đại diện cho các đặc điểm của phim. Sau đó, các ma trận này có thể được sử dụng để dự đoán các đánh giá phim mà người dùng chưa xem. Có một số biến thể của phương pháp này: Graph Laplacian eigenmaps, Text-associated DeepWalk (TADW), …

- Phương pháp dựa trên mạng thần kinh (Neural-network-based methods): Các mô hình mạng thần kinh trở nên phổ biến trở lại kể từ năm 2010. Lấy cảm hứng từ sự thành công của RNN và CNN, các nhà nghiên cứu cố gắng khái quát hóa và áp dụng chúng vào biểu đồ. Đây là một loại phương pháp học máy sử dụng các mạng nơ-ron để dự đoán các giá trị trong ma trận, cũng như nó có thể học được các mối quan hệ phức tạp trong dữ liệu. Một số thuật toán sử dụng phương pháp này có thể kể đến như: Mạng tích chập đồ thị (Graph convolutional network), GraphSAGE, …

Chen, F., Wang, Y., Wang, B., & Kuo, C. J. (2020). Graph representation learning: a survey. APSIPA Transactions on Signal and Information Processing, 9(1). [🔗](https://arxiv.org/pdf/1909.00958)

1. **GRAPH NEURAL NETWORKS (GNNs)**
   1. **Giới thiệu về Graph Neural Networks** 
      1. **Lịch sử hình thành của Graph Neural Networks**

Bắt đầu từ những năm 1980, khi các nhà nghiên cứu tìm cách áp dụng mạng nơ-ron cho dữ liệu có cấu trúc đồ thị. Đến những năm 1990, mô hình Recursive Neural Network (RNN) được phát triển để xử lý dữ liệu dạng cây, đặt nền móng cho GNNs sau này.

Thuật ngữ “Graph Neural Networks” được giới thiệu chính thức vào năm 2008 bởi Scarselli và cộng sự. GNNs được phát triển để mô hình hóa các hệ thống động sử dụng dữ liệu đồ thị và nhanh chóng trở thành một lĩnh vực nghiên cứu và ứng dụng quan trọng. Kể từ đó, nhiều kiến trúc GNNs khác nhau đã được đề xuất, triển khai các phương pháp truyền thông điệp khác nhau, bắt đầu bằng các cách tiếp cận đệ quy hoặc tích chập.

Tuy nhiên, phải đến đầu những năm 2010, GNNs mới bắt đầu trở nên phổ biến nhờ sự phát triển của các kiến ​​trúc và thuật toán học tập mới. Tới giai đoạn 2015- 2016, nhờ sự phát triển của Graph Convolutional Networks (GCNs) bởi Thomas Kipf và Max Welling đã mở ra một hướng tiếp cận mới cho việc xử lý dữ liệu trên đồ thị.

Kể từ đó, GNNs tiếp tục phát triển với sự xuất hiện của nhiều biến thể và ứng dụng mới. Các nhà nghiên cứu đang tập trung vào việc tối ưu hóa hiệu suất và áp dụng GNNs vào nhiều lĩnh vực thực tế như hóa học, dự báo tài chính, và y học.

<https://medium.com/@saluem/graph-neural-networks-gnn-93b32567a6d9>

* + 1. **Định nghĩa về Graph Neural Networks**

Mạng nơ-ron đồ thị (Graph Neural Networks – GNNs) là một loại mạng nơ-ron nhân tạo được thiết kế để thực hiện xử lý và suy luận các loại dữ liệu được biểu diễn dưới dạng đồ thị.

<https://en.wikipedia.org/wiki/Graph_neural_network>

* 1. **Các kiến trúc phổ biến của Graph Neural Networks**
     1. **Graph Convolutional Networks (GCNs)**
        1. **Định nghĩa và lý do ra đời**

Trước khi GNNs và các kiến trúc phổ biến của nó ra đời, các kỹ thuật nhúng đã đạt được nhiều thành công, nhưng đôi khi nó hạn chế trong việc phát hiện các mẫu phức tạp hơn trong biểu đồ. Trong khi đó, mô hình học sâu như mạng thần kinh tích chập (CNN) đã chứng minh khả năng mạnh mẽ của mình trong nhiều lĩnh vực. CNN đặc biệt tốt trong việc rút trích đặc trưng từ hình ảnh nhờ tính chất phân cấp của chúng.

Tuy nhiên, việc áp dụng CNN trực tiếp lên biểu đồ gặp nhiều thách thức do tính chất phi Euclide của đồ thị. Gần đây, các nhà nghiên cứu đã phát triển các mô hình mạng nơ-ron tích chập đồ thị (GCN) để giải quyết vấn đề này. GCN tận dụng cấu trúc đồ thị và tổng hợp thông tin từ các vùng lân cận theo kiểu tích chập, giúp mô hình hóa hiệu quả các mối quan hệ phức tạp trong biểu đồ. Nhờ vậy, GCN đã đạt được kết quả ấn tượng trong nhiều ứng dụng thực tế.

* + - 1. **Kiến trúc mô hình**

Cho một đồ thị vô hướng 𝐺 = (𝑉, 𝜉, 𝑋), trong đó 𝑉 là tập hợp các đỉnh, số lượng đỉnh là |𝑉| = 𝑛. 𝜉 là tập hợp các cạnh. 𝑋 = {𝑥1, 𝑥2, ⋯, 𝑥𝑛} là ma trận đặc trưng, và 𝑥𝑖 ∈ 𝑅nxc đại diện cho vecto đặc trưng c chiều của đỉnh thứ i. Mục tiêu của mô hình GCN là sử dụng một phần nhỏ các đỉnh đã được đánh dấu, kết hợp với cấu trúc đồ thị để dự đoán điểm đánh dấu của các đỉnh chưa được đánh dấu còn lại.

Nói một cách đơn giản, mô hình GCN sử dụng thông tin về các đỉnh đã được đánh dấu và cấu trúc của đồ thị để dự đoán điểm đánh dấu của các đỉnh chưa được đánh dấu. Khi có một số đỉnh đã được đánh dấu và một ma trận đặc trưng, mô hình GCN sẽ tính toán các biểu diễn đặc trưng cho tất cả các đỉnh trong đồ thị bằng cách kết hợp thông tin từ các đỉnh lân cận. Sau đó, mô hình sử dụng các biểu diễn đặc trưng này để dự đoán điểm đánh dấu cho các đỉnh chưa được đánh dấu.

Mô hình này sử dụng một công cụ toán học đặc biệt gọi là đa thức Chebyshev để xử lý thông tin trên đồ thị. Trong mỗi lớp của mô hình, nó chỉ sử dụng hai phần đầu của đa thức Chebyshev để xấp xỉ việc tích chập trên đồ thị. Điều này giúp nó xây dựng cách dữ liệu được chuyển đổi qua các lớp trong mạng một cách hiệu quả. Công thức chuyển tiếp của GCN được định nghĩa như sau:

Trong công thức trên:

* : là ma trận kề của đồ thị vô hướng G có thêm các vòng lặp tự động. Nó được tính bằng cách cộng ma trận kề ban đầu A với ma trận đơn vị I, nhân với hệ số N.
* : là ma trận đường chéo bậc tương ứng với ma trận kề . Nó thể hiện tổng số liên kết của mỗi đỉnh. Giá trị của phần tử là tổng của các phần tử trong hàng thứ i của ma trận A.
* H(l) đề cập đến biểu diễn ẩn trong tầng thứ l. Ban đầu, H(0) được khởi tạo bằng ma trận đặc trưng ban đầu X.
* W(l) là ma trận trọng số có thể điều chỉnh cho tầng thứ i.
* = (.) là hàm kích hoạt được áp dụng cho kết quả tính toán trong dấu ngoặc đơn.

Thông thường một mô hình GCN thường có 2 tầng và sử dụng một bộ phân loại softmax trên các đặc trưng đầu ra. Một công thức để tính xác suất cho bài toán phân loại phổ biến sử dụng GCN với ma trận kề A và ma trận trọng số W như sau:

Trong đó:

* : là ma trận kề mở rộng bằng cách chuẩn hóa ma trận kề ban đầu A. Điều này được thực hiện bằng cách chia tổng các hàng của ma trận kề cho căn bậc hai của tổng số liên kết của mỗi đỉnh.
* và : là ma trận trọng số từ lớp input đến lớp ẩn thứ nhất và từ lớp ẩn này đến lớp output. Lớp ẩn có H tính năng. C là số chiều đặc trưng của lớp input và F là số lượng đặc trưng ứng với lớp output.
* F là số lượng đặc trưng của lớp đầu ra.

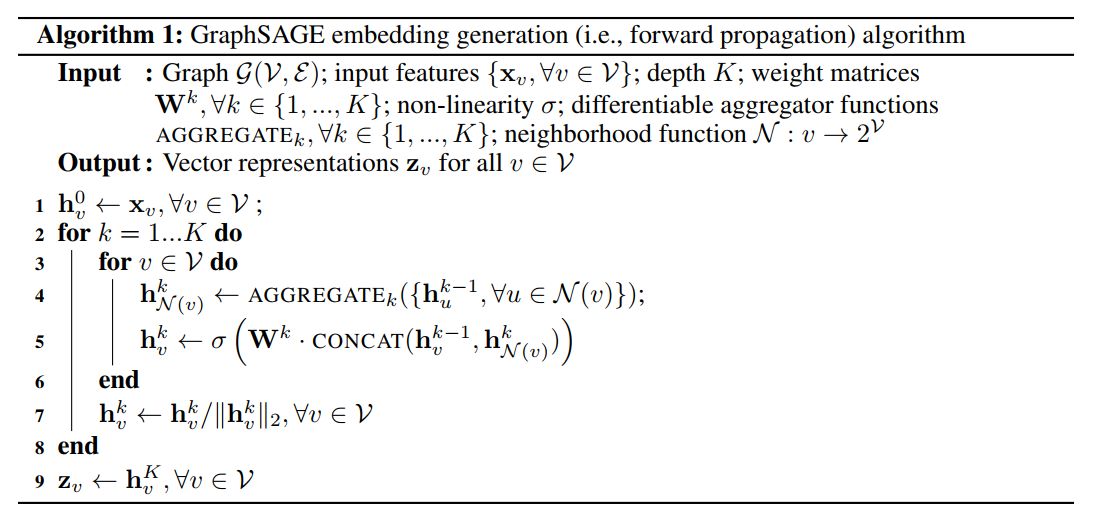
Nhìn vào công thức trên ta thấy lớp đầu tiên là hàm kích hoạt chuẩn hóa và lớp thứ hai là hàm softmax. Hàm softmax có công thức và đây là một hàm đặc biệt được sử dụng để dự đoán xác suất trong các mô hình phân loại ở lớp đầu ra. Hàm này được dùng để chuyển đổi đầu ra của mạng neural thành một vector xác suất.

* + 1. **Graph Sample and Aggregation (GraphSAGE)**
       1. **Định nghĩa**

GraphSEA là một phương pháp quan trọng trong việc nhúng nút trong các biểu đồ lớn. Nó cho phép tạo ra các nhúng nút hiệu quả cho dữ liệu mới mà không cần đào tạo lại từ đầu. Thay vì tạo nhúng riêng lẻ cho từng nút, GraphSAGE sử dụng thông tin tính năng của các nút láng giềng để tổng hợp nhúng cho một nút. Điều này cho phép tạo nhúng cho các nút chưa từng được thấy trước đó. Điểm mạnh của GraphSAGE là khả năng quy nạp, rất phù hợp cho các hệ máy học với thông lượng lớn, hoặc những hệ thống đang xử lý các biểu đồ liên tục phát triển và gặp gỡ các nút mới.

* + - 1. **Thuật toán tạo nhúng**

Thuật toán tạo nhúng hoặc thuật toán lan truyền chuyển tiếp, giả định rằng mô hình đã được đào tạo và các tham số đã được cố định.



**Hình 1. Hình ảnh minh họa cách hoạt động của thuật toán tạo nhúng [14]**

Thuật toán này dựa trên ý tưởng rằng trong mỗi lần lặp hoặc độ sâu tìm kiếm, các nút sẽ tổng hợp thông tin từ các nút lân cận cục bộ của chúng. Quá trình này được lặp lại nhiều lần, và qua thời gian, các nút sẽ thu thập được ngày càng nhiều thông tin từ các phạm vi xa hơn trong biểu đồ.

Ban đầu, mỗi nút chỉ chứa một lượng thông tin hạn chế từ các nút lân cận trực tiếp. Nhưng qua mỗi bước lặp, các nút sẽ trích xuất và tổng hợp thông tin từ các nút lân cận, tạo ra một cái nhìn rõ ràng hơn về cấu trúc và đặc điểm của biểu đồ.

Khi quá trình lặp lại diễn ra, việc tổng hợp thông tin từ các nút lân cận cục bộ sẽ lan rộng và lan truyền thông tin từ các nút xa hơn. Điều này đồng nghĩa với việc các nút sẽ ngày càng nhận được nhiều thông tin từ các phạm vi xa hơn trong biểu đồ. Quá trình này có thể tiếp tục cho đến khi mọi nút trong biểu đồ đã thu thập được thông tin từ toàn bộ mạng lưới.

Bằng cách sử dụng ý tưởng này, thuật toán có khả năng tìm kiếm và lọc thông tin từ phạm vi rộng hơn trên biểu đồ, mang lại cái nhìn toàn diện hơn về cấu trúc và thuộc tính của dữ liệu.

* + - 1. **Các hàm tổng hợp trong GraphSAGE**

Có rất nhiều hàm tổng hợp khác nhau được sử dụng trong graphSAGE nhưng trong số đó có một vài hàm mà ta thường hay gặp và sử dụng đó là: Mean, LSTM và hàm Pooling.

Hàm Mean hoạt động bằng cách lấy giá trị trung bình của các vectơ đặc trưng từ các nút lân cận. Bằng cách sử dụng giá trị trung bình, hàm Mean có thể tái hiện lại trung bình của các đặc điểm hoặc thuộc tính của các nút lân cận, cho ta cái nhìn tổng quan về môi trường xung quanh mỗi nút. Điều này giúp nó thể hiện mối quan hệ giữa các nút lân cận một cách rất trực quan.

Hàm tổng hợp LSTM (Long Short-Term Memory) trong GraphSAGE là một công cụ mạnh mẽ được sử dụng để xử lý thông tin từ các nút lân cận theo một cách có cấu trúc và có thứ tự. LSTM là một loại kiến trúc mạng nơ-ron hồi quy, được biết đến với khả năng xử lý dữ liệu dạng chuỗi và giữ lại thông tin quan trọng trong thời gian dài. Các biểu diễn của hàng xóm được truyền qua mạng LSTM theo thứ tự cụ thể. Mỗi hàng xóm được xem như một thời điểm trong chuỗi đầu vào của LSTM. Quá trình này cho phép LSTM nắm bắt thông tin từ các hàng xóm lân cận theo một thứ tự cụ thể, giúp nắm bắt các mẫu phức tạp và mối quan hệ chuỗi giữa các nút.

Pooling trong GraphSAGE là một phương pháp tổng hợp thông tin từ tập con các nút lân cận. Hàm Pooling áp dụng một phép tổng hợp (như max pooling, mean pooling, min pooling) lên các đặc trưng của các nút hàng xóm. Quá trình này giúp tạo ra một đại diện không gian nhỏ hơn nhưng vẫn giữ được thông tin quan trọng từ tập hợp các nút lân cận.

* + 1. **Graph Attention Networks (GATs)**
       1. **Định nghĩa và lý do ra đời**

- Graph Attention Network (GAT) là một kiến trúc mạng nơ-ron mới hoạt động dựa trên dữ liệu có cấu trúc đồ thị, được giới thiệu bởi Petar Veličković và các cộng sự vào năm 2018. GAT là sự kết hợp giữa mạng thần kinh đồ thị (GNNs) và lớp chú ý (attention layer), sử dụng các lớp tự chú ý được che giấu để giải quyết những hạn chế của các phương pháp trước đó dựa trên tích chập đồ thị hoặc các xấp xỉ của chúng.

Với các mô hình truyền thống hoặc cụ thể hơn là mô hình GNNs thì các mô hình đó thường giả định rằng tất cả các nút trong một đồ thị đều có quan hệ như nhau. Tuy nhiên, điều này thường không đúng với thực tế. Trong một số trường hợp, một số nút trong một đồ thị có liên quan chặt chẽ hơn với nhau hơn những nút khác. Mà khi ta sử dụng các mô hình đó có thể làm tổn hại đến tính tổng quát của dữ liệu có cấu trúc đồ thị do sự tổng hợp thông tin của cấu trúc đồ thị, cụ thể ở đây mà ta có thể kể đến như GCN. Chính vì điều đó, họ đã tận dụng được khả năng xử lý dữ liệu có cấu trúc đồ thị của GNNs và cơ chế trích xuất thông tin có hữu ích từ dữ liệu lớn của Attention Layer lại với nhau để tạo ra GAT.

Veličković, P. (2017, October 30). Graph Attention Networks. arXiv.org. <https://arxiv.org/abs/1710.10903>

* + - 1. **Kiến trúc của mô hình**

GAT khác với các mô hình GNNs khác ở chỗ là nó sự kết hợp với graph attentional layer (GAT layer). Mục tiêu của GAT layer là tạo ra một tập hợp các vectơ đặc trưng mới của các đỉnh trong đồ thị. Sau đó, ta có thể áp dụng các bước tiếp theo của GNNs như bình thường để thực hiện các bài toán khác nhau. GAT layer hoạt động như sau:

- Tính trọng số chú ý (Attention Weights):

+ Đầu tiên ta có đầu vào là tập hợp các vectơ đặc trưng của các đỉnh trong đồ thị, được biểu diễn bằng .Trong đó: N là số nút, F là số lượng các đặc trưng của mỗi nút.

+ Sau đó, ta sử dụng một biến đổi tuyến tính để chuyển đổi đặc trưng của mỗi đỉnh thành một không gian đặc trưng mới: , trong đó là vectơ đặc trưng của đỉnh *i*.

+ Sau đó, chúng ta tính toán trọng số chú ý (attention weights) cho mỗi cặp đỉnh bằng cách sử dụng một cơ chế chú ý độc lập, trong đó mỗi cặp đỉnh được biểu diễn bởi hai vectơ đặc trưng tương ứng:

Trong đó, là vectơ trọng số của cơ chế chú ý với a: , và LeakyReLU là hàm kích hoạt phi tuyến.

- Chuẩn hóa trọng số chú ý:

+ Chuẩn hóa trọng số chú ý bằng cách sử dụng hàm softmax, biến chúng thành các hệ số chuẩn hóa tổng hợp thành 1:

Trong đó, *Nj* là tập hợp các đỉnh lân cận của đỉnh *i* trong đồ thị.

- Tính tổ hợp tuyến tính của đặc trưng:

+ Sử dụng các trọng số chú ý chuẩn hóa để tính tổ hợp tuyến tính của các đặc trưng của các đỉnh lân cận: . Với là hàm kích hoạt phi tuyến tùy chọn.

- Mở rộng sang chú ý đa phần (Multi-Head Attention):

+ Để cải thiện ổn định và đa dạng của quá trình học, chúng ta có thể sử dụng nhiều cơ chế chú ý độc lập (multi-head attention) trong đó mỗi cơ chế chú ý đại diện cho một "đầu".

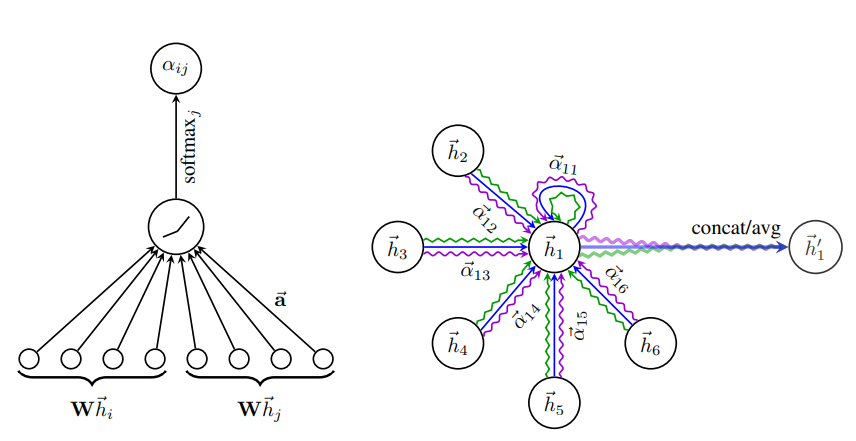
+ Quá trình này áp dụng cho toàn bộ hoặc một phần của đồ thị. Thông thường, trong quá trình này, chúng ta sẽ có nhiều cơ chế chú ý (heads) độc lập, và kết quả từ mỗi head sẽ được kết hợp hoặc tổng hợp để tạo ra đặc trưng cuối cùng cho mỗi nút.

Trong đó, *K* là số lượng cơ chế chú ý đa phần, *Wk* là ma trận trọng số tương ứng với cơ chế chú ý thứ *k* và là các hệ số chú ý chuẩn hóa được tính bằng cách sử dụng cơ chế chú ý thứ *k.*

- Xử lý lớp cuối cùng (Final Layer Processing):

+ Nếu như ta thực hiện multi-head attention trên layer cuối của mạng, việc sử dụng concat sẽ không còn hợp lý. Thay vào đó, chúng ta tính trung bình cộng của chúng và sau đó áp dụng một hàm kích hoạt cuối cùng để tạo ra kết quả cuối cùng của Attention Layer.

+ Hàm kích hoạt này thích hợp cho nhiệm vụ cụ thể như phân loại hoặc dự đoán, thường là softmax cho phân loại nhiều lớp hoặc sigmoid cho phân loại nhị phân.



Veličković, P. (2017, October 30). Graph Attention Networks. arXiv.org. <https://arxiv.org/abs/1710.10903>

* 1. **Ứng dụng của Graph Neural Networks**

GNNs được ứng dụng ở nhiều lĩnh vực trong thực tế. Một số ứng dụng có thể kể đến như:

- Phân tích mạng xã hội: Dùng GNNs để tìm hiểu và phân tích cấu trúc của một mạng xã hội. Từ đó có thể đưa ra những gợi ý, đề xuất kết bạn hoặc theo dõi cho người dùng, …

- Khám phá các loại thuốc: Dùng GNNs để tìm hiểu và phân tích sự tương tác giữa các loại nguyên tử và phân tử có trong các loại thuốc. Rồi từ đó có thể dự đoán được tiềm năng và công dụng của các loại thuốc trong từng trường hợp cụ thể.

- Xử lý ngôn ngữ tự nhiên: Có thể phân tích một đoạn văn bản bằng các sử dụng GNNs khi biểu diễn mối quan hệ giữa các từ có trong đoạn văn thành một sơ đồ.

- Phát hiện gian lận: Có thể phát hiện việc gian lận trong kinh doanh bằng việc mô hình hóa các giao dịch tài chính dưới dạng sơ đồ, rồi sau đó dùng GNNs để tiến hành phân tích và xử lý.

<https://www.xenonstack.com/blog/graph-neural-network-applications>

* 1. **Hạn chế của Graph Neural Networks**

Mặc dù hiện nay, GNNs được ứng dụng trong nhiều vấn đề của thực tế, nhưng GNNs vẫn gặp nhiều hạn chế mà các nhà khoa học nghiên cứu đang cố gắng khắc phục. Một số hạn chế có thể kể đến như:

* Hầu hết các mạng thần kinh có thể đi sâu để có được hiệu suất tốt hơn, trong khi GNNs lại bị hạn chế về hiệu suất khi gặp các tập dữ liệu lớn.
* GNNs đối mặt nhiều vấn đề về mở rộng hoặc áp dụng vào trong sản xuất do mô hình này rất tốn kém về mặt tính toán và lưu trữ nếu gặp cấu trúc đồ thị lớn và phức tạp.
* Việc đào tạo mô hình GNNs trở nên khó khăn hơn khi cấu trúc đồ thị liên tục thay đổi.

<https://www.xenonstack.com/blog/graph-neural-network-applications>

* 1. **Tương lai của Graph Neural Networks**

Trong tương lai, GNNs không chỉ phát triển trong nhiều lĩnh vực và ứng dụng cụ thể, mà GNNs sẽ còn tiếp tục phát triển và cải thiện thêm nhiều mô hình kỹ thuật của GNNs. Một số cải tiến trong tương lai có thể kể đến như:

* Có thể kết hợp thêm nhiều cơ chế và kỹ thuật khác để cải thiện hiệu suất mô hình.
* GNNs có thể cải thiện khả năng mở rộng và tốc độ tính toán đối với các đồ thị lớn và phức tạp hơn.
* Hiện nay trong lĩnh vực học có giám sát thì GNNs được nghiên cứu khá nhiều, nếu trong tương lai có khả năng thì GNNs có thể phát triển sang lĩnh vực học không giám sát.

<https://www.xenonstack.com/blog/graph-neural-network-applications>

**CHƯƠNG 3: ỨNG DỤNG**

1. **GIỚI THIỆU VỀ BÀI TOÁN LINK PREDICTION**
   1. **Định nghĩa bài toán Link Prediction trong mạng đồ thị**

Dự đoán liên kết (Link Prediction) trong mạng đồ thị là việc dự đoán sự tồn tại của một liên kết trong tương lai giữa hai nút, biết rằng không có mối liên hệ nào giữa các nút đó ở trạng thái hiện tại của đồ thị. Nó là một ứng dụng quan trọng của GNNs.

Ví dụ về dự đoán liên kết có thể là dự đoán sự kết bạn giữa hai người dùng trong mạng xã hội, dự đoán sự tương tác giữa gen và protein trong mạng sinh học, …

<https://en.wikipedia.org/wiki/Link_prediction>

* 1. **Cách tiếp cận và phương pháp**
     1. **Các phương pháp truyền thống**

Có 3 lớp phương pháp link prediction truyền thống: lớp phương pháp heuristic (heiristic methods), lớp phương pháp dựa trên đặc trưng ẩn (tatent- feature methods), lớp phương pháp dự trên nội dung (content- base methods).

* + - 1. **Phương pháp heuristic (heiristic methods)**

Với x và y lần lượt là nút nguồn và nút đích trong việc dự đoán liên kết. Khí đó, ta có F(x) là tập hợp các nút hàng xóm của x, cũng như F(y) là tập hợp các nút hàng xóm của y.

**1.2.1.1.1. Local Heuristic**

- Một phương pháp heuristic đơn giản nhất trong phần này là common neighbors (CNs), trong đó thì ta sẽ đếm xem giữa các hàng xóm của hai nút cần dự đoán thì có bao nhiêu nút chung, nếu càng nhiều nút chung thì khả năng liên kết của 2 nút trong tương lai là cao.

- Ngoài phương pháp common neighbors, thì còn có Jaccard Score (dựa vào việc tính tỷ lệ hàng xóm chung so với tổng số hàng xóm của cả 2 nút cần dự đoán), Preferential Attachment (PA - dựa trên nguyên tắc là đỉnh có nhiều liên kết hiện tại sẽ có xu hướng tạo ra nhiều liên kết mới hơn trong tương lai), Adamic-Adar (được tính dựa trên độ nút của các nút có liên kết kề chung với 2 nút cần dự đoán), …

**1.2.1.1.2. Global Heuristic**

- Katz Index là phương pháp dựa trên đường đi ngắn nối liền 2 đỉnh cần dự đoán với nhau, hay nói cách khác thì nếu hai đỉnh gần nhau trong mạng, có nhiều đường đi ngắn kết nối chúng, thì khả năng có liên kết giữa chúng là cao:

Trong đó: b là hệ số giảm dần nằm trong khoảng 0 đến 1, đếm số lượng đường đi có độ dài l bước giữa x và y.

- Ngoài phương pháp Katz Index, thì trong phần này còn một số phương pháp khác có thể kể đến như Rooted PageRank (một hướng mở rộng của PageRank), SimRank (một phương pháp dựa trên giả thiết rằng hai nút sẽ càng tương tự nhau nếu các nút láng giềng của chúng cũng tương tự nhau)

* + - 1. **Phương pháp dựa trên đặc trưng ẩn (Tatent- feature methods)**

Là lớp phương pháp thứ 2 trong phương pháp link prediction truyền thống. Lớp này sẽ tính toán các đặc trưng ẩn hay biểu diễn ẩn của các nút, thường thu thấp bằng cách phân tích ma trận xấp xỉ từ mạng như ma trận kề hoặc ma trận xác suất chuyển tiếp.

* + - * 1. **Phương pháp phân tích ma trận (Matrix Factorization)**

Phương pháp phân tích ma trận (Matrix Factorization) là một trong những phương pháp Latent-Feature Methods phổ biến nhất trong dự đoán liên kết. Matrix Factorization là một phương pháp trong đó ma trận kề của mạng được phân tách thành hai ma trận nhỏ hơn có kích thước thấp hơn.

Trong đó:

- Ma trận kề A mô tả mối liên hệ hiện có trong mạng.

- Ma trận biểu diễn ẩn Z có kích thước N x k, với k << N. Mỗi hàng của Z cho biểu diễn vector k-chiều của một nút.

- Phương pháp này tối ưu hóa sai số của A so với tích ZZT nhằm học được ma trận biểu diễn ẩn.

Sau khi được học, biểu diễn vector của hai nút được sử dụng để dự đoán xác suất kết nối của chúng:

* + - * 1. **Phương pháp Network Embedding**

Network embedding là một phương pháp Latent-Feature Methods phổ biến để học biểu diễn vector cho các nút trong mạng. Là một phương pháp trong đó mỗi đỉnh trong mạng được ánh xạ vào một không gian vectơ có số chiều thấp hơn. Thuật toán bao gồm các bước tuần hoàn trên mạng, mỗi bước trích xuất một cặp nút gần nhau rồi cập nhật vector đại diện cho chúng. Phương pháp này thường sử dụng các mô hình học sâu như Skip-gram, DeepWalk, Node2vec, hoặc các biến thể của các mô hình này để học các vectơ nhúng cho các đỉnh. Đầu ra là các vector đặc trưng kích thước thấp hơn cho mỗi nút. Vector này được sử dụng thay cho nút trong các bài toán phân loại, dự đoán.

* + - 1. **Phương pháp dựa trên nội dung (Content- base methods)**

Content-based methods sử dụng các đặc trưng riêng của đỉnh (explicit content features) liên kết với các đỉnh để dự đoán liên kết.

Content-based methods thường có hiệu suất kém hơn các phương pháp Heuristic và Latent-feature methods do không khai thác cấu trúc đồ thị. Do đó, chúng thường được kết hợp với hai loại phương pháp kia nhằm tăng cường hiệu suất dự đoán liên kết.

* + 1. **Áp dụng Graph Neural Networks cho bài toán Link prediction**
       1. **Phương pháp dựa trên nút (Node-Based Methods)**

Các phương pháp dựa trên nút đầu tiên học cách biểu diễn một nút thông qua một GNNs, sau đó tổng hợp các biểu diễn của các cặp nút để tạo ra biểu diễn cho liên kết (link representation) và dùng để dự đoán liên kết.

Graph AutoEncoder (GAE) là một trong những mô hình phổ biến của phương pháp này. Phương pháp này tập trung vào việc học biểu diễn nút và sử dụng thông tin về cặp nút để dự đoán liên kết. Graph AutoEncoder (GAE) gồm có hai bộ chính là:

- Bộ mã hóa (encoder): Bộ này sẽ áp dụng GNNs để biến đổi biểu đồ đầu vào thành một biểu diễn tiềm ẩn (latent representation) có chiều thấp hơn.

- Bộ giải mã (decoder) nhằm tái tạo lại biểu đồ từ biểu diễn tiềm ẩn đã học trước đó.

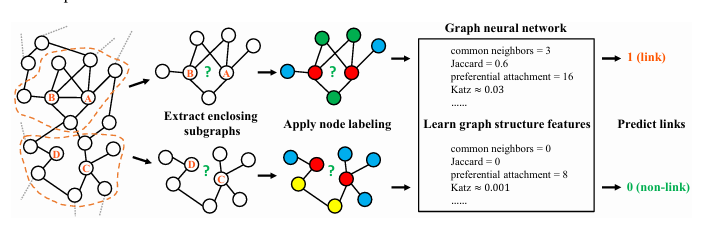
Có nhiều biến thể của GAE. Mỗi một biến thể lại có một đặc trưng riêng nhằm tối ưu mô hình, có thể kể đến như: Variational Graph AutoEncoder (VGAE), MGAE (Marginalized Graph AutoEncoder), S-VAE (Spherical Variational AutoEncoder), …

* + - 1. **Phương pháp dựa trên tiểu đồ thị (Subgraph-Based Methods)**

Phương pháp subgraph-based sẽ chia đồ thị lớn thành các tiểu đồ thị nhỏ xung quanh mỗi liên kết mục tiêu. Sau đó, mỗi tiểu đồ thị này sẽ được đưa vào một mô hình để học biểu diễn của nó. Điều này giúp mô hình học được các đặc trưng cục bộ và cấu trúc của đồ thị xung quanh mỗi liên kết mục tiêu, từ đó dự đoán khả năng tồn tại của liên kết.

* + - * 1. **SEAL Framework (Subgraph-based Embedding and Aggregation Learning Framework)**

Công trình tiên phong của các phương pháp dựa trên đồ thị con là SEAL. Khái quát ý tưởng: Đầu tiên SEAL sẽ tách đồ thị ban đầu thành các đồ thị con cục bộ xung quanh mục tiêu cần dự đoán. Rồi tiến hành gán nhãn cho từng nút để phân biệt vai trò của từng nút trong đồ thị. Sau đó, áp dụng GNNs cho tìm hiểu cấu trúc đồ thị và dự đoán liên kết (Hình minh họa)



Các bước của SEAL được tiến hành cụ thể như sau:

- Trích xuất và gán nhãn đỉnh của đồ thị con:

+ SEAL bắt đầu bằng cách trích xuất đồ thị con *Gh* chứa các đỉnh *x* và *y*, đó là các đỉnh mục tiêu nằm giữa các liên kết mục tiêu. Ngoài các đỉnh mục tiêu thì đồ thị con này còn có các đỉnh và liên kết liên quan trực tiếp đến liên kết mục tiêu. Quá trình trích xuất đồ thị con giúp tập trung vào những thông tin quan trọng nhất cho dự đoán liên kết.

+ Tiếp theo, nó áp dụng một kỹ thuật Double Radius Node Label (DRNL) để gán nhãn số nguyên cho mỗi đỉnh trong đồ thị con. Mục đích là phân biệt các đỉnh dựa trên vai trò của chúng trong đồ thị con. Ví dụ, các đỉnh trung tâm *x* và *y* được phân biệt với các đỉnh khác, và các đỉnh cách xa khác nhau từ *x* và *y* có thể có ý nghĩa cấu trúc khác nhau đối với sự tồn tại của liên kết.

- Mã hóa các đặc trưng:

+ Sau khi có các nhãn số nguyên từ kỹ thuật DRNL, SEAL sẽ chuyển đổi chúng thành các vector mã hóa one-hot hoặc đưa chúng vào một lớp embedding để tạo ra các embedding.

+ Các vector đặc trưng mới này được nối với các đặc trưng nội dung đỉnh ban đầu, nếu có, để tạo ra biểu diễn đặc trưng đỉnh được cập nhật.

+ Ngoài ra, SEAL cũng cho phép nối các embedding đỉnh được huấn luyện trước, như các embedding của kỹ thuật node2vec, với các đặc trưng đỉnh. Tuy nhiên, dựa trên các thí nghiệm cho thấy việc thêm này không cải thiện đáng kể hiệu suất và có thể làm mất khả năng học có quy định của SEAL.

- Xử lý mạng nơ-ron đồ thị (GNNs):

+ Các đồ thị con đã được xử lý cùng với các biểu diễn đặc trưng đỉnh đã được cập nhật được đưa vào một graph-level GNNs, Dynamic Graph Convolutional Neural Network (DGCNN) để học một hàm phân loại đồ thị.

- Huấn luyện và dự đoán:

+ SEAL sẽ ngẫu nhiên lấy mẫu N liên kết hiện có từ mạng làm các ví dụ huấn luyện tích cực (positive training links) và chọn một số bằng nhau của các liên kết chưa quan sát (có nghĩa là cặp đỉnh ngẫu nhiên chưa có liên kết) làm các ví dụ huấn luyện tiêu cực (negative training links).

+ Sau khi huấn luyện, SEAL sẽ áp dụng GNNs đã được huấn luyện vào các đồ thị con của các cặp đỉnh mới chưa quan sát (cặp đỉnh chưa có liên kết) để dự đoán các liên kết của chúng.

* + - * 1. **Weisfeiler-Lehman Graph Neural Networks**

Trong lý thuyết đồ thị, phương pháp Weisfeiler-Leman dựa trên việc biểu diễn các đồ thị dưới dạng dạng bình thường (normal form) để tạo ra một biểu diễn chuẩn cho các đồ thị. Quá trình này giúp chuẩn hóa cấu trúc của các đồ thị và tạo ra một dạng biểu diễn đồ thị mà có thể so sánh được.

Weisfeiler-Lehman (1-WL) Test là phương pháp phổ biến để xác định cấu trúc đồ thị, tuy nhiên nó có nhược điểm là mất thông tin điều kiện và sự phụ thuộc giữa hai đỉnh mục tiêu, dẫn đến việc phân biệt các liên kết chỉ ở mức độ của các nút.

Chính vì điều đó, k-dimensional WL tests (k-WL) đã ra đời và khắc phục nhược điểm này bằng cách học biểu diễn cho toàn bộ liên kết, cải thiện khả năng phân biệt các liên kết có cấu trúc tương tự nhưng vị trí khác nhau, và cho phép dự đoán tất cả các liên kết trong một bước tính toán GNNs duy nhất, nâng cao hiệu suất.

Trong k-dimensional WL tests (k-WL), với k=2 có nghĩa là việc cập nhật biểu diễn sẽ dựa trên từng cặp nút thay vì từng nút riêng lẻ. Mỗi cặp nút sẽ cập nhật biểu diễn của nó dựa trên các cặp nút lân cận, tức là các cặp nút khác mfa nó có liên kết trực tiếp trong đồ thị. Điều này giúp 2-WL có thể khắc phục được các nhược điểm của 1-WL.

Giả sử chúng ta có một đồ thị đơn giản với 5 nút: A, B, C, D và E. Các cạnh kết nối giữa các nút tạo nên các cặp nút sau: (A, B), (B, C), (C, D), (D, E) và (E, A).

Bắt đầu từ bất kỳ cặp nút nào, ví dụ (A, B), chúng ta sẽ xem xét các cặp nút "lân cận". Trong trường hợp này, các cặp nút lân cận có thể là (B, C) và (E, A) vì chúng có chung một nút với cặp nút (A, B). Chúng ta sẽ cập nhật biểu diễn của cặp nút (A, B) dựa trên thông tin từ các cặp nút lân cận này.

K-WL có rất nhiều biến thể nhưng có 2 biến thể mà thường hay được sử dụng đó là K-WL và Folklore WL (k-FWL)

Cả 2 k-WL và k-FWL đều cập nhật biểu diễn cho các bộ k-tuple của các nút, nơi mà một k-tuple s được định nghĩa bởi với s1,…., sk là các nút nhưng khác nhau ở chỗ cách định nghĩa tập lân cận.

Đối với K-WL tập nút lân cận được định nghĩa như sau:

Các hàng xóm thứ j của bộ k-tuple s được thu được bằng cách thay thế phần tử thứ j của s bằng s’ thuộc [n], trong đó [n] đại diện cho tập hợp các nút trong đồ thị. Và tập hợp toàn bộ lân cận của bộ k-tuple s được định nghĩa là N(s) = N1(s), N2(s), …, Nk(s). Do đó, k-WL có k tập hợp lân cận chi tiết Nj(s); j [k], và mỗi tập hợp lân cận này có n k-tuple.

Đối với K-FWL tập nút lân cận được định nghĩa như sau:

Để xây dựng mỗi vùng lân cận của đỉnh s , thuật toán thực hiện một quy trình lặp. Trong quá trình này, mỗi phần tử trong đỉnh s được thay thế bằng một đỉnh j khác nhau từ 1 đến n. Kết quả cuối cùng là một tập hợp các đỉnh mới tạo ra từ việc thay thế mỗi phần tử trong s bằng một đỉnh j khác nhau. Về cơ bản, k-WL và k-FWL đều xác định nk nhóm hàng xóm, nhưng chúng sắp xếp và nhóm các nhóm này theo cách khác nhau dẫn đến hiệu quả của chúng khác nhau.

* 1. **Ứng dụng của bài toán Link prediction**

Dự đoán liên kết có rất nhiều ứng dụng trong các ứng dụng trong thế giới thực. Dưới đây là một số trường hợp sử dụng quan trọng của dự đoán liên kết:

- Dự đoán khách hàng nào có khả năng mua sản phẩm nào trên các thị trường trực tuyến như Amazon từ những thông tin, sở thích của người dùng. Nó có thể giúp đưa ra khuyến nghị sản phẩm tốt hơn.

- Dự đoán sự kết bạn giữa hai người dùng trên các mạng xã hội như facebook. Từ đó, nó giúp cho nền tảng mạng xã hội có thể gởi các đề suất kết bạn mới cho người dùng.

- Có thể đề xuất sự tương tác hoặc hợp tác giữa các nhân viên trong các dự án hoặc trong các tổ chức.

- Dựa trên những thông tin hiện có, nó có thể trích xuất được những hiểu biết quan trọng từ các mạng lưới khủng bố.

<https://www.analyticsvidhya.com/blog/2020/01/link-prediction-how-to-predict-your-future-connections-on-facebook/>

* 1. **Các thách thức của bài toán Link prediction**

Bài toán dự đoán liên kết gặp nhiều thách thức. Một trong số đó có thể kể đến như:

- Độ chính xác: Việc dự đoán chính xác các liên kết tiềm năng là không dễ dàng, đặc biệt khi mạng xã hội phát triển và thay đổi liên tục.

- Độ phức tạp: Mạng xã hội là động và phức tạp với số lượng lớn các biến tham số, làm tăng độ khó của việc mô hình hóa và dự đoán.

- Sự đa dạng của mối quan hệ: Các mối quan hệ trong mạng xã hội có thể rất đa dạng và phức tạp, từ bạn bè, đồng nghiệp, đến các mối quan hệ gia đình, yêu cầu các phương pháp tiếp cận khác nhau để hiểu và dự đoán.

- Tính động của mạng: Mạng xã hội không ngừng thay đổi với các nút và cạnh mới được thêm vào, điều này đòi hỏi các mô hình dự đoán phải liên tục cập nhật và thích ứng.

- Dữ liệu thiếu hoặc nhiễu: Trong thực tế, dữ liệu thu thập được có thể không đầy đủ hoặc chứa nhiễu, làm giảm khả năng dự đoán chính xác của mô hình.

- Quy mô: Kích thước lớn của mạng xã hội đòi hỏi các thuật toán phải hiệu quả về mặt tính toán để có thể xử lý dữ liệu lớn mà không làm giảm hiệu suất.

- Sự ảnh hưởng của các nút khác: Các nút khác trong mạng có thể ảnh hưởng đến khả năng hình thành liên kết giữa hai nút cụ thể, làm cho việc dự đoán trở nên phức tạp hơn.

<https://lnhutnam.github.io/post/link-prediction-algorithms/>

1. **-THỰC NGHIỆM VÀ ĐÁNH GIÁ**
   1. **Môi trường cần thiết**

Bài toán được thực hiện trên môi trường chính là Visual Studio Code. Visual Studio Code là mã nguồn mở IDE mạnh dễ sử dụng và được phát triển bởi Microsoft dành cho Windows, Linux và macOS. Visual Studio Code hỗ trợ trên nhiều ngôn ngữ lập trình và nhiều mở rộng khác nhau có thể kể đến như Java, JavaScript, TypeScript, Node.js, Python, C, C++, SQL, Git, Docker, ...

Nhóm sử dụng hệ điều hành Windows với phiên bản 10, Visual Studio Code phiên bản 1.85.1, Python 3.7.16 và sử dụng nhiều thư viện khác nhau của python để tiến hành chạy chương trình. Một số thư viện có thể kể đến như:

- Thư viện Streamlit: Streamlit là một framework mã nguồn mở và miễn phí để nhanh chóng xây dựng và chia sẻ các ứng dụng web khoa học dữ liệu và học máy tuyệt đẹp. Đây là một thư viện dựa trên Python được thiết kế dành riêng cho các kỹ sư học máy để có thể hiển thị dữ liệu và thu thập các tham số cần thiết cho mô hình hóa.

- Thư viện Tensorflow: Tensorflow là một thư viện mã nguồn mở được sử dụng cho tính toán số học và xử lý dữ liệu bằng cách sử dụng đồ thị luồng dữ liệu. Nó được phát triển bởi Google và là một trong những thư viện phổ biến nhất cho học máy và deep learning. Nó cung cấp khả năng tích hợp với nhiều thư viện machine learning, hỗ trợ tốt cho việc xây dựng và huấn luyện các mô hình học máy và nó sử dụng cấu trúc đồ thị (graph) để biểu diễn các phép tính và dữ liệu trong mô hình. <https://www.tensorflow.org/?hl=vi>

- Thư viện Scikit-learn (Sklearn) là một công cụ mạnh mẽ dành cho việc thực hiện các tác vụ học máy và mô hình hóa thống kê bằng ngôn ngữ Python. Nó cung cấp các công cụ đơn giản và hiệu quả cho phân tích dự đoán dữ liệu. Scikit-learn được xây dựng trên các thư viện NumPy, SciPy và matplotlib. <https://scikit-learn.org/>

- Thư viện Pandas: Pandas là một thư viện mã nguồn mở, hỗ trợ đắc lực trong thao tác dữ liệu. Đây cũng là bộ công cụ phân tích và xử lý dữ liệu mạnh mẽ của ngôn ngữ lập trình Python. Thư viện này sử dụng một cấu trúc dữ liệu riêng là Dataframe. <https://pandas.pydata.org/>

- Thư viện NumPy: NumPy là một công cụ mạnh mẽ trong Python dành cho tính toán số học và xử lý dữ liệu. Nó được phát triển bởi Travis Oliphant vào năm 2005 và là một trong những thư viện phổ biến nhất cho khoa học máy tính, tính toán số học trong Python và cung cấp một đối tượng mảng đa chiều hiệu suất cao và các công cụ để làm việc với các mảng này. <https://numpy.org/>

* 1. **Mô tả dữ liệu**

Tập dữ liệu “fb-pages-food” là một tập dữ liệu mạng xã hội, cụ thể là một mạng xã hội trên Facebook liên quan đến các trang có tích xanh về thực phẩm. Tập dữ liệu này được thu thập vào tháng 11 năm 2017 và được trình bày dưới dạng đồ thị vô hướng. Tập dữ liệu chứa thông tin về các trang Facebook và các mối quan tâm chung giữa chúng. Các chú thích và tài liệu tham khảo liên quan đến tập dữ liệu cũng được cung cấp.

Thông tin chi tiết về tập dữ liệu:

- Loại dữ liệu: Mạng xã hội

- Lĩnh vực: Thực phẩm

- Nguồn: Network Repository ( <https://networkrepository.com/fb-pages-food.php> )

- Kích thước tập dữ liệu: 620 nút (đại diện cho các trang facebook có liên quan đến thực phẩm) và hơn 2100 liên kết (đại diện cho mối quan hệ giữa các trang, ví dụ: theo dõi, thích, …)

- Một số thông tin thống kê khác: mật độ (density), bậc tối đa (maximum degree), bậc tối thiểu (minimum degree), bậc trung bình (average degree), hệ số đồng loạt (assortativity), số tam giác, hệ số gom cụm trung bình (average clustering coefficient), và nhiều thông tin khác về cấu trúc mạng.

* 1. **Xử lý dữ liệu**
     1. **SEAL Framework**

Quá trình đọc dữ liệu:

- Đầu tiên, nhóm sẽ tiến hành đọc dữ liệu các cạnh có trong đồ thị, sau đó chuyển các dữ liệu đọc được thành mảng NumPy, rồi lưu mảng đó vào biến “*positive*”.

- Tiếp theo, dựa vào các cạnh có trong đồ thị từ bước trước để vẽ đồ thị vô hướng “*G*”, nhóm tiến hành trích xuất ra các cạnh không tồn tại trong đồ thị, rồi lưu danh sách đó vào biến “*negative*”, với số lượng các cạnh có trong danh sách này bằng với số cạnh có trong “*positive*”.

- Tiếp đến, vì để đảm bảo tính ngẫu nhiên và tính đa dạng trong quá trình chia dữ liệu, nhằm phục vụ cho quá trình huấn luyện và kiểm tra trên mô hình nên nhóm đã tiến hành xáo trộn danh sách “*positive*” và “*negative*”.

- Đồng thời, vì để đảm bảo rằng các chỉ số và giá trị của các cạnh có trong “*positive*” và các cạnh có trong “*negative*” phù hợp với chỉ số và giá trị của các nút trong đồ thị nên đã tiến hành giảm đi 1 nếu giá trị nhỏ nhất trong mảng là 1.

Quá trình nhúng dữ liệu:

- Đầu tiên, để thực hiện quá trình embedding, nhóm đã tiến hành chia dữ liệu tập train và tập test với kích thước tập test theo tỷ lệ 0.1.

- Sau đó, vì để tăng khả năng học được cấu trúc, mối quan hệ trong mạng và tăng khả năng dự đoán, khả năng tổng quát hóa trên dữ liệu thì chương trình đã tạo ra một đồ thị “A” có các cạnh là các cạnh trong “*positive*” và “*negative*” nếu ta cho phép các cạnh trong danh sách “*negative*” được tham gia vào quá trình embedding.

- Tiếp đến ta khởi tạo một đồ thị “*G*” vô hướng từ đồ thị “*A*” sử dụng thuật toán Node2Vec, với việc không có sự ưu tiên nào giữa các đỉnh cùng loại (gần nhau) hoặc khác loại (xa nhau) thông qua tham số p = 1 (pronounced “p-factor”) và q = 1 (pronounced “q-factor”).

- Sau đó, dùng hàm “*preprocess\_transition\_probs*” trong node2vec để tiền xử lý các xác suất chuyển đổi nhằm hướng dẫn các random walks (các bước di chuyển ngẫu nhiên trên đồ thị).

- Tiếp đến, sử dụng các chuỗi random walk có được từ phương pháp “*simulate\_walks*” của đồ thị “*G*” trong node2vec (với mỗi chuỗi sẽ có 80 bước di chuyển) để huấn luyện một mô hình Word2Vec, trong đó kích thước của vector embedding là “*dimension*” và thuật toán sử dụng là *Skip-gram* (sg=1).

- Cuối cùng, ta trích xuất vector embedding của mỗi nút từ mô hình Word2Vec đã huấn luyện trước đó. Trong trường hợp một số nút không có vector embedding (do chúng không xuất hiện trong các chuỗi random walk), ta tính toán trung bình của tất cả các vector embedding đã học được từ các nút khác và sử dụng nó cho các nút không có embedding.

Trích xuất tiểu đồ thị

- Đầu tiên, chia tập “*positive*” và “*negative*” thành tập “*train*” và tập “*test*”.

- Sau đó, ta tiến hành tạo một ma trận kề A với kích thước của ma trận là kích thước của đồ thị, trong đó giá trị 1 được gián cho (j, i) nếu có một cạnh từ i đến j.

- Tiếp đến, ta tính toán hệ số tương đồ giữa các nút trong đồ thị dựa trên thông tin cấu trúc để từ đó xác định tham số “*hop*” trong quá trình tạo ra đồ thị con. Trong đó, hai phương pháp được sử dụng để tính toán là Common Neighbors (cn) và Adamic-Adar (aa), nếu phương pháp nào có kết quả AUC lớn hơn thì sẽ sử dụng tham số “*hop*” của phương pháp đó để tối ưu.

- Tiếp theo, ta sử dụng vòng lặp *enumerate* để duyệt qua từng tập dữ liệu (negative và positive). Với mỗi tập dữ liệu, một nhãn (label) tương ứng được gán: 0 cho negative và 1 cho positive. Dùng vòng lặp *for* để duyệt qua mỗi cặp đỉnh (node\_pair) trong tập dữ liệu đó.

- Với mỗi cặp đỉnh được trích xuất ra thì ta sử dụng hàm *extract\_subgraph* để trích xuất thông tin về đồ thị con xung quanh cặp đỉnh đó. Trong hàm *extract\_subgraph*, ta cập nhật danh sách “*sub\_graph\_nodes*” chứa các đỉnh gốc và các đỉnh láng giềng ngẫu nhiên sao cho tổng số lượng đỉnh trong “*sub\_graph\_nodes*” không vượt quá giá trị của tham số “*max\_neighbors*” được truyền vào, rồi tạo ma trận kề “*sub\_graph\_adj*” của đồ thị con từ ma trận kề ban đầu *A*, chỉ chứa các cạnh giữa các đỉnh trong “*sub\_graph\_nodes*” với việc gán giá trị 0 cho các phần tử tương ứng với cạnh giữa các đỉnh gốc. Sau đó, ta sử dụng một thuật toán gọi là “*node labeling*” để gán nhãn cho các đỉnh trong đồ thị con dựa trên ma trận kề của nó và loại đồ thị (vô hướng hoặc có hướng). Cuối cùng, hàm “*extract\_subgraph*” trả về danh sách các đỉnh trong đồ thị con (*sub\_graph\_nodes*), ma trận kề của đồ thị con (*sub\_graph\_adj*) và thông tin nhãn của các đỉnh trong đồ thị con (*vertex\_tag*).

- Tiếp đến, ta tạo một mảng 2D chứa các nhãn của các cạnh trong tập dữ liệu, với nhãn 0 cho các cạnh “*negative*” và nhãn 1 cho các cạnh “*positive*”.

- Sau đó, ta kết hợp tất cả các nhãn của tất cả các đỉnh trong mỗi đồ thị con thành một tập hợp duy nhất, rồi từ đó chuyển nó thành một danh sách (*vertex\_tags\_set*).

- Tiếp đến, ta kiểm tra xem tất cả các phần tử trong “*vertex\_tags\_set*” có tạo thành một chuỗi số nguyên liên tục từ 0 đến “*len(vertex\_tags\_set) – 1*” không. Nếu không thì ta tạo một bản đồ (*vertex\_map*) từ các nhãn trong “*vertex\_tags\_set*” đến các chỉ số từ 0 đến “*len(vertex\_tags\_set) – 1*”. Từ đó, ta duyệt qua mỗi danh sách nhãn của các đỉnh trong “*vertex\_tags*” và ánh xạ các nhãn này sang các chỉ số tương ứng trong “*vertex\_map*”.

- Cuối cùng, ta trả về thông tin về các đồ thị con (*graphs\_adj*), nhãn của các cạnh (*labels*), nhãn của các đỉnh (*vertex\_tags*), kích thước của các đồ thị con (*node\_size\_list*), danh sách các đỉnh trong các đồ thị con (*sub\_graphs\_nodes*) và kích thước của tập hợp các nhãn (*tags\_size*).

Xây dựng đầu vào cho GNN

- Tạo một mảng Y với các phần tử có giá trị 1 tương ứng với các cạnh *positive* và các phần tử có giá trị 0 tương ứng với các cạnh *negative*.

- Thêm ma trận đơn vị vào mỗi ma trận kề của mỗi đồ thị con bằng cách sử dụng np.eye để tạo ma trận đơn vị có kích thước phù hợp và sau đó cộng vào ma trận kề ban đầu.

- Tạo một danh sách *D\_inverse* trong đó mỗi phần tử là ma trận nghịch đảo của đường chéo của ma trận kề tương ứng trong *graphs\_adj*.

- Nếu danh sách nhãn của các đỉnh (*vertex\_tags*) không phải là None, sử dụng mã hóa one-hot cho các nhãn của các đỉnh. Mỗi hàng của ma trận one-hot biểu diễn một nhãn của một đỉnh. Ngược lại, sử dụng độ bậc chuẩn hóa của các đỉnh trong đồ thị con làm đặc trưng và kết hợp các ma trận đặc trưng này để tạo ma trận đặc trưng cuối cùng.

- Cuối cùng, nếu có sẵn các đặc trưng embedding, ta thêm chúng vào ma trận đặc trưng bằng cách kết hợp với các đặc trưng đã có.

Chia dữ liệu

- Vì để đảm bảo tính ngẫu nhiên và đồng nhất trong cả hai tập dữ liệu, nên trước khi chia dữ liệu, dữ liệu sẽ được xáo trộn.

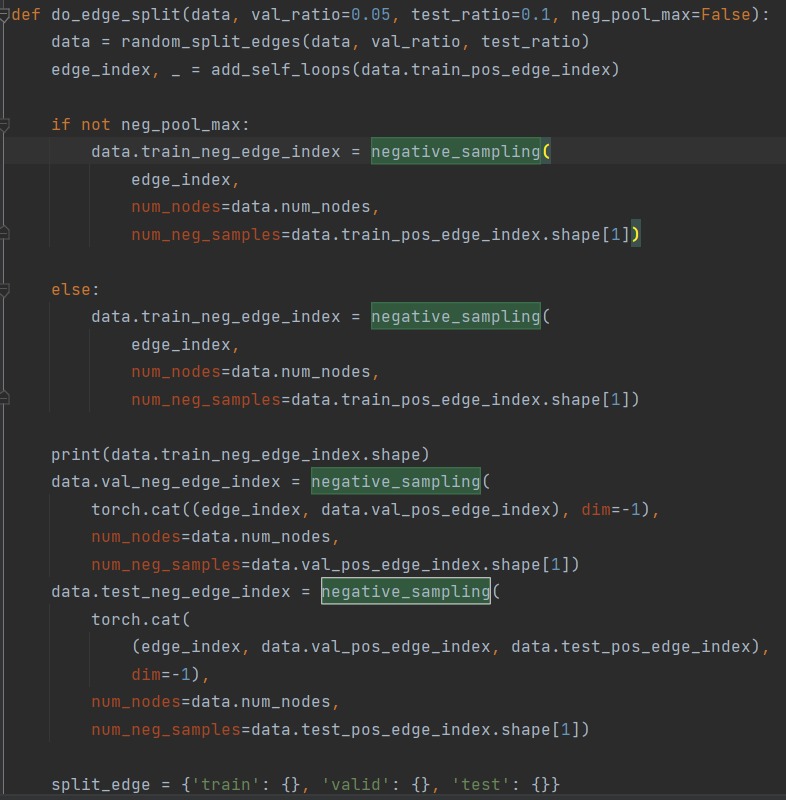
- Sau khi xáo trộn dữ liệu xong thì ta tính kích thước của tập dữ liệu được sử dụng để xác định số lượng mẫu trong tập huấn luyện và tập kiểm tra.

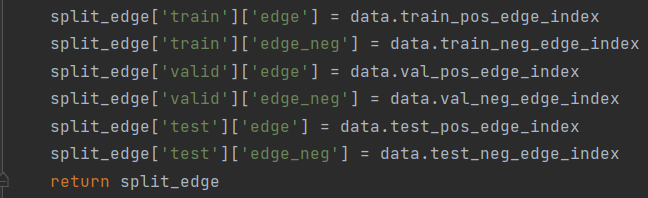
- Cuối cùng, dữ liệu sẽ được chia thành hai phần, một phần cho tập huấn luyện và một phần cho tập kiểm tra, dựa trên kích thước đã tính ở bước trước.

* + 1. **TwoWL**



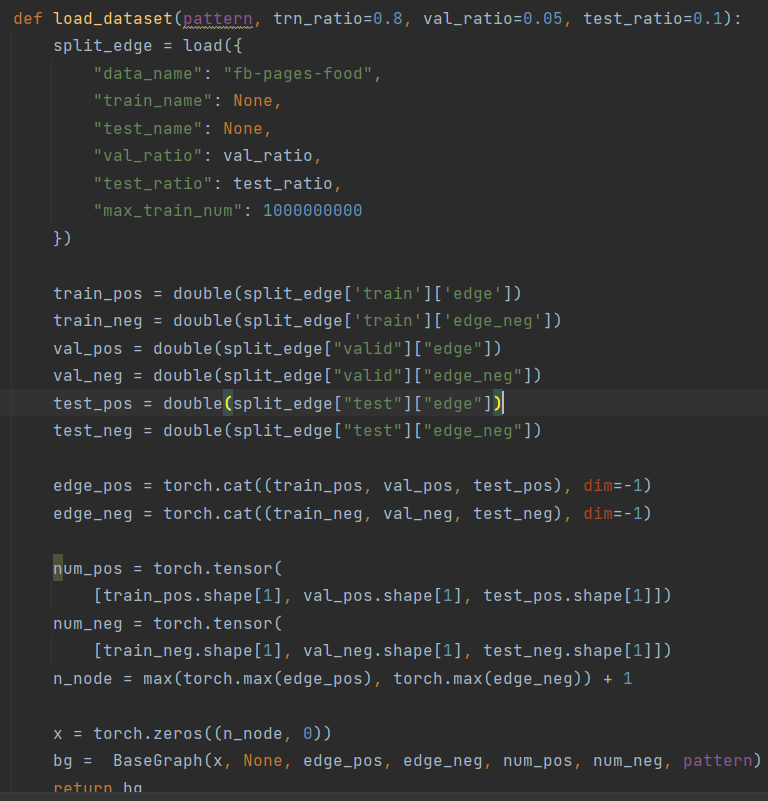
Đoạn code trên dùng để thực hiện việc tải dữ liệu từ một file CSV chứa thông tin về các cạnh trong đồ thị và tiền xử lý nó để chuẩn bị cho việc đào tạo mô hình. Trong đoạn code trên ta thấy có hàm *do\_edge\_split ()* dùng để thực hiện việc chia dữ liệu thành các tập train/val/test với các tham số truyền vào như là val\_ratio, test\_radio và neg\_pool\_max. Hàm này được định nghĩa như sau:





Hàm *do\_edge\_split()* được sử dụng để chia các cạnh trong đồ thị thành các tập train/val/test và tạo các cạnh âm tương ứng cho mỗi tập. Nó thực hiện việc chia tập ngẫu nhiên, thêm các cạnh tự tham chiếu vào tập train, và lấy mẫu các cạnh âm từ đồ thị. Trong đó, các vòng lặp tự do được thêm vào để đảm bảo mỗi đỉnh có ít nhất một cạnh kết nối. Số lượng cạnh âm được tạo dựa trên số lượng cạnh dương trong từng tập. Kết quả trả về là một từ điển chứa các tập cạnh dương và cạnh âm cho mỗi phần của dữ liệu (huấn luyện, validation và test).

Sau khi đã định nghĩa hàm *load()* ở trên, ta tiếp tục gọi hàm này trong hàm *load\_dataset()* để lấy dữ liệu với tỷ lệ các tập train, val, test được xác định cụ thể.



Dữ liệu cạnh dương và cạnh âm của mỗi tập (train, val, test) được ép kiểu thành các đối tượng bằng cách sử dụng hàm double. Sau đó, các cạnh dương và cạnh âm được tổng hợp vào một tensor lớn cho mỗi loại cạnh. Dữ liệu này được sử dụng để tạo một đối tượng graph thông qua lớp “BaseGraph”. Đối tượng này bao gồm các thuộc tính như *edge\_pos, edge\_neg, num\_pos, num\_neg…*

* 1. **Xây dựng mô hình**
     1. **SEAL Framework**

- Khai báo placeholder: Đầu tiên, khai báo các placeholder để chứa dữ liệu đầu vào của mô hình mà không cần biết giá trị cụ thể của chúng trước. Mỗi placeholder đại diện cho một loại dữ liệu như ma trận nghịch đảo D-1 (*D\_inverse\_pl*), ma trận kề chuẩn hóa (*A\_tilde\_pl*), ma trận đặc trưng cho các nút (*X\_pl*), nhãn (*Y\_pl*), số lượng nút (*node\_size\_pl*), biến chỉ dẫn chế độ huấn luyện/dự đoán (*is\_train*).

- Khai báo các biến trainable: Khai báo các biến trọng số có thể được huấn luyện trong lớp Graph Convolution. Các biến này bao gồm các ma trận trọng số cho các lớp Graph Convolution, được khởi tạo ngẫu nhiên với phân phối chuẩn bằng phương pháp *Truncated Normal* với độ lệch chuẩn 0.1.

- Lớp Graph Convolution (GCN): GCN là lớp chính trong mô hình, thực hiện các phép tính để biến đổi ma trận đặc trưng đầu vào thành ma trận đặc trưng đầu ra. Trong này ta triển khai GCN với 4 lớp cụ thể như sau:

+ Lớp GCN đầu tiên:

+ Lớp GCN thứ hai:

+ Lớp GCN thứ ba:

+ Lớp GCN thứ tư:

Trong đó: là ma trận nghịch đảo (*D\_inverse\_pl*), là ma trận kề chuẩn hóa (*A\_tilde\_pl*), là ma trận đặc trưng cho các nút (*X\_pl*), là kết quả đầu ra của mỗi lớp trong GCN.

Sau khi có được kết quả của từng lớp, ta thực hiện nối các kết quả đó lại với nhau theo chiều ngang thành một tensor duy nhất (*graph\_conv\_output*) có kích thước (node\_size, 96).

- Lớp Thresholding Pooling: Lớp này tính toán một ngưỡng dựa trên phần trăm của kích thước nút trong một tập hợp các đồ thị. Nó giúp giảm kích thước của đầu ra của mô hình, giảm độ phức tạp tính toán.

+ Đầu tiên, tính toán danh sách kích thước các đồ thị trong tập train và test.

+ Sau đó, áp dụng phép tính *percentile* để tính giá trị *threshold\_k* dựa theo *top\_k%* của danh sách trên.

+ Tiếp đến, ta lấy chỉ số của *k* giá trị lớn nhất trong những giá trị của cột đầu tiên của trong .

+ Cuối cùng, dùng hàm *gather* trong *tensorflow* để sử dụng các chỉ số vừa tính bên trên để lấy các phần tử tương ứng trong *graph\_conv\_output* và gán nó vào biến *graph\_conv\_output\_stored*. Việc dùng kết quả của lớp thứ 4 trong GCN để thực hiện việc pooling vì nó mang thông tin tổng hợp và đầy đủ nhất về cấu trúc đồ thị sau 3 lần tính toán trước, đồng thời nó cho phép xác định được các nút quan trọng nhất một cách toàn diện hơn so với các kết quả của các lớp trước đó.

+ Cuối cùng, nếu kích thước của nút nhỏ hơn ngưỡng *k* ta đã tính trước đó, thì ta tiến hành nối *graph\_conv\_output\_stored* với mảng *zeros* để bổ sung các phần tử còn thiếu. Ngược lại, ta tiến hành cắt *graph\_conv\_output\_stored* thành độ dài *threshold\_k* và gán vào biến *graph\_conv\_output\_top\_k*.

- Lớp Flatten: Lớp này được sử dụng để biến đổi kết quả của bước pooling trước (*graph\_conv\_output\_top\_k*) thành một tensor mới với hình dạng 1D phù hợp cho việc sử dụng các lớp neural network tiếp theo, với kích thước là *GRAPH\_CONV\_LAYER\_CHANNEL\*3\*threshold\_k*.

- Lớp Convolution 1D: Lớp nãy sẽ áp dụng tichs chập 1 chiều với các bộ lọc được khởi tạo.

+ Lớp Convolutional 1D đầu tiên (*conv1d\_kernel\_1*):

* Ở đây, chúng ta định nghĩa một kernel cho lớp convolutional 1D bằng cách sử dụng biến *conv1d\_kernel\_1*. Kernel này có kích thước *shape = [CONV1D\_1\_FILTER\_WIDTH, 1, CONV1D\_1\_OUTPUT]*, trong đó *CONV1D\_1\_FILTER\_WIDTH* là chiều rộng của kernel, *1* là số kênh đầu vào, và *CONV1D\_1\_OUTPUT* là số kênh đầu ra.
* Chúng ta sử dụng hàm *tf.nn.conv1d* để thực hiện phép tích chập 1D giữa dữ liệu đầu vào (*graph\_conv\_output\_flatten*) và kernel đã được định nghĩa trước đó.
* Đầu ra của lớp convolutional 1D đầu tiên, được lưu trong biến *conv\_1d\_a*, có kích thước là *[1, threshold\_k, CONV1D\_1\_OUTPUT]*.

+ Lớp Convolutional 1D thứ hai (conv1d\_kernel\_2):

* Tương tự như lớp convolutional đầu tiên, chúng ta định nghĩa một kernel cho lớp convolutional 1D thứ hai bằng biến *conv1d\_kernel\_2*. Kernel này có kích thước *shape = [CONV1D\_2\_FILTER\_WIDTH, CONV1D\_1\_OUTPUT, CONV1D\_2\_OUTPUT]*, trong đó *CONV1D\_2\_FILTER\_WIDTH* là chiều rộng của kernel, *CONV1D\_1\_OUTPUT* là số kênh đầu ra của lớp convolutional trước, và *CONV1D\_2\_OUTPUT* là số kênh đầu ra của lớp convolutional thứ hai.
* Sử dụng hàm *tf.nn.conv1d* để tính toán đầu ra của lớp convolutional thứ hai từ đầu ra của lớp convolutional trước đó.
* Đầu ra của lớp convolutional 1D thứ hai, được lưu trong biến *conv\_1d\_b*, có kích thước là *[1, threshold\_k - CONV1D\_2\_FILTER\_WIDTH + 1, CONV1D\_2\_OUTPUT]*.

- Lớp Flatten:

+ Lớp này được sử dụng để biến đổi tensor đầu ra từ lớp convolutional thành một vector dài, để có thể đưa vào các lớp fully connected sau đó.

+ Đầu ra (*conv\_output\_flatten*) của lớp này có kích thước *[1, (threshold\_k - CONV1D\_2\_FILTER\_WIDTH + 1) \* CONV1D\_2\_OUTPUT]*.

- Lớp Fully Connected:

+ Lớp Fully Connected 1:

* Lớp fully connected đầu tiên được tạo ra bằng cách nhân đầu ra của lớp flatten với ma trận trọng số *weight\_1* và cộng với bias *bias\_1*. Trong đó, *weight\_1* được khởi tạo với giá trị ngẫu nhiên từ phân phối chuẩn có mean là 0 và độ lệch chuẩn là 0.1*.* Kích thước của *weight\_1* là *[conv\_output\_flatten.shape[1], DENSE\_NODES], bias\_1* được khởi tạo là một vector có kích thước *[DENSE\_NODES]* và các phần tử đều có giá trị 0.
* Đầu ra của lớp này được đưa qua hàm kích hoạt ReLU để tạo ra một không gian đặc trưng mới và được lưu trong biến *dense\_z*.
* Nếu đang trong quá trình huấn luyện (*is\_train == 1*), chúng ta sẽ áp dụng *dropout* lên đầu ra từ lớp dense đầu tiên. Việc này giúp ngăn chặn hiện tượng overfitting bằng cách loại bỏ ngẫu nhiên một phần của đầu ra.

+ Lớp Fully Connected 2:

* Lớp fully connected thứ hai tạo ra đầu ra cuối cùng của mạng neural network bằng cách nhân đầu ra của lớp trước với ma trận trọng số *weight\_2* và cộng với bias *bias\_2*. Trong đó, *weight\_2* được khởi tạo ngẫu nhiên từ phân phối chuẩn có mean là 0 và độ lệch chuẩn là 0.1. Kích thước của weight\_2 là [*DENSE\_NODES, 2*], *bias\_2* là một vector có kích thước [*2*] và các phần tử đều có giá trị 0.
* Đầu ra cuối cùng (*pre\_y*) là kết quả trước khi áp dụng hàm softmax.

- Hàm softmax và loss function:

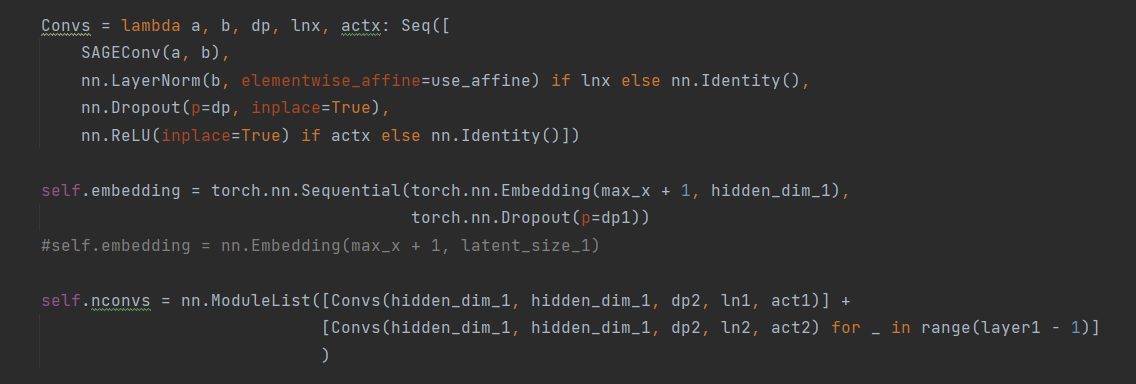
+ Kết quả của mạng neural network được đưa qua hàm softmax để tính xác suất phân loại. Điều này cho phép mô hình đưa ra dự đoán dưới dạng xác suất.

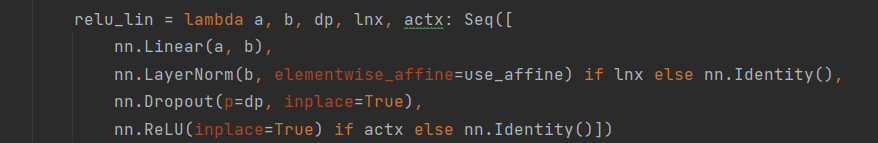
+ Sử dụng hàm softmax cross-entropy để tính toán hàm mất mát giữa dự đoán và nhãn thực tế. Hàm này sẽ tính trung bình của cross-entropy loss giữa dự đoán của mạng (*pre\_y*) và nhãn thực tế (*Y\_pl*). Mục tiêu của quá trình huấn luyện là tối thiểu hóa hàm mất mát này.

+ Sử dụng thuật toán tối ưu hóa AdamOptimizer để điều chỉnh các trọng số của mạng neural network dựa trên hàm mất mát được tính toán

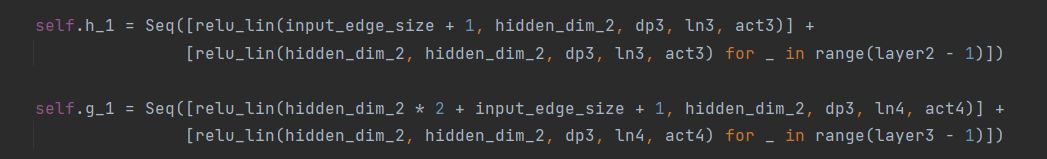
* + 1. **Mô hình 2WL**

*Biểu diễn đặc trưng đỉnh:* Mô hình sử dụng một lớp nhúng (embedding layer) để biểu diễn các đặc trưng của đỉnh trong đồ thị. Đặc trưng của mỗi đỉnh được biểu diễn dưới dạng một vectơ trong không gian latent với kích thước *hidden\_dim\_1*.

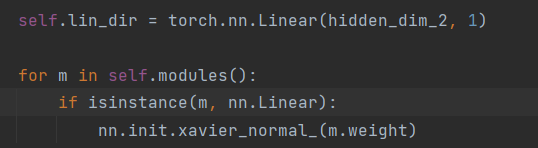
*Lớp tích chập đỉnh (Node Convolutional Layers):* Sau khi biểu diễn các đặc trưng đỉnh, mô hình sử dụng các lớp tích chập đỉnh (Node Convolutional Layers) để áp dụng các phép biến đổi đỉnh theo cấu trúc đồ thị. Mỗi lớp tích chập đỉnh bao gồm một lớp SAGEConv để tính toán biểu diễn đỉnh mới và các lớp điều chuẩn (Layer Normalization) và dropout để kiểm soát quá trình học.

*Lớp kết hợp (Aggregation Layer):* Mô hình sử dụng lớp kết hợp để tổng hợp thông tin từ các đỉnh láng giềng và đặc trưng đỉnh đã được biểu diễn. Lớp kết hợp này được định nghĩa bằng một hàm lambda có tên là relu\_lin. Hàm này nhận vào các tham số là *a, b, dp, lnx*, và *actx*, và trả về một chuỗi các lớp mạng nơ-ron. Điều này giúp mô hình học được biểu diễn toàn cục của đồ thị

Tổng quan 2 lớp kết hợp được định nghĩa như sau:



*Lớp kết quả (Output Layer):* Cuối cùng, mô hình sử dụng một lớp tuyến tính để dự đoán kết quả cuối cùng từ biểu diễn học được.



* 1. **Huấn luyện mô hình**
     1. **SEAL Framework**

- Gán các biến từ mô hình vào các biến cục bộ: Ở đây, các biến trong mô hình đã được khai báo trước đó được gán cho các biến tương ứng trong hàm huấn luyện. Các biến này sẽ được sử dụng để xây dựng các phần của đồ thị tính toán TensorFlow.

- Sau đó, ta tính kích thước dữ liệu huấn luyện

- Tiếp theo, ta bắt đầu một phiên làm việc TensorFlow và khởi tạo một saver để lưu trạng thái của mô hình.

- Với mỗi vòng lặp, ta khởi tạo các biến trong phiên làm việc.

- Trong mỗi epoch, một vòng lặp nội bộ được sử dụng để lặp qua từng mẫu dữ liệu trong tập huấn luyện:

+ Tính giá trị hàm mất mát (loss):

* Ta tăng giá trị batch\_index lên 1 nếu batch\_index nhỏ hơn train\_data\_size - 1, ngược lại gán batch\_index bằng 0.
* Sau đó, ta tạo dữ liệu đầu vào feed\_dict dựa trên giá trị batch\_index để cung cấp dữ liệu cho các placeholder trong mô hình. Trong trường hợp này, *is\_train* được gán giá trị 1, đại diện cho việc huấn luyện.
* Dựa trên dữ liệu đầu vào từ feed\_dict, ta chạy một bước huấn luyện bằng cách thực hiện các phép tính trong danh sách [loss, train\_op, global\_step] và lấy giá trị của loss: Biến loss\_value được gán giá trị của loss, đại diện cho giá trị hàm mất mát (loss) của mô hình trong quá trình huấn luyện. Biến \_ và \_ là các biến không được sử dụng và dùng để bỏ qua kết quả của các phép tính train\_op và global\_step.

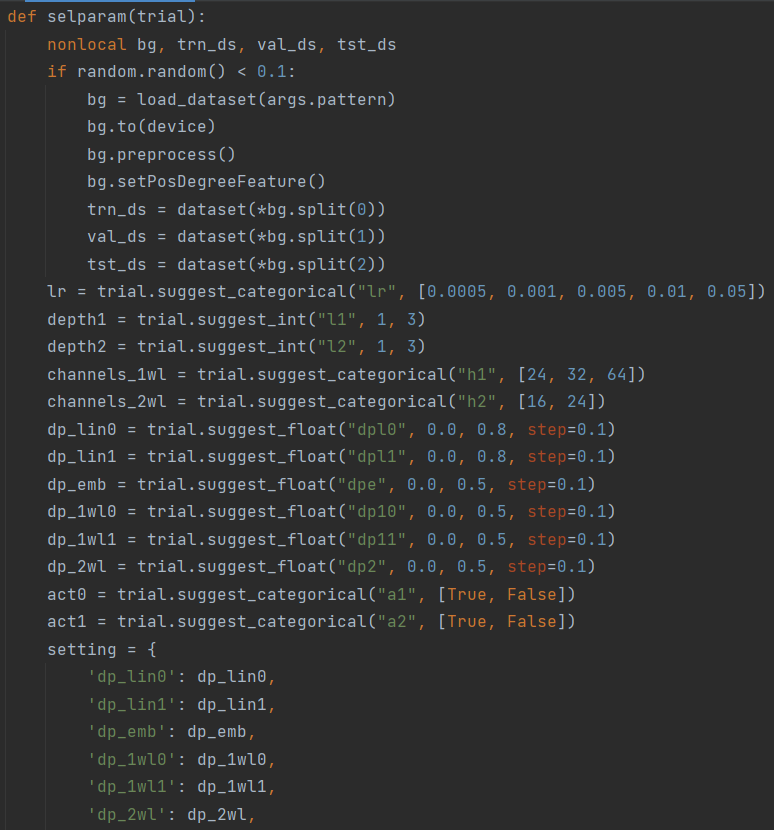
+ Tính toán độ chính xác trên tập huấn luyện:

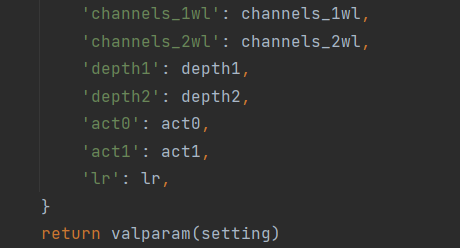
* Ta khởi tạo biến train\_acc với giá trị ban đầu là 0
* Sau đó, ta tạo dữ liệu đầu vào feed\_dict dựa trên giá trị i của từng vòng lặp (I đại diện cho vòng lặp thứ i) để cung cấp dữ liệu cho các placeholder trong mô hình. Trong trường hợp này, is\_train được gán giá trị 0, đại diện cho việc không huấn luyện.
* Dựa trên dữ liệu đầu vào từ feed\_dict, ta chạy phép tính *pre\_y* để dự đoán đầu ra của mô hình trên mẫu huấn luyện hiện tại và gán kết quả vào biến *pre\_y\_value*.
* Sử dụng hàm np.argmax() để lấy chỉ số của phần tử có giá trị lớn nhất trong pre\_y\_value, và so sánh với nhãn thực tế Y\_train[i]. Nếu chỉ số dự đoán trùng khớp với nhãn thực tế, tăng giá trị của train\_acc lên 1.
* Sau khi duyệt qua toàn bộ tập huấn luyện, tính toán độ chính xác bằng cách chia train\_acc cho train\_data\_size, tức số lượng mẫu huấn luyện.

- Cuối cùng, ta lưu trạng thái của mô hình sau khi huấn luyện.

* + 1. **TwoWL**

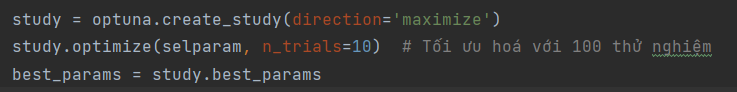
Sau khi quá trình xử lý dữ liệu thành công, mô hình Two-WL-GNN sử dụng thư viện Optuna của Pytorch để tự động điều chỉnh tham số mô hình để mô hình có thể đạt được hiệu năng tốt nhất. Trước khi sử dụng Optuna thì ta cần định nghĩa một hàm để khởi tạo các tham số ban đầu của mô hình cũng như để lựa chọn các tham số cho mô hình dựa trên các giá trị ngẫu nhiên. Hàm này được định nghĩa như sau:





Quá trình lựa chọn tham số cho mô hình được thực hiện thông qua việc thử nghiệm một loạt các giá trị tham số và đánh giá hiệu suất của mô hình với từng cấu hình tham số. Các tham số được chọn bao gồm tốc độ học (lr), độ sâu của lớp mạng (depth1 và depth2), số kênh của mạng 1WL (channels\_1wl) và mạng 2WL (channels\_2wl), độ rơi (dropout) trong quá trình huấn luyện, và các tham số khác. Các giá trị của các tham số được lựa chọn thông qua việc thử nghiệm ngẫu nhiên từ một tập giá trị có sẵn. Sau khi có các tham số ngẫu nhiên ta đưa vào mô hình 2WL để train:



Tiếp đó ta sử dụng thư viện Optuna để tối ưu hóa siêu tham số cho một mô hình. 

Quá trình tối ưu hoá được thực hiện như sau:

*optuna.create\_study(direction='maximize'):*  Đầu tiên, chúng ta tạo một đối tượng nghiên cứu (study) trong Optuna. Trong trường hợp này, chúng ta muốn tối đa hóa *(direction='maximize')* giá trị của hàm mục tiêu.

*study.optimize(selparam, n\_trials=10):* Tiếp theo, chúng ta sử dụng phương thức optimize của đối tượng nghiên cứu để thực hiện quá trình tối ưu hoá. Ta truyền hàm selparam làm đối số, đây là hàm mà Optuna sẽ tối ưu hoá bằng cách thử nghiệm các giá trị tham số khác nhau. n\_trials=10 chỉ định số lượng thử nghiệm (trials) cần thực hiện, trong trường hợp này là 10.

Sau khi quá trình tối ưu hoá hoàn tất, *study.best\_params* chứa các giá trị tham số tốt nhất được tìm thấy trong quá trình tối ưu hoá. Các giá trị tham số này có thể được sử dụng để xây dựng một mô hình với hiệu suất tối ưu.

* 1. **Dự đoán kết quả**
     1. **SEAL Framework**

- Dựa vào mô hình đã lưu ở quá trình huấn luyện mô hình, ta cũng sử dụng một vòng lặp để duyệt qua từng mẫu trong tập kiểm tra.

- Tạo feed\_dict để cung cấp dữ liệu cho các placeholder trong mô hình. Trong trường hợp này, *is\_train* được gán giá trị 0, đại diện cho việc không huấn luyện.

- Chạy phép tính pre\_y và pos\_score để dự đoán đầu ra của mô hình và tính toán điểm tích cực (pos\_score) trên mẫu kiểm tra hiện tại.

- Sử dụng hàm np.argmax() để lấy chỉ số của phần tử có giá trị lớn nhất trong pre\_y\_value, đại diện cho dự đoán của mô hình. Ghi lại dự đoán và điểm số tích cực (pos\_score) vào danh sách prediction và scores.

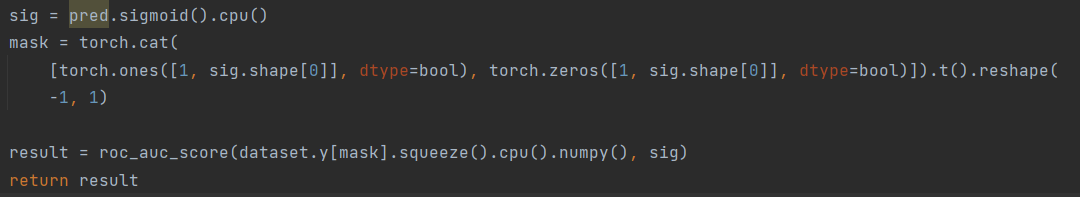
- Nếu chỉ số dự đoán trùng khớp với nhãn thực tế Y\_test[i], tăng giá trị của test\_acc lên 1.

- Tiếp đến, ta tính toán độ chính xác trên tập kiểm tra bằng cách chia test\_acc cho test\_data\_size.

- Cuối cùng, ta dùng hàm metrics.roc\_auc\_score() được sử dụng để tính toán diện tích dưới đường cong ROC (Area Under the ROC Curve - AUC). Với, *y\_true* là nhãn thực tế của các mẫu dữ liệu trong tập kiểm tra và Là điểm số (score) được dự đoán bởi mô hình cho các mẫu trong tập kiểm tra. *pos\_scores* (*scores*) là xác suất hoặc một đại lượng có thể liên quan đến mức độ chắc chắn của mô hình về việc mỗi mẫu thuộc lớp positive (1).

* + 1. **Two-WL**

Sau khi thực hiện qua nhiều vòng lặp để train mô hình, ta được tập các tham số tốt nhất của mô hình. Với các tham số này ta tiến hành chạy mô hình trên tập test. Quá trình đánh giá trên tập test diễn ra như sau:



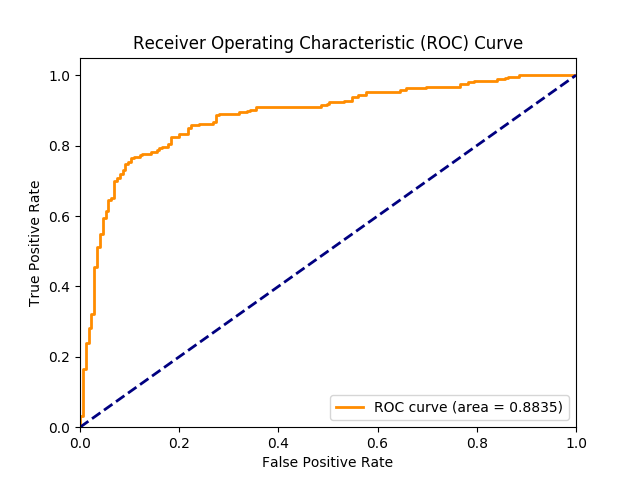
*sig = pred.sigmoid().cpu():* Kết quả dự đoán pred được chuyển qua hàm sigmoid để chuyển đổi thành xác suất. Kết quả này sẽ chứa các xác suất dự đoán cho mỗi mẫu trong tập dữ liệu kiểm tra.

*mask = torch.cat([torch.ones([1, sig.shape[0]], dtype=bool), torch.zeros([1, sig.shape[0]], dtype=bool)]).t().reshape(-1, 1):* Một mask được tạo ra để chỉ định các mẫu dương (positive) và mẫu âm (negative) trong tập dữ liệu. Mask này có cùng kích thước với số lượng mẫu trong tập dữ liệu kiểm tra. Các mẫu dương được đánh dấu là True và các mẫu âm được đánh dấu là False.

*result = roc\_auc\_score(dataset.y[mask].squeeze().cpu().numpy(), sig):* Sử dụng hàm roc\_auc\_score từ thư viện Scikit-learn để tính toán điểm số ROC AUC dựa trên các xác suất dự đoán và nhãn thực tế của các mẫu trong tập dữ liệu kiểm tra. Các nhãn được chọn dựa trên mask đã tạo trước đó. Điểm số ROC AUC cuối cùng được trả về là kết quả của quá trình kiểm tra hiệu suất của mô hình.

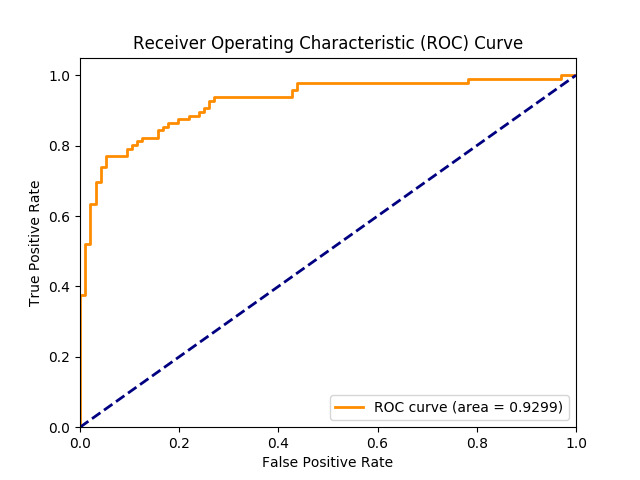
### **Đánh giá và nhận xét kết quả**

* + 1. **SEAL Framework**



Dựa trên kết quả nhận được, biểu đồ AUC của SEAL Framework cho bài toán link prediction là một đường cong dương. Điều này có nghĩa là xác suất phát hiện đúng (TPR) cao hơn xác suất báo động giả (FPR) cho tất cả các ngưỡng. Diện tích dưới đường cong AUC (AUC) là 0,8835, đây là một giá trị cao cho thấy SEAL Framework có khả năng phân biệt tốt giữa các đối tượng mục tiêu với liên kết không thực tế ở tất cả các ngưỡng.

* + 1. **TwoWL**



Dựa trên kết quả nhận được, biểu đồ AUC của TwoWL cho bài toán link prediction là một đường cong dương. Điều này có nghĩa là xác suất phát hiện đúng (TPR) cao hơn xác suất báo động giả (FPR) cho tất cả các ngưỡng. Diện tích dưới đường cong AUC (AUC) là 0,9299, đây là một giá trị cao cho thấy TwoWL có khả năng phân biệt tốt giữa các liên kết thực tế với liên kết không thực tế ở tất cả các ngưỡng.

* + 1. **So sánh**

|  |  |  |
| --- | --- | --- |
|  | **AUC** | **Time(s)** |
| **SEAL Framework** | 0.8835 | 38.4845 |
| **TwoWL** | 0.9299 |  |
| **Logistic regrestion** | 0.8223 | 0.1725 |

- Dựa trên các thông số của bảng trên thì ta có thể đưa ra một vài nhận xét sau:

+ Hiệu suất:

* TwoWL có hiệu suất tốt nhất trong việc phân biệt giữa các liên kết thực tế và liên kết không thực tế, tiếp theo là SEAL Framework và cuối cùng là mô hình logistic.
* Kết luận này dựa trên diện tích dưới đường cong AUC (AUC) và độ dốc của đường cong AUC.

+ Thời gian huấn luyện: TwoWL có thời gian huấn luyện lâu nhất, tiếp theo là SEAL Framework và cuối cùng là mô hình logistic.

- Nhận xét tổng quát:

+ TwoWL là mô hình có hiệu suất tốt nhất trong ba mô hình, nhưng nó cũng có thời gian huấn luyện lâu nhất. SEAL Framework có hiệu suất tốt thứ hai và thời gian huấn luyện nhanh hơn TwoWL. Mô hình logistic có hiệu suất thấp nhất nhưng thời gian huấn luyện nhanh nhất.

+ Mô hình logistic là một mô hình đơn giản, trong khi SEAL Framework và TwoWL là những mô hình phức tạp hơn. Do đó, có thể khó khăn hơn để triển khai và đào tạo SEAL Framework và TwoWL so với mô hình logistic.

+ Dựa trên kết quả nhận được, TwoWL là mô hình hiệu quả nhất trong ba mô hình. Tuy nhiên, SEAL Framework có thể là một lựa chọn tốt nếu cần hiệu suất cao và thời gian huấn luyện không phải là vấn đề. Mô hình logistic là một lựa chọn đơn giản hơn, nhưng hiệu suất của nó có thể thấp hơn so với SEAL Framework và TwoWL.

# **CHƯƠNG 4: KẾT LUẬN**

1. **KẾT LUẬN**

Qua quá trình tìm hiểu, nhóm đã trình bày được những kiến thức tổng quan về mô hình Graph Neural Network như kiến trúc chung, cách hoạt động của mô hình, cũng như tìm hiểu các mô hình cụ thể của mô hình Graph Neural Network (Graph Attention Network, Graph Convolution Network, GraphSAGE), …. Đồng thời qua đó, nhóm cũng đã thực hiện được thực nghiệm để xây dựng chương trình thực hiện các mô hình cụ thể của Graph Neural Network trên các tập dữ liệu khác nhau cho bài toán phân loại nút.

1. **HẠN CHẾ**

* Về mặt lý thuyết: nhóm chưa thật sự hiểu hết về cấu trúc cũng như nguyên lý sâu bên trong của mô hình Graph Neural Networks.
* Về mặt thực nhiệm: nhóm chưa thể xây dựng chương trình áp dụng cho tất cả các mô hình cụ thể của Graph Neural Networks với nhiều tập dữ liệu khác nhau trong từng bài toán khác.

1. **HƯỚNG PHÁT TRIỂN**

Dựa trên cơ sở sẵn có, trong tương lai, chương trình có thể được cải tiến bằng cách xây dựng một website cho nhiều tập dữ liệu khác nhau, sau đó tạo dashboard cho mỗi tập dữ liệu và cho từng bài toán tương ứng khác.

Các lý thuyết, thuật toán được trình bày trong báo cáo có thể được áp dụng để giải quyết các bài toán trong nhiều lĩnh vực khác nhau như xử lý ngôn ngữ tự nhiên, hệ thống gợi ý, phát hiện gian lận, …

Bài báo cáo cũng mở ra nhiều hướng nghiên cứu, khám phá các vấn đề khác của Graph Neural Network mà nhóm còn trình bày thiếu sót.

# **CHƯƠNG 5: TÀI LIỆU THAM KHẢO**