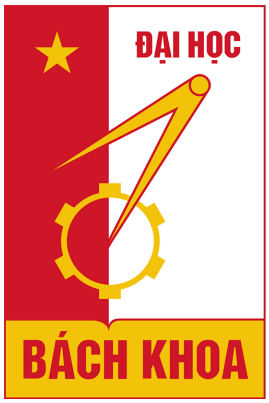
ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG



Báo Cáo Bài tập lớn

Đề tài: Áp dụng Mạng nơ-ron đồ thị cho bài toán gợi ý game trên nền tảng Steam

Học phần: Project II

Mã học phần: IT3930 - Mã lớp: 750645

Giảng viên hướng dẫn: PGS.TS. Lê Đức Hậu

Sinh viên thực hiện: Đặng Tiến Cường 20220020

Hà Nội, Ngày 14 tháng 6 năm 2025

Mục lục

Chương 1

Giới thiệu đề tài

1.1 Đặt vấn đề

Trong kỷ nguyên bùng nổ thông tin, chúng ta đang phải đối mặt với một "biển" thông tin về sản phẩm và dịch vụ xuất hiện trên các nền tảng trực tuyến. Việc tìm kiếm nội dung phù hợp với nhu cầu cá nhân trở thành một thách thức không nhỏ. Trước thực trạng đó, **Hệ thống gợi ý (Recommendation System)** đã ra đời như một giải pháp công nghệ mang tính cá nhân hóa cao, giúp người dùng dễ dàng tiếp cận các lựa chọn phù hợp hơn với sở thích và nhu cầu của mình.

Chắc hẳn, chúng ta đã từng một lần bước vào một cửa hàng, mà "mặt tiền" trưng bày nhiều mặt hàng "bán chạy", các mặt hàng này có thể phù hợp nhu cầu của đa số khách hàng, nhưng không không phù hợp với nhu cầu của ta. Điều mà chúng ta thực sự muốn bây giờ là ngay lập tức thấy được những mặt hàng phù hợp, nhưng điều này trong thực tế là khó, chúng ta chỉ còn cách hỏi nhân viên cửa hàng hoặc "tự mò mẫm" trong một "mớ hỗn độn" các mặt hàng. Tuy nhiên, việc gợi ý mặt hàng phù hợp trên nền tảng trực tuyến là dễ dàng do sự phát triển mạnh mẽ của **Trí tuệ nhân tạo (AI)**, đặc biệt là **Học máy (Machine Learning)**, **Học sâu (Deep Learning)** trong những năm gần đây đã góp phần quan trọng trong việc nâng cao hiệu quả và độ chính xác của các hệ thống gợi ý, đặc biệt là trong môi trường trực tuyến – nơi dữ liệu người dùng và hành vi của họ được thu thập và xử lý liên tục. Nhờ đó, việc đề xuất sản phẩm phù hợp cho từng cá nhân trở nên khả thi và hiệu quả hơn bao giờ hết.

Nhiều tập đoàn công nghệ lớn trên thế giới như **Netflix, Amazon, Steam,...** đã ứng dụng mạnh mẽ hệ thống gợi ý vào hoạt động kinh doanh của mình. Những hệ thống này không chỉ nâng cao trải nghiệm người dùng mà còn góp phần cải thiện tỷ lệ chuyển đổi, tăng mức độ hài lòng của khách hàng và tối ưu hoá doanh thu một cách đáng kể. Thực ra, việc gợi ý sản phẩm cho người dùng cũng rất quen thuộc trong bối cảnh các quy trình nghiệp vụ ngày càng được số hoá như ngày nay, ví dụ như khi ta mua sách online trên ứng dụng Tiki chẳng hạn, ta thực hiện tìm kiếm tên quyển sách, sau đó cho quyển sách vào giỏ hàng thì ngay lập tức ở đâu đó trên màn hình ứng dụng, ta sẽ được Tiki gợi ý những quyển sách liên quan với quyển sách ta định mua, có thể là cùng một tác giả hoặc cùng một chủ đề. Ngoài ra, việc Facebook gợi ý kết bạn khi chúng ta lướt News Feed cũng là một tình huống dễ bắt gặp nhất.

Trong bối cảnh của báo cáo này tôi nghiên cứu một case study cụ thể hơn là: **Hệ thống gợi ý game trên nền tảng Steam**. Việc xây dựng hệ thống gợi ý trên nền tảng Steam thường phải đối diện với những thách thức lớn như:

- **Vấn đề 1 - Khởi đầu lạnh (Cold-Start)**: Khi một nhà phát triển phát hành một **trò chơi mới (Item Cold-Start)** trên Steam thì chúng thường không đủ dữ liệu về người chơi (playtime, đánh giá), hay trường hợp người dùng mới (User Cold-Start) đăng ký tài khoản Steam, hệ thống không có đủ lịch sử để hiểu sở thích của họ. Mô hình dựa trên CF thường kém hiệu quả khi không thể tính toán được độ tương đồng giữa user-user và game-game.
- **Vấn đề 2 - Hiện tượng đuôi dài (Long-Tail Phenonmenon)**: Steam có hơn 50.000 tựa game nhưng khoảng 20% các Game phổ biến sẽ chiếm phần lớn tương tác (recommendation, số lượt chơi) hoặc số giờ chơi, 80% còn lại (các game ít phổ biến) thì nhận được một số ít tương tác cũng như số giờ chơi, tạo nên hiện tượng đuôi dài (long-tail), điều này dẫn đến ma trận tương tác user-item

rất thưa, điều này gây khó khăn cho các mô hình học máy dựa trên CF như kNN, MF, FM,..., có thể tìm pattern chung giữa user-item.

- **Vấn đề 3 - Dữ liệu nhiễu (Noise Data)**: Trên Steam, những tương tác "rác" từ tài khoản ít hoạt động (1–5 review) hay bot shilling rating không chỉ làm loãng ma trận user–item mà còn dẫn đến đề xuất sai lệch, khi mô hình học "thích" những pattern giả tạo thay vì sở thích thật. Bên cạnh đó, các item phụ như DLC hay bundle, vốn không phản ánh trải nghiệm chơi cốt lõi, cũng cần được tách biệt để tránh làm nhiễu tín hiệu chính. Việc thiếu cơ chế phát hiện và loại bỏ hiệu quả các nguồn noise này sẽ làm giảm đáng kể độ tin cậy của hệ gợi ý.

Bộ dữ liệu mà tôi sử dụng trong báo cáo này là **"Game Recommendations on Steam"**, nó có thể được truy cập online thông qua link: https://tinyurl.com/neaky77f. Mặc dù bộ dữ liệu này đã được cung cấp sẵn từ nền tảng Kaggle, việc hiểu rõ nguồn gốc và quy trình thu thập dữ liệu ban đầu vẫn rất quan trọng. Ở đây, nguồn dữ liệu đến từ cơ sở dữ liệu của Steam (tất nhiên được xử lý để không để lộ thông tin riêng tư), giấy phép đi kèm là **CC0: Public Domain**, cho phép tái tạo và sử dụng công khai.

1.2 Tổng quan về bài toán hệ gợi ý

Bản chất của bài toán hệ gợi ý là xếp hạng, trong đó mô hình sẽ tìm cách sắp xếp và đưa ra một nhóm nhỏ các sản phẩm phù hợp nhất trên tập hợp gồm rất nhiều các sản phẩm để gợi ý cho người dùng, dựa trên sở thích cá nhân của mỗi người và các ràng buộc liên quan.

Cụ thể, giả sử ta có một hàm đánh giá mức độ phù hợp giữa người dùng và sản phẩm:

$$f : \mathcal{U} \times \mathcal{I} \to \mathbb{R}$$

trong đó $\mathcal{U}$ là tập người dùng, $\mathcal{I}$ là tập sản phẩm. Mục tiêu của bài toán là học ra một quan hệ thứ tự toàn phần (hoặc xấp xỉ toàn phần) được cá nhân hóa cho mỗi người dùng $u \in \mathcal{U}$:

$$>_u \subseteq \mathcal{I} \times \mathcal{I}$$

sao cho:

$$i >_u j \iff f(u,i) > f(u,j)$$

Một quan hệ $>_u$ được gọi là ***thứ tự toàn phần (total order)*** trên tập $\mathcal{I}$ nếu với mọi $i, j, k \in \mathcal{I}$, quan hệ này thỏa mãn các điều kiện sau:

- **Tính phản xạ (*Irreflexivity*)**:
$$i \in \mathcal{I} \text{ sao cho } i >_u i$$
- **Tính bất đối xứng (*Antisymmetry*)**:
$$i >_u j \Rightarrow \neg(j >_u i)$$
- **Tính bắc cầu (*Transitivity*)**:
$$i >_u j \wedge j >_u k \Rightarrow i >_u k$$
- **Tính khả so sánh toàn phần (*Totality*)**:
$$\forall i, j \in \mathcal{I}, i \neq j \Rightarrow ((i >_u j) \vee (j >_u i))$$

Từ quan hệ thứ tự cá nhân học được, hệ gợi ý sẽ sắp xếp các sản phẩm và đề xuất cho người dùng dựa trên thứ tự này. Hàm đánh giá f có thể được giả định là cố định từ trước, hoặc tùy vào mô hình mà có thể được học cùng quan hệ thứ tự $>_u$.

Để học được một quan hệ thứ tự toàn phần cho hệ gợi ý, có hai hướng tiếp cận chính: ***learn to predict*** và ***learn to rank***.

(1) Learn to predict (pointwise approach): Hướng tiếp cận này xem bài toán là dự đoán *điểm số* phản ánh mức độ người dùng ưa thích một sản phẩm, thông qua một hàm $f(u, i)$. Từ các điểm số này, mô hình có thể xếp hạng các sản phẩm theo thứ tự giảm dần của $f(u, i)$. Tùy vào dạng dữ liệu thu thập được mà bài toán có thể là *regression* hoặc *classification*. Việc huấn luyện thường được thực hiện thông qua tối ưu hóa một hàm mất mát dạng:

$$\min_{\theta} \sum_{(u,i,y_{ui})\in\mathcal{D}} \mathcal{L}_{\text{pred}}(f_\theta(u, i), y_{ui}) + \lambda\mathcal{R}(\theta)$$

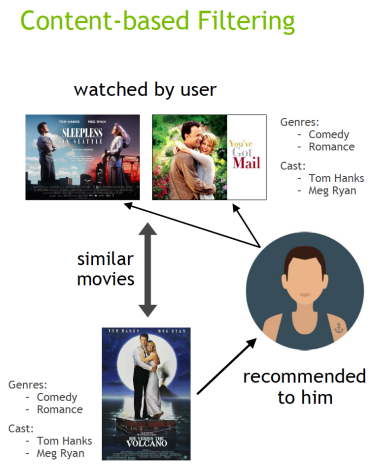
(2) Learn to rank (pairwise hoặc listwise approach): Hướng tiếp cận này không quan tâm đến giá trị tuyệt đối của $f(u, i)$, mà tập trung vào việc học trực tiếp quan hệ thứ tự giữa các sản phẩm, ví dụ như học để đảm bảo $f(u, i) > f(u, j)$ nếu i được ưu tiên hơn j. Nếu thu thập được dữ liệu có thứ tự toàn phần giữa các sản phẩm, mô hình sẽ tập trung tối ưu hóa thứ tự này cho cá nhân mỗi người dùng - hướng tiếp cận *listwise*. Còn nếu dữ liệu có được chỉ là tập thứ tự một phần (hữu hạn các cặp so sánh được), mô hình tìm cách tối ưu hóa khả năng mở rộng thứ tự này thành thứ tự toàn phần - hướng tiếp cận *pairwise*. Quá trình này thường được mô hình hóa dưới dạng bài toán tối ưu sau:

$$\min_{\theta} \sum_{(u,i,j)\in\mathcal{D}} \mathcal{L}_{\text{rank}}\big(f_\theta(u, i) - f_\theta(u, j)\big) + \lambda\mathcal{R}(\theta)$$

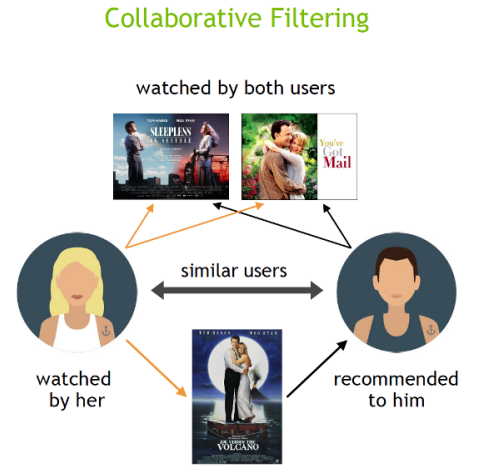
trong đó mỗi bộ ba (u, i, j) biểu thị rằng người dùng u thích sản phẩm i hơn j. Đây có thể xem là một dạng *self-supervised learning*, khi mô hình tự sinh ra nhãn huấn luyện từ dữ liệu tương tác chưa gán nhãn.

Dữ liệu đầu vào của dataset bao gồm 3 loại chính:

- **Thông tin của người dùng (user):** độ tuổi, giới tính, khu vực sinh sống, công việc,...
- **Thông tin của sản phẩm (item): tiêu đề, thể loại, mô tả nội dung, hình ảnh minh họa,...**
- **Lịch sử tương tác giữa user và item**, thường có dạng danh sách các bộ ba (user, item, interaction). Lịch sử này sẽ được chuyển về dạng ma trận tương tác $\mathbf{R} \in \mathbb{R}^{M\times N}$, trong đó mỗi cột ứng với lịch sử của mỗi user còn mỗi hàng ứng với các item.



Hình 1.1: Content-based Filtering



Hình 1.2: Collaborative Filtering

Dựa vào loại dữ liệu được sử dụng để huấn luyện, các mô hình gợi ý có thể được chia thành ba nhóm chính:

- **Content-based Filtering (Lọc theo nội dung):** Phương pháp này sử dụng thông tin nội dung của từng sản phẩm (item) để xây dựng đặc trưng riêng biệt cho từng item. Dựa vào hồ sơ sở thích cá nhân của mỗi người dùng (được hình thành từ lịch sử tương tác trước đó), hệ thống sẽ đề xuất các item có đặc điểm tương đồng với hồ sơ này. Ví dụ: nếu một người dùng thường xuyên xem các bộ phim thuộc thể loại hình sự, hệ thống có thể gợi ý phim *Người phán xử* vì nó chia sẻ đặc trưng thể loại *hình sự* với các phim mà người dùng đã xem.

  - **Ưu điểm:** Khả năng cá nhân hóa cao, độc lập với các người dùng khác. Đặc biệt phòng tránh được tình huống *cold-start item*, khi có sản phẩm mới mà chưa có ai tương tác.
  - **Nhược điểm:** Do chỉ dựa vào thông tin của từng người dùng nên không tận dụng được các xu hướng phổ biến trong toàn bộ hệ thống. Ngoài ra, với các nội dung phức tạp như hình ảnh hoặc âm thanh, việc xử lý và trích xuất đặc trưng có thể gặp nhiều khó khăn.

- **Collaborative Filtering (Lọc cộng tác):** Phương pháp này chủ yếu dựa vào lịch sử tương tác giữa người dùng và sản phẩm để đưa ra gợi ý, mà không cần quan tâm đến nội dung cụ thể của item.
  - **Ưu điểm:** Có khả năng khai thác các xu hướng chung trong cộng đồng người dùng. Đồng thời, không phụ thuộc vào đặc trưng nội dung của sản phẩm nên dễ dàng mở rộng cho nhiều loại item khác nhau.
  - **Nhược điểm:** Rất nhạy cảm với hiện tượng *cold-start* ở cả phía người dùng và sản phẩm, do phụ thuộc mạnh vào dữ liệu tương tác. Để khắc phục, thường cần những kĩ thuật kết hợp thêm thông tin phụ về người dùng và item vào mô hình.

- **Hybrid:** Sử dụng các chiến lược để kết hợp 2 loại mô hình nhằm khắc phục nhược điểm và tối đa hóa ưu điểm của mỗi loại.

Đối với **Collaborative Filtering**, dữ liệu tương tác giữa user và item có thể được chia thành hai loại:

- **Explicit Feedback (Phản hồi tường minh):** Đây là dữ liệu yêu cầu người dùng chủ động đưa ra đánh giá cụ thể về item, ví dụ như cho điểm số (rating) từ 1 đến 5, hoặc like/dislike.
  - ***Ưu điểm:*** Là dữ liệu chất lượng cao, ít nhiễu hơn so với phản hồi ẩn do user thường ít khi quan tâm đến đánh giá, vì nó trực tiếp phản ánh ý kiến của người dùng.
  - ***Nhược điểm:*** Khó thu thập hơn, người dùng thường không có động lực để đánh giá tường minh cho mọi vật phẩm họ tương tác. Hoặc nếu có đánh thì giá thường đánh giá một cách cực đoan, hoặc rất tốt hoặc rất tệ - thay vì suy xét kĩ mức độ yêu thích của bản thân với item. Vì lí do này mà nhiều nền tảng chuyển sang kiểu đánh giá nhị phân (like/dislike) thay vì rating, nhưng chất lượng của *feedback* cũng giảm xuống

- **Implicit Feedback (Phản hồi tiềm ẩn):** Đây là dữ liệu không trực tiếp đến từ đánh giá của người dùng mà đến từ việc hệ thống thu thập dữ liệu về hành vi của người dùng, ví dụ như click vào sản phẩm, thời gian sử dụng, tìm kiếm từ khóa, lượt mua, thêm vào giỏ hàng,...
  - ***Ưu điểm:*** Dễ dàng thu thập với số lượng lớn, vì nó được ghi lại tự động từ các tương tác hàng ngày của người dùng.
  - ***Nhược điểm:***
    * **Khó suy luận sở thích:** Sở thích của người dùng khó có thể suy luận ra từ *implicit feedback* do còn phụ thuộc vào rất nhiều yếu tố ngoài, vì vậy chất lượng kém hơn *explicit feedback*. Ví dụ, một người dùng xem một bộ phim nhiều lần có thể vì họ rất thích nó, hoặc vì họ đang nghiên cứu cho một dự án, hoặc đơn giản là họ để quên.
    * **Không có phản hồi tiêu cực rõ ràng:** Đối với *explicit feedback*, có sự phân biệt rõ ràng giữa rating = 0 (đánh giá tệ) và chưa rating (chưa tương tác với item). Còn đối với *implicit feedback* thì ranh giới này rất mơ hồ và khó phân biệt
    * **Độ nhiễu cao:** Độ nhiễu của *implicit feedback* cũng cao hơn nhiều so với *explicit feedback* do các yếu tố bên ngoài ảnh hưởng đến hành vi người dùng.

Ở báo cáo này tôi chỉ áp dụng các mô hình dựa trên hướng tiếp cận **Collaborative Filtering (CF)** để nghiên cứu bài toán hệ gợi ý.

Chương 2

Tiền xử lý dữ liệu

2.1 Mô tả tổng quan dữ liệu (Data Overview)

Bộ dữ liệu **Game Recommendations on Steam** chứa **hơn 41 triệu** mẫu đề xuất của người dùng ở trên Steam Store (trang mua bán trò chơi, và các vật phẩm liên quan của Steam). Tất cả mẫu dữ liệu đã được làm sạch và tiền xử lý về dạng bảng (file csv) và dạng file json, với các tiêu mục dễ đọc. Bộ dữ liệu gồm ba file dữ liệu quan trọng:

- **games.csv** - file csv chứa thông tin về các trò chơi (hoặc các gói mở rộng), bao gồm đánh giá, giá bán bằng đô la Mỹ $, ngày phát hành. Một số thông tin bổ sung không có dạng bảng về trò chơi, chẳng hạn như mô tả và thẻ, được lưu trong một tệp metadata;
- **users.csv** - bảng chứa thông tin công khai về hồ sơ người dùng: số lượng sản phẩm đã mua và số lượng đánh giá đã đăng;
- **recommendations.csv** - bảng chứa các đánh giá từ người dùng: liệu người dùng có đề xuất một sản phẩm hay không. Bảng này thể hiện mối quan hệ nhiều-nhiều giữa một thực thể trò chơi và một thực thể người dùng.

Các thông tin riêng của game còn được lưu vào một file **games_metadata.json**. Mỗi vật thể trong file lưu các khóa: app_id, description và tags. Để không chệch hướng sang lĩnh vực **xử lý ngôn ngữ tự nhiên (NLP)**, nên tôi sẽ không dùng file này. Bộ dữ liệu không chứa bất kỳ thông tin cá nhân nào về người dùng trên nền tảng Steam. Quy trình tiền xử lý đã ánh xạ tất cả các ID người dùng sang ID mới.

Ta xem xét tổng quan dữ liệu. Để rà soát dữ liệu Về kích thước bộ nhớ, ta sử dụng phương thức **.memory_usage()**; về kích cỡ, ta sử dụng trường .shape; về các cột, ta sử dụng phương thức **.describe()**

- **games.csv** - kích thước 4.86 MB, gồm 50 872 hàng, 13 cột, bao gồm các trường:
  - **app_id:** ID của phần mềm trên nền tảng (kiểu dữ liệu int64)
  - **title:** Tên của app (kiểu dữ liệu string)
  - **date_release:** Ngày phát hành phần mềm trên cửa hàng, theo khuôn dạng YY-MM-DD để phục vụ sắp xếp (kiểu dữ liệu string)
  - **win:** Phần mềm hỗ trợ hệ điều hành Windows không (kiểu dữ liệu bool)
  - **mac:** Phần mềm hỗ trợ hệ điều hành macOS không (kiểu dữ liệu bool)
  - **linux:** Phần mềm hỗ trợ hệ điều hành Linux không (kiểu dữ liệu bool)
  - **rating:** Xếp hạng độ tuổi của phần mềm (kiểu dữ liệu string)
  - **positive_ratio:** Tỷ lệ đánh giá tích cực của người dùng (kiểu dữ liệu int64)
  - **user_reviews:** Tổng số lượt đánh giá của người dùng (kiểu dữ liệu int64)
  - **price_final:** Giá cuối cùng của phần mềm sau khi áp dụng giảm giá (kiểu dữ liệu float64)
  - **price_original:** Giá gốc của phần mềm trước khi áp dụng giảm giá (kiểu dữ liệu float64)
  - **discount:** Phần trăm giảm giá được áp dụng (kiểu dữ liệu float64)

  - **steam_deck:** Phần mềm có hỗ trợ Steam Deck không (kiểu dữ liệu bool)

- **users.csv** - kích thước 193.07 MB, gồm 14 306 064 hàng, 3 cột; bao gồm các trường:
  - **user_id:** ID của người dùng (kiểu dữ liệu int64)
  - **products:** Số lượng sản phẩm mà người dùng đã sở hữu (kiểu dữ liệu int64)
  - **reviews:** Số lượng đánh giá mà người dùng đã thực hiện (kiểu dữ liệu int64)
- **recommendations.csv** - kích thước 2.02 GB, gồm 41 154 794 hàng, 8 cột; bao gồm các trường:
  - **app_id:** ID của phần mềm được đánh giá (kiểu dữ liệu int64)
  - **helpful:** Số lượt đánh giá "hữu ích"cho bài đánh giá (kiểu dữ liệu int64)
  - **funny:** Số lượt đánh giá "hài hước"cho bài đánh giá (kiểu dữ liệu int64)
  - **date:** Ngày người dùng thực hiện đánh giá (kiểu dữ liệu string)
  - **is_recommended:** Trạng thái đề xuất của người dùng (True nếu đề xuất, False nếu không) (kiểu dữ liệu bool)
  - **hours:** Số giờ người dùng đã chơi phần mềm tại thời điểm đánh giá (kiểu dữ liệu float64)
  - **user_id:** ID của người dùng thực hiện đánh giá (kiểu dữ liệu int64)
  - **review_id:** ID của bài đánh giá (kiểu dữ liệu int64)

Dễ thấy file **users.csv** không chứa nhiều thông tin quan trọng nên tôi sẽ không xét đến file này trong suốt quá trình hoàn thành project.

2.2 Phân tích khám phá dữ liệu (EDA)

2.2.1 Phát hiện dữ liệu bị thiếu (missing data)

Tôi thực hiện kiểm tra xem các trường thông tin trong tất cả các file. Sau khi thực hiện thao tác kiểm tra thì tôi đưa ra kết luận rằng: ***bộ dữ liệu không chứa bất kỳ mẫu bị thiếu nào.***

app_id	False
title	False
date_release	False
win	False
mac	False
linux	False
rating	False
positive_ratio	False
user_reviews	False
price_final	False
price_original	False
discount	False
steam_deck	False

Hình 2.1: games.csv

user_id	False
products	False
reviews	False

Hình 2.2: users.csv

app_id	False
helpful	False
funny	False
date	False
is_recommended	False
hours	False
user_id	False
review_id	False

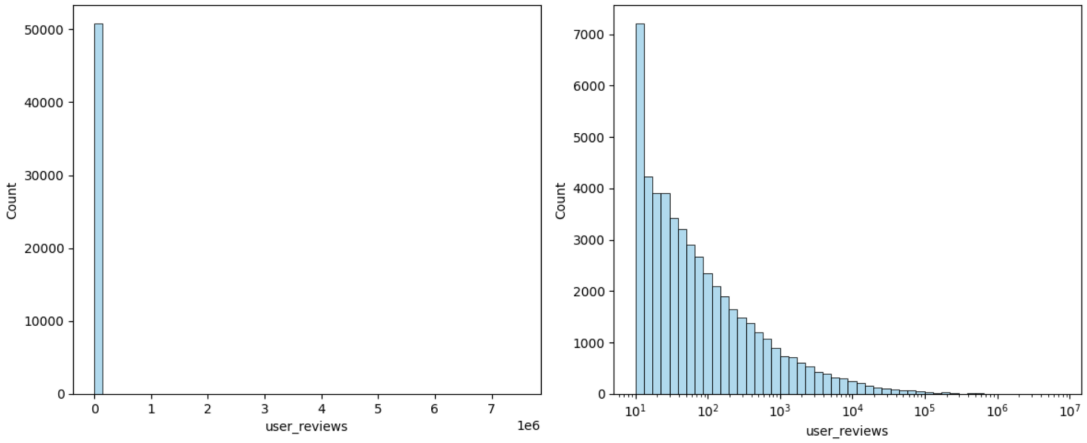
Hình 2.3: recommendations.csv

2.2.2 Phân tích phân phối dữ liệu (data distribution)

Ở thao tác này, tôi chỉ quan tâm đến các trường có thông tin định lượng/ định tính. Các trường thông tin liên quan đến ID của user, game không được xét đến. Vì có rất nhiều trường thông tin, nên tôi trực quan hoá phân phối dữ liệu của một số trường thông tin mang nhiều thông tin quan trọng. Ở file **games.csv** tôi xét đến các trường **date_release, user_reviews, rating, positive_ratio**, còn ở file **recommendations.csv** tôi thực hiện trực quan hoá dữ liệu trên các trường thông tin **date, is_recommended, hours**

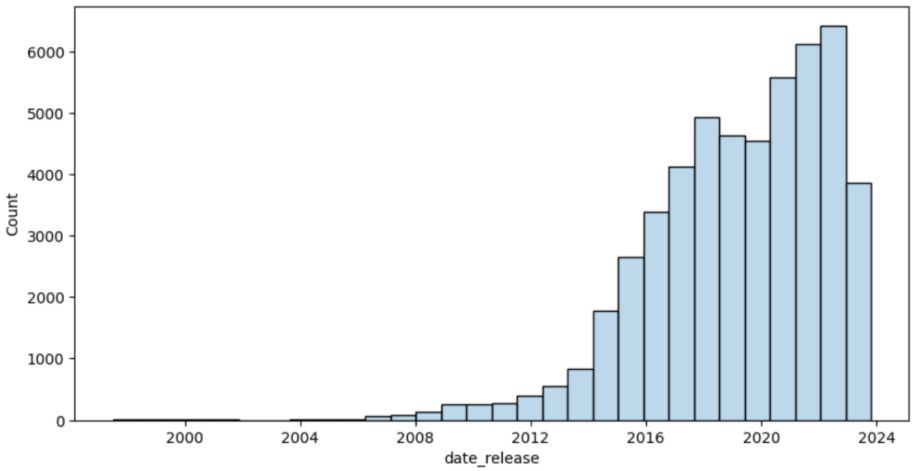
2.2.2.1 games.csv

- **Hình vẽ 2.4** cho thấy hầu hết các game được review rất ít, tuy nhiên một số ngoại lệ lại có được số review rất nhiều, lên đến gần 10 triệu. Khi biểu diễn phân phối áp dụng log-scale vẫn bị long-tailed, điều này cho thấy dữ liệu này tuân theo phân phối pareto [1]



Hình 2.4: Phân phối só lượng đánh giá của người dùng khi chưa và đã log-scale

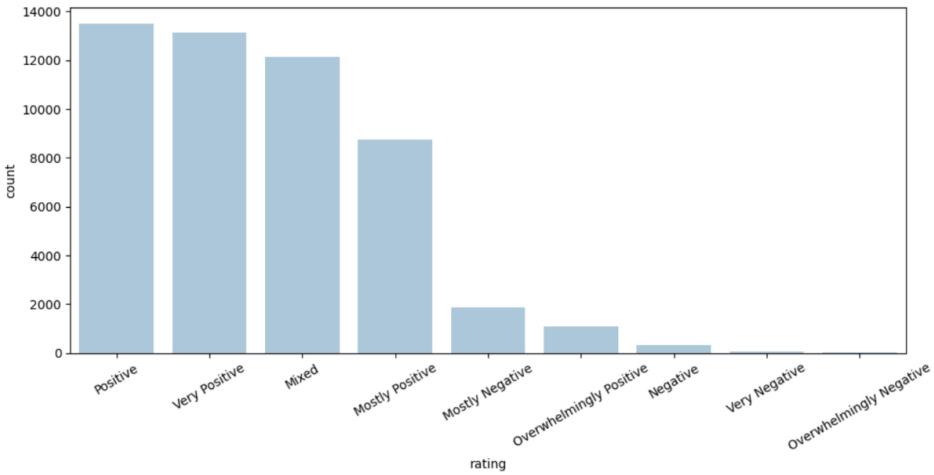
- **Hình vẽ 2.5** cho thấy đa phần các game được ra mắt trong giai đoạn 2020 - 2023 (khả năng có sự tác động của đại dịch COVID-19). Một số ít mẫu dữ liệu ngoại lai (game) có thời gian phát hành quanh mốc 2000.



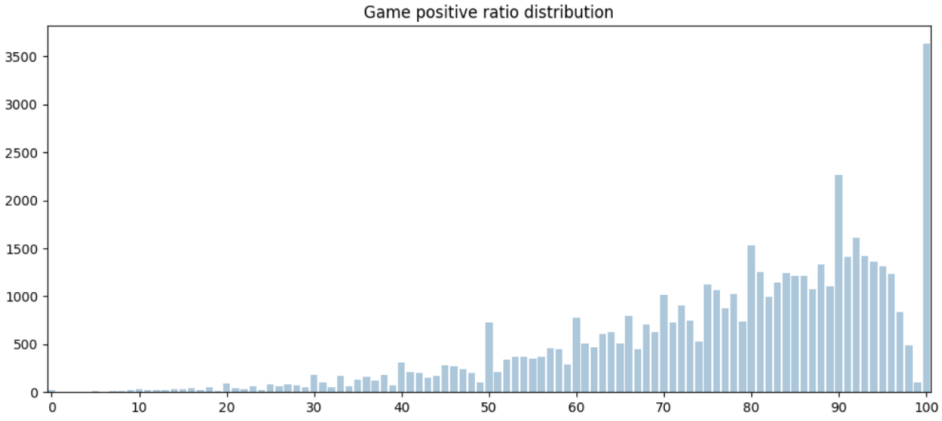
Hình 2.5: Phân phối ngày ra mắt của các game

- **Các hình vẽ 2.6, 2.7** cho thấy phần lớn các loại rating hướng đến đánh giá tích cực, tỉ lệ đánh giá tích cực khá cao, nhảy vọt ở các tỉ lệ phần chục như 70, 80, 90, 100.

[1] https://en.wikipedia.org/wiki/Pareto_distribution



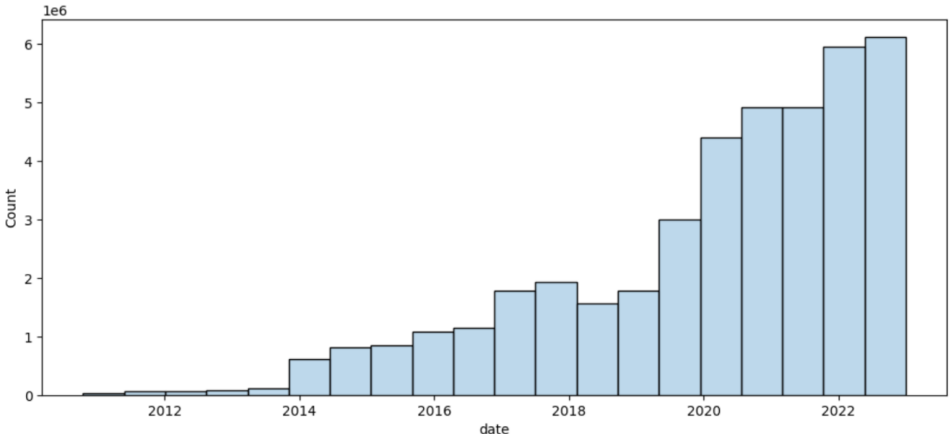
Hình 2.6: Phân phối đánh giá (rating) của người dùng



Hình 2.7: Phân phối tỉ lệ đánh giá của người dùng

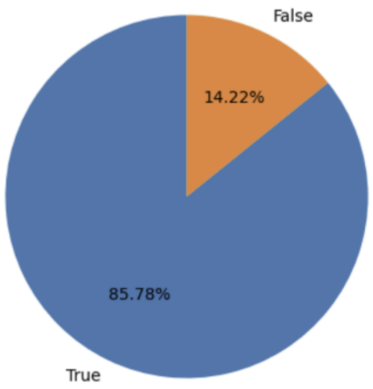
2.2.2.2 recommendations.csv

- **Hình vẽ 2.8** cho thấy phân phối khá tương đồng với trường **date_release** trong file **games.csv**, phản ánh rằng số lượng đánh giá của người dùng tăng lên qua các năm đi cùng với sự phát triển của ngành game.



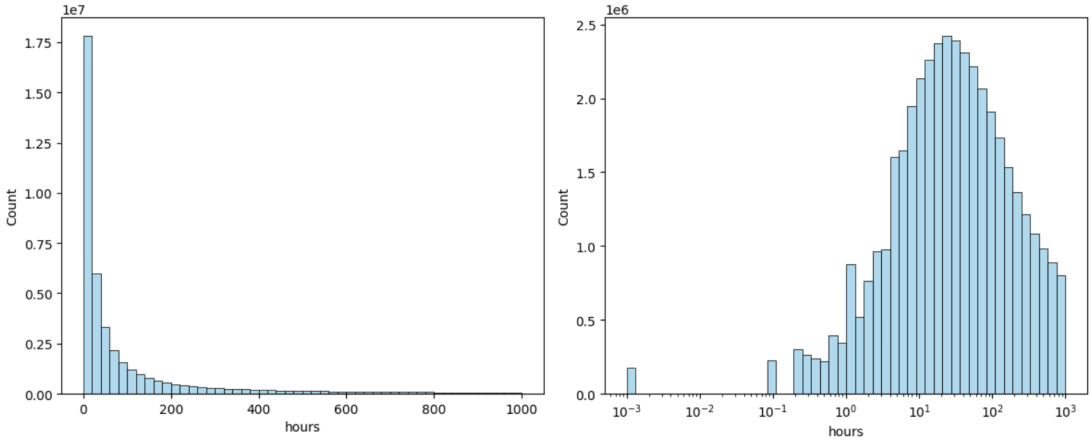
Hình 2.8: Phân phối ngày đánh giá của người dùng

- **Hình vẽ 2.9** cho thấy phần lớn các đánh giá là tích cực, điều này là tương thích về mặt logic với phân phối của các trường **rating, positvie_ratio** ở file **games.csv**. Ngoài ra đây cũng là vấn đề tiềm ẩn về tình trạng mất cân bằng (imbalanced) giữa mẫu dữ liệu.



Hình 2.9: Tỉ lệ người dùng đề xuất game

- **Hình vẽ 2.10** cho thấy phân bố khá đồng đều khi ở log-scale. Phân phối này cho thấy phần lớn người dùng sẽ chơi game từ 10 giờ đến 100 giờ, chơi đã được cắt ở mức trần là 1000 giờ.

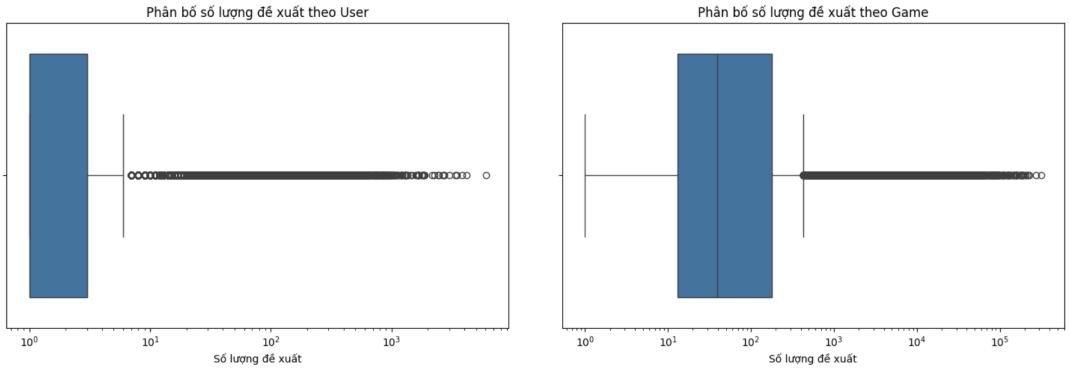


Hình 2.10: Phân phối số giờ chơi game của người dùng khi chưa và đã log-scale

Tóm lại, hầu hết phân phối của các trường thông tin trong bộ dữ liệu là nhất quán, không có sự nhiễu loạn lớn.

2.2.3 Phát hiện ngoại lai (outlier detection)

Tôi thực hiện thao tác đếm số lượng đề xuất của mỗi user, số lượng đề xuất của mỗi game, đồng thời trực quan hoá bằng biểu đồ hộp (box plot). **Hình vẽ 2.11** không chỉ thấy rõ ràng hơn phân bố của các điểm dữ liệu tập trung ở vùng nào, mà còn xác định được các điểm ngoại lai.



Hình 2.11: Phân bố số lượng đề xuất của User/Game

Ta có thể kết luận được rằng:

- **Số lượng đánh giá của user** đa phần nằm trong khoảng từ 1 - 10, khi chuẩn bị dữ liệu ta cần lọc và loại bỏ các user ít nhiệt tính, chỉ nên xét đến các user nhiệt tình có từ 30-50 đánh giá trở lên.
- **Số lượng đánh giá cho game** đa phần nằm trong khoảng từ 10 - 100, khi chuẩn bị dữ liệu ta cần lọc và loại bỏ các game chưa thực sự nổi tiếng do quá ít đánh giá, chỉ nên xét các game phổ biến hơn có số lượng đánh giá từ 1000 trở lên.
  - **Lý giải rõ hơn vấn đề này:** Ở đây, rất nhiều game có positive_ratio đạt 99-100 phần trăm. Tuy nhiên các game này chẳng qua có số review rất ít, nên positive_ratio mới ở mức tuyệt đối như vậy. Số lượng review này có thể được làm giả dễ dàng bởi các shilling bot.

2.3 Chuẩn bị dữ liệu (Data Preparation)

Như đã đề cập, báo cáo này tôi nghiên cứu bài toán hệ gợi ý theo hướng tiếp cận **lọc cộng tác (collaborative filtering)**, do đó dữ liệu đầu vào cho các mô hình CF cần có là *lịch sử tương tác (hisorical interaction)* của user với item. Rõ ràng, chỉ có file **recommendations.csv** mới chứa loại dữ liệu này. Trong đó:

- **is_recommended** (user có đề xuất game hay không?) đại diện cho **phản hồi tường minh (explicit feedback)**
- **hours** (số giờ chơi của user) đại diện cho **phản hồi ngầm định (implicit feedback)**

Do đó, khi chuẩn bị dữ liệu. Tôi chỉ sử dụng file **recommendations.csv** là file chính cho dữ liệu đầu vào cho các mô hình CF, file **games.csv** với các trường thông tin như **date_release**, **user_reviews** được sử dụng như là các tiêu chí để lọc ra các game phù hợp.

Toàn bộ quy trình chuẩn bị dữ liệu như sau:

1. **Làm sạch dữ liệu (data cleaning):** Loại bỏ các bản ghi trùng lặp và dữ liệu bị thiếu, tức là loại bỏ game có trong **games.csv** nhưng không có trong **recommendations.csv** và loại bỏ đánh giá cho các game có trong **recommendations.csv** nhưng không có trong **games.csv.**
2. **Biến đổi dữ liệu (data transformation):** Thay đổi định dạng dữ liệu trên các trường thông tin **is_recommended**, **hours**:
   - Với trường **is_recommended**, tôi thực hiện chuyển đổi định dạng dữ liệu sang kiểu int, trong đó **is_recommended = True** sẽ được chuyển thành **is_recommended = 1**, ngược lại **is_recommended = False** sẽ được chuyển thành **is_recommended = 0**.
   - Với trường **hours**, tôi thực hiện log hoá số giờ chơi bằng công thức **log10(1 + hours)**, mục đích là nén phạm vi dữ liệu lớn, giảm ảnh hưởng của ngoại lai, và tạo phân phối gần chuẩn hơn, phù hợp với các thuật toán học máy. Trường mới có tên là hours_log sau đó được thêm vào bộ dữ liệu **recommendations.csv.**

3. Lọc ra các game mới từ năm 2009 đến năm 2023 (2009 <= date_release <= 2023) và có số lượng đánh giá (review) lớn hơn 1000 (user_reviews >= 1000).

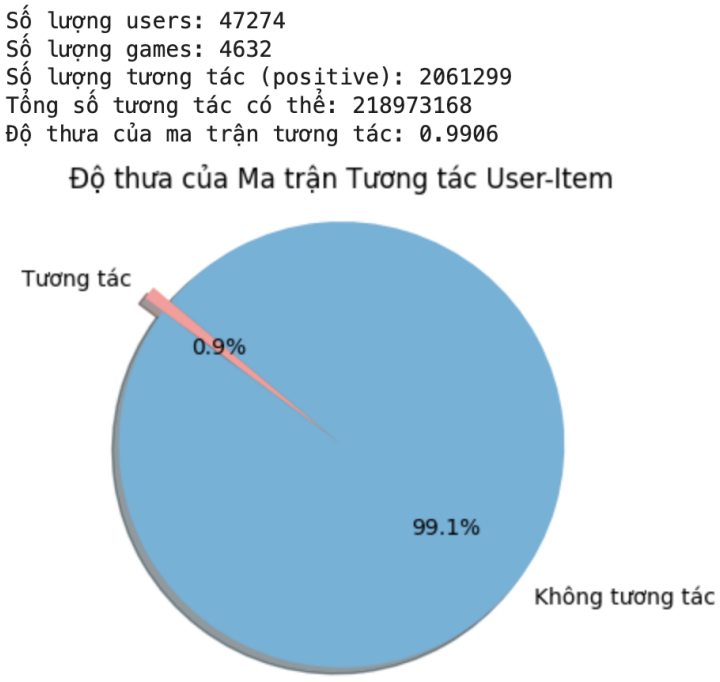
4. Lọc ra các đề xuất chỉ sau khi khi game đã phát hành chính thức (pre-release), tránh các đánh giá trên các phiên bản dùng thử trước khi game được phát hành.

5. Lọc ra các user nhiệt tình có từ 30 đánh giá trở lên để tránh những đánh giá thiên kiến từ những user ít nhiệt tình.

6. Sắp xếp tập dữ liệu theo thời gian, để tránh hiện tượng **data leakage**, sau đó chia tập dữ liệu theo tỉ lệ xấp xỉ **80/20 (80% cho train, 20% cho test)**, Tiếp tục lọc ra các user thoả mãn tiêu chí sau:

   - User này cần có ít nhất **một đánh giá tích cực (positive rating) (is_recommended = 1)** và một **đánh giá tiêu cực (negative rating) (is_recommended = 0)** trong tập train.
   - User này cần có ít nhất **đánh giá tích cực (positive rating) (is_recommended = 1)** trong tập test.

7. Lưu lại tập dữ liệu sau khi lọc và chuyển thành file csv. File cuối cùng sau các bước trên là **file recommendations_processed.csv.**. Kích thước tập dữ liệu sau khi lọc là 2535033 bản ghi, cùng 8 trường sau khi đã lọc đi các trường không có ý nghĩa là **funny, helpful** và thêm trường **hours_log.** Hình 2.12 cho biết thống kê đơn giản về số lượng user, số lượng item, độ thưa của ma trận tương tác user-item. Mặc dù đã lọc ra những user nhiều tương tác nhưng độ thưa vẫn rất lớn lên đến 99.1%.



Hình 2.12: Thống kê đơn giản về dữ liệu sau quá trình tiền xử lý

Chương 3

Các mô hình CF áp dụng

Trong báo cáo này tất cả các mô hình mà tôi áp dụng đều dựa trên việc **học biểu diễn (representation learning)**. Phương pháp này tập trung vào việc học các **biểu diễn tiềm ẩn (latent representations)** cho người dùng (user) và sản phẩm (item) trực tiếp từ dữ liệu tương tác.

Mục tiêu chính của hướng tiếp cận là tự động trích xuất và học **các vector đặc trưng ẩn (embeddings) cho user và item dựa trên lịch sử đánh giá (ratings)**, thay thế cho việc thiết kế đặc trưng thủ công. Các vector này cho phép ước lượng mức độ tương đồng giữa user và item, hỗ trợ việc dự đoán mức độ yêu thích hoặc tương tác trong các mô hình **MF, NeuMF, NGCF, LightGCN,...**

Theo đó, quy trình chung bao gồm:

1. Xây dựng ma trận tương tác user–item.
2. Thiết lập kiến trúc học biểu diễn **(MF, NeuMF, NGCF, LightGCN,..)**
3. Huấn luyện mô hình để học embeddings tối ưu dưới hàm mất mát phù hợp **(BCE, BPR).**
4. Sử dụng embeddings thu được để dự đoán rating hoặc xếp hạng đề xuất.

3.1 Các mô hình CF dựa trên MF

3.1.1 Matrix Factorization (MF)

Matrix Factorization (MF) hướng đến việc ánh xạ **người dùng (user)** và **sản phẩm (item)** vào một **không gian đặc trưng tiềm ẩn (*latent feature space*)**. Trong không gian này, **mỗi user và item được biểu diễn bởi một vector đặc trưng ẩn, còn gọi là vector embedding**, với mỗi chiều đại diện cho một đặc trưng tiềm ẩn không quan sát được trực tiếp.

Nếu xem hàm tính độ phù hợp là tích vô hướng của **user-embedding** và **item-embedding**

$$f(u,i) = u \cdot i = \sum_{k=1}^{n} u_k i_k$$

thì bài toán sẽ trở thành bài toán **phân tích ma trận (matrix factorization)** tương tác $\mathbf{R} \in \mathbb{R}^{M \times N}$ thành tích của 2 ma trận:

$$\mathbf{R} \approx \hat{\mathbf{R}} = \mathbf{U}\mathbf{V}^\top$$

trong đó $\mathbf{U} \in \mathbb{R}^{M \times n}$ là ma trận embedding của người dùng và $\mathbf{V} \in \mathbb{R}^{N \times n}$ là ma trận embedding của sản phẩm. Mỗi cột của ma trận $\mathbf{U}, \mathbf{V}$ được gọi là một ***đặc trưng (feature)***, đại diện cho một đặc trưng ẩn nào đó trong dữ liệu được nhúng thông qua các số trong cột. Chú ý rằng số thuộc tính của $\mathbf{U}$ và $\mathbf{V}$ là bằng nhau. Khi sử dụng phép nhân vô hướng, các phần tử cùng thuộc tính của hai ma trận sẽ nhân với nhau; tất cả tích này sẽ được cộng lại, thu về một dự đoán. Vì thế, các thuộc tính có sự tương đồng cao khi nhân lại sẽ ra kết quả lớn, các thuộc tính ít tương đồng sẽ triệt tiêu nhau. Điều này khá giống cơ chế hoạt động của độ đo tương đồng **cosine similarity**. Nhìn chung, mỗi phần tử của **ma trận tương tác user-item (interaction matrix)** chính là một tổ hợp tuyến tính các đặc trưng (feature) của cặp

user-item đó. Đây là lí do phương pháp này được xếp vào loại **lọc cộng tác (collaborative filtering).**

Xét mô hình **Matrix Factorization** trong trường hợp đơn giản nhất, khi đầu vào là dạng feedback với giá trị số thực liên tục — ví dụ như *explicit feedback* đánh giá theo thang điểm (1–5 sao), hoặc *implicit feedback* được lượng hóa thành các giá trị thực (số lần xem, thời gian xem, tần suất tương tác, v.v.). Khi đó, mục tiêu tối ưu là xấp xỉ càng gần càng tốt các giá trị trong ma trận $\mathbf{R}$. Hàm mất mát phổ biến để đo lường sai lệch giữa dự đoán và giá trị là hàm **Mean Squared Error (MSE)**.

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(u,i)\in\mathcal{D}} (r_{ui} - \hat{r}_{ui})^2$$

Nhìn kĩ hơn, bài toán tối ưu trên **không phải** là bài toán quy hoạch lồi. Ngoài ra, cũng không tồn tại giải thuật chính xác được xây dựng ở thời điểm này. Vì thế, các phương pháp giải đều đi đến việc **xấp xỉ** lời giải dần qua các phương pháp lặp (iterative methods).

Một trong những giải thuật được sử dụng phổ biến là **Gradient Descent (GD):** Đây là phương pháp cơ bản để tìm cực tiểu của các bài toán ***quy hoạch phi tuyến không ràng buộc***. Ý tưởng là cập nhật dần tham số theo hướng ngược chiều *gradient* của hàm mất mát, với một tốc độ học đã cho. Phương pháp này tuy đơn giản nhưng còn tiềm ẩn nhiều vấn đề như lựa chọn tốc độ học phù hợp, tính *gradient* khó khăn trên tập huấn luyện lớn hay dễ bị kẹt tại các điểm yên ngựa (*saddle point*) hoặc cực tiểu cục bộ. Để khắc phục những nhược điểm này, nhiều biến thể của **Gradient Descent (GD)** đã được phát triển và sử dụng rộng rãi, mà nổi tiếng nhất là **Stochastic Gradient Descent (SGD)** hay **Adaptive Moment Estimation (Adam)**.

3.1.2 Neural Matrix Factorization (NeuMF)

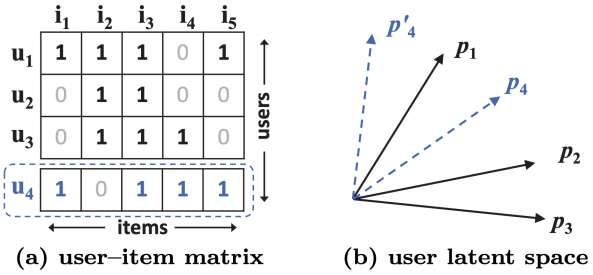
Nói một cách đơn giản, mô hình **NeuMF** là sự kết hợp của **GMF** và **MLP**. Tuy nhiên trước khi trình bày về hai kiến trúc này, tôi sẽ trình bày sơ qua về hạn chế của **MF.**

3.1.2.1 Hạn chế của MF

Hạn chế của MF nằm ở chỗ nó giả định rằng các đặc trưng ẩn của embedding người dùng và sản phẩm là độc lập và tính ra giá trị tương tác bằng cách tổ hợp các chiều đó lại với cùng một trọng số. Điều này đồng nghĩa với việc mô hình MF chỉ có khả năng mô hình hóa các quan hệ tuyến tính giữa các đặc trưng ẩn mà chưa xét được đến các quan hệ phi tuyến phức tạp.

Ví dụ trong Hình 3.1 chứng minh rằng hàm tích vô hướng có thể giới hạn chất lượng của MF, sử dụng ***Cosine similarity*** để đánh giá độ tương đồng giữa 2 embedding của người dùng.

Từ số liệu của ba dòng đầu tiên của hình (a) của ví dụ, ta sẽ tính được $s_{2,3}(0.66) > s_{1,2}(0.5) > s_{1,3}(0.4)$. Như vậy, quan hệ trên không gian của p_1, p_2, p_3 (p_u là vector ẩn của user u) có thể được vẽ như trong hình (b) của ví dụ. Xét user u_4 có $s_{4,1}(0.6) > s_{4,3}(0.4) > s_{4,2}(0.2)$, điều này có nghĩa là user u_4 tương đồng nhất với u_1, tiếp theo là u_3 và cuối cùng là u_2. Tuy nhiên, do mô hình MF đưa vector người dùng lên cùng một không gian ẩn, nên với MF, có 2 cách đặt vector của user u_4 thỏa mãn gần u_1 nhất như hình vẽ trên (2 vector p_4 và p'_4). Nhưng cả hai trường hợp này đều không thể thỏa mãn được tính chất thực tế là u_4 gần u_3 hơn u_2. Từ đó, có thể thấy rằng MF không thể mô tả được độ đo Cosine hay chính là độ đo sự tương tự giữa các người dùng trong trường hợp này.



(a) user–item matrix **(b) user latent space**

Hình 3.1: Ví dụ về hạn chế của MF

Qua ví dụ trên, ta có thể thấy được MF có một số giới hạn nhất định khi cố định sử dụng một hàm tích trong (inner product) đơn giản để dự đoán cho một tương tác phức tạp giữa user và item. Một trong những cách khắc phục điều này đó là tăng số chiều k của embedding. Tuy nhiên, việc này sẽ làm mất tính tổng quát của mô hình, hay chính là vấn đề overfitting, đặc biệt là trong ma trận thưa.

3.1.2.2 Generalized Matrix Factorization (GMF)

Thoạt nhìn thì Matrix Factorization giống với 1 bài toán *unsupervised learning* - cụ thể là *dimensionality reduction* - hơn là *supervised learning*, khi mà đầu vào là 1 ma trận không nhãn và đầu ra là các vector embedding. Nhưng nếu nhìn nhận dưới một góc độ tổng quát hơn, MF thực chất có thể được diễn giải như một bài toán *supervised learning* - cụ thể hơn là 1 biến thể của *linear regression*.

(1) Đầu vào (Input): Mỗi mẫu dữ liệu đầu vào có dạng $\{(u, i), r\}$ tương ứng với tương tác giữa 1 cặp chỉ số của user và item. Các chỉ số u và i được biểu diễn *one-hot encoding* thành 2 vector $\mathbf{e}_u \in \mathbb{R}^N$ và $\mathbf{e}_i \in \mathbb{R}^M$, trong đó N và M lần lượt là tổng số người dùng và sản phẩm. *One-hot encoding* là phương pháp biểu diễn một giá trị rời rạc dưới dạng vector nhị phân, trong đó chỉ có một phần tử bằng 1 tại vị trí tương ứng với giá trị đó, các phần tử còn lại bằng 0. Ví dụ, nếu có 5 người dùng, người dùng thứ 3 sẽ được biểu diễn bởi vector $\mathbf{e}_3 = [0, 0, 1, 0, 0]^\top$. Sau đó, các vector one-hot này được nhân với ma trận embedding — là các tham số được học trong quá trình huấn luyện — để thu được các vector đặc trưng (embedding vectors):

$$\mathbf{p}_u = \mathbf{P}^\top \mathbf{e}_u, \quad \mathbf{q}_i = \mathbf{Q}^\top \mathbf{e}_i$$

(2) Kết hợp (Combination): Quá trình kết hợp giữa một embedding user và một embedding item để trở thành 1 vector (hoặc tensor) duy nhất làm đầu vào cho các bước xử lí tiếp theo. Đối với MF kiểu truyền thống, toán tử kết hợp này là *element-wise product*, trong đó mỗi phần tử tương ứng của hai vector embedding được nhân với nhau để tạo thành vector đặc trưng tổng hợp.

$$z_1 = \phi(p_u, q_i) = p_u \odot q_i$$

Bên cạnh *element-wise product* còn nhiều cách khác để kết hợp 2 vector thành 1 vector hoặc tensor mới, điển hình như phép ghép nối (*concatenate*), phép tích ngoài (*outer product*) hay phép tích Kronecker.

(3) Tổng hợp (Aggregation): Tổng hợp tuyến tính các phần tử của vector z thu được từ bước kết hợp lại để cho ra 1 giá trị vô hướng duy nhất — chính là dự đoán $\hat{y}_{ui}$ cho cặp người dùng u và sản phẩm i.. Ở *linear regression*, bước tổng hợp này bao gồm trọng số và bias học được trong quá trình huấn luyện, nhưng ở MF cơ bản thì tất cả trọng số được cho bằng 1 và không có bias.

$$\hat{y}_{ui} = \mathbf{h}^\top (\mathbf{p}_u \odot \mathbf{q}_i), \quad \mathbf{h} = [1, 1, 1, ..., 1] \text{ nếu là MF truyền thống}$$

(4) Kích hoạt (Activation): Bước này là không bắt buộc, nhưng nếu muốn mở rộng mô hình MF thành một dạng *Perceptron* để đưa vào các mạng neuron học sâu thì có thể gắn thêm hàm kích hoạt vào biểu thức ở bước tổng hợp.

$$\hat{y}_{ui} = a_{out}(\mathbf{h}^T (p_u \odot q_i))$$

(5) Huấn luyện (Training): Với đầu vào đến từ bước 2, có thể đưa dữ liệu này vào để huấn luyện với nhiều loại hàm loss khác nhau tùy thuộc vào mục đích ban đầu là hồi quy, phân loại hay xếp hạng.
Từ đó có thể thấy rằng mô hình Matrix Factorization kiểu cơ bản có nhiều điểm tương đồng với *linear regression*. Dựa trên quan sát này, một hướng mở rộng tự nhiên là bổ sung trọng số riêng và độ dời (bias)

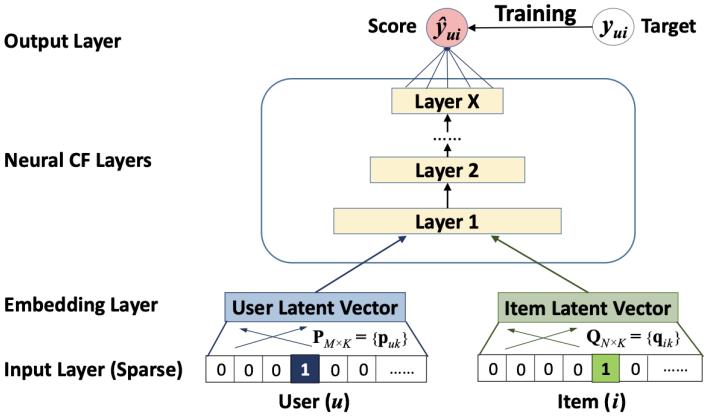
cho từng chiều trong vector đặc trưng ẩn (embedding). Biến thể này được gọi là *Generalized Matrix Factorization* (GMF), trong đó quá trình tổng hợp không còn là phép cộng đơn giản mà trở thành một tổ hợp tuyến tính có tham số:

$$\hat{R} = \mu + \mathbf{b}_u \mathbf{1}_m^\top + \mathbf{1}_n \mathbf{b}_i^\top + \left[\sum_{k=1}^{K} w_k \cdot \mathbf{u}_k \mathbf{v}_k^\top \right]$$

Quay lại với Matrix Factorization. Ở phần trước, chúng ta mới chỉ đề cập đến MF với dữ liệu dạng số thực - explicit feedback dạng rating 1-5 sao hoặc implicit feedback lượng hóa thành giá trị thực. Nhưng trên thực tế, việc thu thập được dữ liệu *explicit feedback* là tương đối khan hiếm. Thay vào đó, các hệ gợi ý chủ yếu phải sử dụng *implicit feedback*. Và cũng vì đặc thù nhiễu cao và không trực tiếp phản ánh mức độ yêu thích của *implicit feedback* mà các hệ gợi ý thường chuyển sang dạng nhị phân: đã tương tác (1) hoặc chưa tương tác (0). Với kiểu phản hồi này, ta cần một hàm mất mát phù hợp hơn MSE, vốn không phản ánh đúng bản chất của bài toán phân loại nhị phân. Nhờ vào framework tổng quát được đề xuất ở trên, có rất nhiều sự lựa chọn thay thế cho hàm mất mát có thể áp dụng:

3.1.2.3 Multi-Layer Perceptron (MLP)

Như đã đề cập ở những phần trước, MF tồn tại nhiều hạn chế khi chỉ học được các quan hệ tuyến tính giữa người dùng và sản phẩm mà chưa xét đến các quan hệ phi tuyến phức tạp. Nhờ vào framework tổng quát đã được đề xuất với một góc nhìn khái quát hơn, ta có thể phát triển MF thành một mô hình Multi-Layer Perceptron (MLP) có khả năng khai thác các đặc trưng ẩn theo hướng này và thực hiện nó bằng mạng neuron.



Hình 3.2: Kiến trúc mô hình của phương pháp MLP

Kiến trúc của mô hình được chia thành nhiều tầng, đầu ra tầng trước là đầu vào của tầng sau.

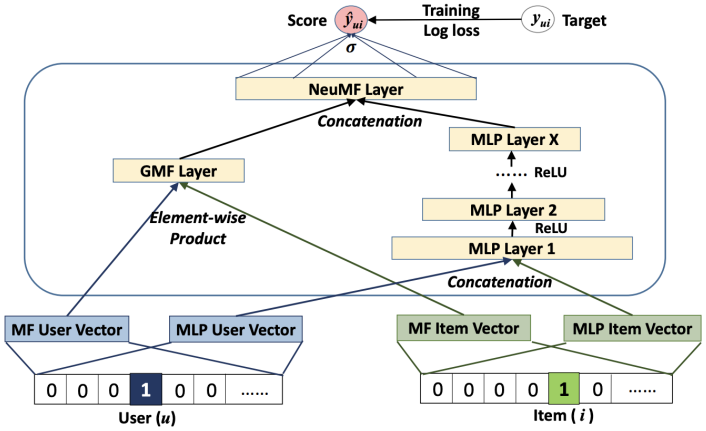
- **Tầng input:** Chúng ta sẽ mã hoá id của user và item biểu diễn user và item dưới dạng one-hot vector.
- **Tầng embedding:** Đây thực chất là một tầng kết nối đầy đủ với đầu vào là vector one-hot của tầng input. Qua tầng này, user và item sẽ được biểu diễn dưới dạng 1 vector dày đặc với số chiều thấp hơn nhiều so với one-hot vector. Ta có thể coi các vector này chính biểu diễn của các thuộc tính ẩn của user và item.
- **Các tầng neuron:** Đây là một mạng nơ-ron kết nối đầy đủ nhiều tầng với đầu vào là một vector - kết quả của phép nối vector nhúng của user với item và đầu ra là một vector.
- **Tầng output:** Với đầu vào vector output của tầng trước, ta sẽ sử dụng một tầng kết nối đầy đủ, thu được đầu ra là tương tác giữa user và item.

Ký hiệu p_u và q_i lần lượt là các vector đầu ra của tầng embedding với đầu vào là user u và item i. Từ đó ta định nghĩa model của phương pháp MLP như sau:

$$
\begin{aligned}
z_1 &= \phi(p_u, q_i) = \begin{bmatrix} p_u \\ q_i \end{bmatrix} \\
\phi_2(z_1) &= a_2(W_2^T z_1 + b_2) \\
&\cdots\cdots \\
\phi_L(z_{L-1}) &= a_L(W_L^T z_{L-1} + b_L) \\
\hat{y}_{ui} &= \sigma(h^T \phi_L(z_{L-1}))
\end{aligned}
$$

trong đó, W_i, b_i và a_i lần lượt là ma trận weight, vector bias và activation function của lớp neuron thứ i; σ là hàm sigmoid.

3.1.3 Neural Matrix Factorization (NeuMF)



Hình 3.3: Kiến trúc mô hình của phương pháp NeuMF

Ta có thể thấy rằng phương pháp GMF cho ta một mô hình tuyến tính, còn MLP cho ta một mô hình phi tuyến. Vậy tại sao ta không thử kết hợp mô hình của hai phương pháp này lại với nhau với hy vọng nó sẽ tận dụng được ưu điểm của các mô hình thành phần? Từ đó, phương pháp NeuMF ra đời như là sự kết hợp của tuyến tính trong GMF và phi tuyến trong MLP.

Ta sẽ chia kiến trúc của mô hình của phương pháp NeuMF thành 2 phần: GMF và MLP.

- **Tầng input và tầng embedding:** Tương tự với tầng input và tầng embedding của MLP. Tuy nhiên, với NeuMF ta sẽ sử dụng các vector embedding riêng cho mỗi phần GMF và MLP.
- **Với GMF:** ta có đầu vào là vector embedding của user và item, sau đó thực hiện phép element-wise product và thu được đầu ra là một vector với kích thước bằng với vector embedding của user và item.
- **Với MLP:** ta thực hiện tương tự theo kiến trúc mạng ở phần trước với đầu ra thu được là một vector.
- **Tầng output:** Ta thực hiện phép nối vector đầu ra của 2 phần lại với nhau thu được một vector, sau đó cho nó đi qua một lớp kết nối đầy đủ thu được đầu ra là dự đoán tương tác của user với item.

Ký hiệu p_u^G, q_i^G và p_u^M, q_i^M lần lượt là các vector đầu ra của tầng embedding với đầu vào là user u và item

i ứng với mỗi phần GMF và MLP. Từ đó ta định nghĩa model của phương pháp NeuMF như sau:

$$\phi^{GMF} = p_u^G \odot q_i^G$$
$$\phi^{MLP} = a_L(W_L^T(a_{L-1}(...a_2(W_2^T \begin{bmatrix} p_u^M \\ q_i^M \end{bmatrix} + b_2)...)) + b_L)$$
$$\hat{y}_{ui} = \sigma(h^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix})$$

trong đó, W_i, b_i và a_i lần lượt là ma trận weight, vector bias và activation function của lớp neuron thứ i; σ là hàm sigmoid.

3.2 Các mô hình CF dựa trên GNN

3.2.1 Tổng quan về GNN (Overview)

Mạng nơ-ron đồ thị (GNN - Graph Neural Network), là một khung tổng quát để định nghĩa mô hình **Mạng nơ-ron sâu (deep neural network)** cho cấu trúc **dữ liệu đồ thị (graph data)**. Ý tưởng cốt lõi của GNN là **tạo ra các biểu diễn ẩn (embedding) của các nút** sao cho chúng thực sự phụ thuộc vào **cấu trúc đồ thị**, cũng như bất kỳ **thông tin đặc trưng (feature information)** nào mà chúng ta có.

Mạng nơ-ron đồ thị (GNN) kết hợp thành tựu của hai lĩnh vực:

- **Mạng nơ-ron tích chập (CNN - Convolutional Neural Network)**, CNN vốn rất hiệu quả trong việc trích xuất đặc trưng (feature extraction) cục bộ trên dữ liệu dạng Euclid như ảnh, văn bản, tuy nhiên với dữ liệu dạng non-Euclid như đồ thị (graph), CNN cần phải được thiết kế một cách tổng quát hơn để xử lý được dữ liệu có kích thước không cố định (non-fixed size). Đó là lý do mà GNN ra đời để khắc phục vấn đề này:
  - **GNN lấy cảm hứng từ phép tích chập (convolution)**: Nếu như CNN dùng một bộ lọc (filter) nhỏ "quét" qua từng vùng cục bộ (local receptive field) và chia sẻ cùng một tập trọng số (weight sharing) thì GNN mở rộng khái niệm này sang dữ liệu dạng đồ thị: Mỗi nút thu thập thông tin từ các nút hàng xóm trong một "vùng lân cận" không định hình, rồi áp dụng một phép biến đổi tuyến tính chung cho tất cả các cạnh. Kết quả là chúng ta có được cơ chế **message passing (lan truyền thông điệp)** với đầy đủ các đặc tính cục bộ, chia sẻ trọng số và bất biến khi đổi trật tự hàng xóm (permutation invariance).
  - **Xây dựng biểu diễn đa tầng (hierarchical representations):** Nếu như kiến trúc CNN bao gồm nhiều tầng như: **Convolution Layer, Pooling Layer, Fully-Connected Layer,...** được xếp chồng lên nhau để học đặc trưng tiềm ẩn (latent feature) thì GNN cũng xếp chồng nhiều lớp lan truyền và tổng hợp thông điệp **(Message Propagation & Aggregation Layer)** để mô hình hiểu sâu hơn về biểu diễn của các nút, cũng như cấu trúc toàn cục của đồ thị.
- **Học biểu diễn đồ thị (GRL - Graph Representation Learning)**, mục tiêu GRL là tổng quát hoá vector biểu diễn ẩn có số chiều thấp (low-demensional embedding vector) cho nút, cạnh,...
  - **Phương pháp Embedding không giám sát (Unsupervised Embedding):** Trước khi GNN ra đời, các thuật toán như **DeepWalk, node2vec** dùng **random walk** kết hợp **Skip-Gram** để học Embedding cho mỗi nút, bảo toàn thứ tự và khoảng cách lân cận trong đồ thị. GNN kế thừa mục tiêu này nhưng cho phép tối ưu trực tiếp biểu diễn (end-to-end) theo downstream task (ví dụ: dự đoán liên kết user–item).
  - **Thiết kế hàm mất mát và sampling thông minh:** GRL đã phát triển các chiến lược lấy mẫu (negative sampling, importance sampling) và loss functions (ví dụ: KL divergence) để học Embedding hiệu quả trên đồ thị lớn. GNN tận dụng những kỹ thuật này để huấn luyện message-passing layer một cách ổn định, giảm nhiễu từ các cạnh không liên quan nhiều.

Ba cấp độ tác vụ chính trong GNN: Trong học sâu trên đồ thị, các bài toán thường được phân loại theo cấp độ của đối tượng dự đoán, gồm ba nhóm chính:

- **Node-Level Tasks:** Dự đoán nhãn hoặc đặc trưng cho từng nút trong đồ thị. Ví dụ phổ biến là ***node classification***, như phân loại người dùng theo hành vi, dự đoán độ tuổi, vai trò trong mạng xã hội,...
- **Edge-Level Tasks:** Dự đoán đặc trưng hoặc xác suất tồn tại của các cạnh giữa hai nút. Một ví dụ tiêu biểu là ***link prediction*** – được sử dụng rộng rãi trong hệ gợi ý, với mục tiêu dự đoán xem liệu user u có khả năng tương tác với item i trong tương lai hay không. Ngoài ra còn có các bài toán như ***edge classification*** (dự đoán loại quan hệ giữa hai node).
- **Graph-Level Tasks:** Dự đoán nhãn hoặc thuộc tính của toàn bộ đồ thị. Ví dụ: Phân loại cấu trúc phân tử trong hóa học (toxic / non-toxic), chẩn đoán bệnh từ đồ thị tín hiệu não,...

3.2.1.1 Vì sao GNN được coi là phương pháp tiên tiến nhất (state-of-the-art)?

Mạng nơ-ron đồ thị (GNN) áp dụng cho bài toán hệ gợi ý là một hướng tiếp cận tương đối mới gần đây so với các phương pháp đã đề cập ở các mục trước và nhanh chóng trở thành hướng tiếp cận ***tiên tiến, hàng đầu (state-of-the-art)***, điều này có thể lý giải ở 03 khía cạnh chính như sau:

- **Dữ liệu cấu trúc (Structured data):** Dữ liệu từ các nền tảng trực tuyến rất đa dạng (tương tác user-item, hồ sơ người dùng, thuộc tính sản phẩm, v.v.). GNN giúp biểu diễn toàn bộ dữ liệu dưới dạng đồ thị với các node và cạnh, từ đó khai thác đầy đủ thông tin và học được embedding chất lượng cao cho người dùng, sản phẩm và các đặc trưng liên quan.
- **Kết nối bậc cao (High-order connectivity):** Các mô hình truyền thống thường chỉ khai thác mối quan hệ bậc 1 giữa user-item, trong khi GNN có khả năng lan truyền và tổng hợp thông tin từ các lân cận nhiều bậc (multi-hop neighbors), từ đó tận dụng hiệu ứng lọc cộng tác một cách đầy đủ và nâng cao hiệu suất của hệ gợi ý.
- **Tín hiệu giám sát (Supervision signal):** GNN có thể xử lý tốt các tình huống thiếu hụt **tín hiệu giám sát (supervised signal)** bằng cách tận dụng các hành vi không mục tiêu (như tìm kiếm, thêm vào giỏ) thông qua **học bán giám sát (self-supervised learning).** Ngoài ra, việc thiết kế các tác vụ phụ trợ trên đồ thị còn giúp tận dụng thêm thông tin và cải thiện khả năng học của mô hình.

3.2.1.2 Quy chuẩn thiết kế của GNN

Khi thiết kế các mạng nơ-ron sâu cho dữ liệu đồ thị, một trong những thách thức cốt lõi là xử lý bản chất phi Euclidean và không có thứ tự tự nhiên của dữ liệu đồ thị, khác biệt hoàn toàn so với dữ liệu lưới truyền thống như hình ảnh hay chuỗi. Theo thuật ngữ toán học (mathematical term), chúng ta cần phải thiết kế một hàm f nhận đầu vào là **ma trận kề A (biểu diễn cấu trúc đồ thị)** và **ma trận đặc trưng nút X (chứa thông tin của các nút)**, sao cho hàm này thỏa mãn hai tính chất cơ bản:

- **Tính bất biến hoán vị (Permutation Invariance):** Đầu ra của hàm hoặc mô hình không thay đổi khi thứ tự các nút đầu vào bị hoán vị. Điều này quan trọng cho ***các tác vụ cấp đồ thị (graph-level tasks).***

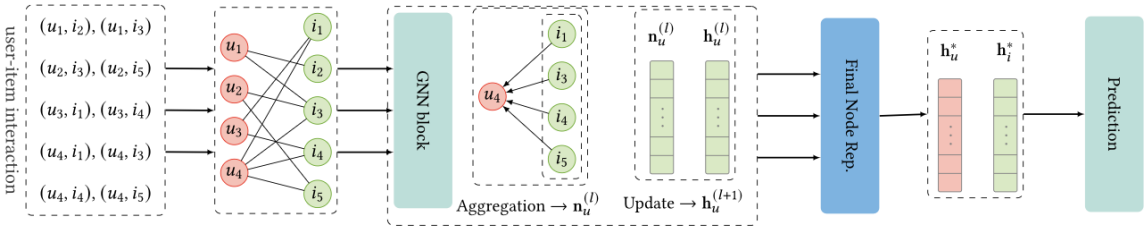
  $$f(A, X) = f(PAP^T, PX)$$

  với P **là ma trận hoán vị (permutation matrix)**. **Tính bất biến hoán vị (permutation invariance)** này là điều kiện cần thiết cho các **phép toán tổng hợp (aggregation functions) thông tin láng giềng** trong GNN. Ví dụ, tổng của các biểu diễn láng giềng không thay đổi bất kể thứ tự mà các láng giềng đó được xem xét.

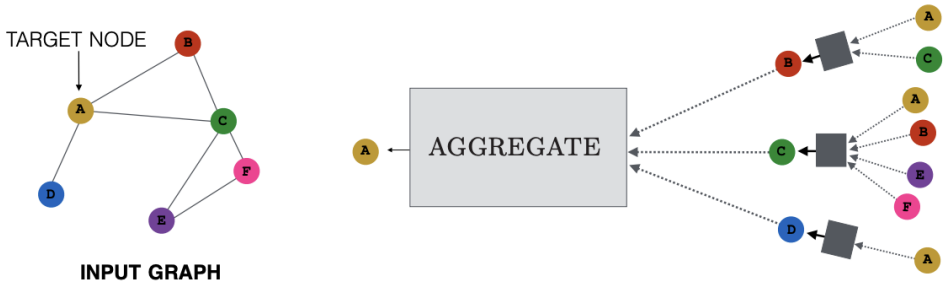
- **Tính đồng biến hoán vị (Permutation Equivariance):** Nếu đầu vào bị hoán vị, đầu ra của hàm hoặc mô hình cũng sẽ bị hoán vị theo cùng cách đó. Điều này cần thiết cho ***các tác vụ cấp nút (node-level tasks) hoặc các tác vụ cấp cạnh (edge-level tasks).***

  $$f(PAP^T, PX) = Pf(A, X)$$

  với P **là ma trận hoán vị (permutation matrix**). **Tính đồng biến hoán vị (permutation equivariance)** này đảm bảo các lớp GNN cập nhật biểu diễn nút một cách nhất quán, dù thứ tự nút đầu vào thay đổi.



Hình 3.4: Framework chung cho các mô hình GNN áp dụng cho bài toán hệ gợi ý dựa trên **lọc cộng tác (CF)**.



Hình 3.5: Mô tả trực quan về cơ chế **lan truyền thông điệp (Message Passing Mechanism)** bên trong **Message Passing Block.** Mô hình sẽ tổng hợp các thông điệp từ các nút láng giềng gần của nút A (cụ thể là B, C và D). Các thông điệp từ những nút láng giềng này lại được tạo ra dựa trên thông tin được tổng hợp từ các nút lân cận của chính chúng, và quá trình này tiếp tục như vậy.

3.2.1.3 GNN Framework cho bài toán gợi ý dựa trên lọc cộng tác (CF)

Quy trình ở hình vẽ 3.4 có thể hiểu đơn giản như sau: Đầu vào của mô hình là cấu trúc **đồ thị hai phía user-item (user-item bipartite graph)**, sau khi đưa vào khối **Message Passing Block** thì **mô hình sẽ học các biểu diễn ẩn (embedding) của các nút (user, item) trên đồ thị**, đầu ra của mô hình là **biểu diễn ẩn cuối cùng (final representation)** của các nút phục vụ cho việc dự đoán. Chi tiết về quy trình như sau:

- **Xây dựng đồ thị (Graph Construction):** Khác với các mô hình gợi ý truyền thống – nơi dữ liệu đầu vào thường được biểu diễn dưới dạng bảng hoặc **ma trận tương tác người dùng – sản phẩm (user-item interaction matrix)**, các mô hình GNN cho bài toán gợi ý dựa trên lọc cộng tác **(Collaborative Filtering – CF)** yêu cầu biểu diễn dữ liệu dưới dạng ***đồ thị hai phía* (bipartite graph).** Cụ thể, đồ thị được định nghĩa là $G = (U \cup V, E)$, trong đó:
  - U là tập các nút người dùng (users), V là tập các nút sản phẩm (items). Trong đồ thị hai phía, không có cạnh nào giữa hai nút trong tập người dùng U hoặc hai nút trong tập sản phẩm V.
  - $E \subseteq U \times V$ là tập các cạnh, mỗi cạnh $(u, v) \in E$ biểu thị một tương tác giữa người dùng u và item v (ví dụ: click, mua hàng, đánh giá, xem...).

  Cấu trúc đồ thị này là nền tảng để mô hình GNN khai thác **tín hiệu cộng tác (collaborative signal)** thông qua các bước lan truyền thông tin giữa các node lân cận.
- **Khối lan truyền thông điệp (Message Passing Block):** Sau khi xây dựng đồ thị hai phía user-item, dữ liệu này được đưa vào **khối lan truyền thông điệp (message passing block)** nhằm học biểu diễn ẩn (embedding) cho từng node thông qua **các lớp lan truyền thông điệp (message passing layers)**. Mỗi lớp bao gồm ba bước chính:
  - **Xây dựng thông điệp (Message Construction):** Tạo thông điệp từ node lân cận, thường dựa trên embedding hiện tại và trọng số cạnh (nếu có).
  - **Tổng hợp thông điệp (Message Aggregation):** Tổng hợp thông điệp từ các node lân cận (sử dụng tổng, trung bình hoặc attention).

  - **Cập nhật thông tin (Information Update):** Cập nhật embedding mới của node trung tâm dựa trên embedding hiện tại và thông điệp tổng hợp.

  Hình 3.5 là hình ảnh trực quan hoá về cơ chế lan truyền thông điệp thực hiện, quá trình này lặp lại qua nhiều tầng cho phép nút đích (target node) thu nhận thông tin từ các lân cận bậc cao hơn (multi-hop). Các mô hình GNN khác nhau như **GraphSAGE, GAT, GCN,...** sẽ có cách thiết kế các thao tác **Message Construction, Message Aggregattion, Information Update** khác nhau.

- **Biểu diễn cuối cùng của nút (Final Node Representation):** Sau L lớp lan truyền, embedding cuối cùng của mỗi node (người dùng hoặc sản phẩm) sẽ được sử dụng trong các tác vụ downstream như tính điểm gợi ý, tính độ tương đồng, hoặc xếp hạng sản phẩm. Tùy theo mô hình, biểu diễn này có thể là embedding từ lớp cuối hoặc tổ hợp từ nhiều lớp (concatenation/sum).

Trong báo cáo này, tôi chỉ tập chung áp dụng các mô hình GNN dựa trên thiết kế của **Mạng tích chập đồ thị (GCN - Graph Neural Network)**. Phần tiếp theo sẽ trình bày tổng quan về GCN và mối quan hệ của GCN với các biến thể của nó được thiết kế dành riêng cho bài toán hệ gợi ý dựa trên ***lọc cộng tác (CF - Colaborative Filtering).***

3.2.2 Mối liên hệ giữa GCN và các mô hình hệ gợi ý

Graph Convolutional Network (GCN) được giới thiệu lần đầu bởi *Kipf và Welling (2016) [14]* như một phương pháp tổng quát hóa *phép tích chập (convolution)* cho dữ liệu đồ thị phi cấu trúc, nhằm giải quyết **bài toán phân loại nút (Node Classification)** trên **đồ thị đồng nhất (Homogeneous Graph)**.

Công thức tổng quát của GCN: GCN học biểu diễn (embedding) cho mỗi nút thông qua cơ chế lan truyền theo tầng (layer) như đã đề cập. Công thức cập nhật embedding cho nút v tại tầng $l+1$ được định nghĩa như sau:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\,\mathbf{H}^{(l)}\,\mathbf{W}^{(l)}\right)$$

Trong đó:

- $\mathbf{A} \in \mathbb{R}^{n\times n}$: ma trận kề của đồ thị (adjacency matrix), n là số node.
- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$: ma trận kề có thêm self-loop.
- $\tilde{\mathbf{D}}$: ma trận bậc tương ứng với $\tilde{\mathbf{A}}$, tức $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.
- $\mathbf{H}^{(l)} \in \mathbb{R}^{n\times d_l}$: ma trận embedding tại tầng l, với d_l là số chiều embedding.
- $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l\times d_{l+1}}$: trọng số học được của tầng l.
- $\sigma(\cdot)$: hàm kích hoạt (thường dùng ReLU).

Giải thích hệ số chuẩn hoá đối xứng: Trong công thức GCN gốc, ta dùng ma trận $\tilde{A} = A + I$ (có self–loop) và $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Phép nhân $\tilde{D}^{-\frac{1}{2}}\,\tilde{A}\,\tilde{D}^{-\frac{1}{2}}$ gọi là *normalized adjacency*, hay *symmetric normalization*.

- **Nguồn gốc từ spectral graph theory:** Normalized Laplacian định nghĩa là $L = I - D^{-\frac{1}{2}}A\,D^{-\frac{1}{2}}$. *Kipf và Welling* lấy xấp xỉ bậc nhất của bộ lọc phổ trên đồ thị (ChebNet) dẫn đến phép nhân $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$.
- **Ổn định giá trị eigen:** Tất cả eigenvalue của $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ nằm trong khoảng $[-1, 1]$, giúp tránh vanishing/exploding gradient khi xếp chồng nhiều tầng GCN.
- **Bất biến theo hoán vị (Permutation Invariance):** Chuẩn hoá đối xứng bảo đảm mỗi cạnh (u, v) đóng góp đồng nhất, không phụ thuộc thứ tự node.
- **Ổn định biên độ thông điệp:** Node có degree quá lớn hoặc quá nhỏ sẽ không chi phối quá mức tổng thông điệp, giúp embedding cân bằng giữa các node.

GCN là một trong những kiến trúc GNN đầu tiên mang tính tổng quát, với cơ chế **lan truyền thông điệp (message passing)** và **tổng hợp lân cận (neighborhood aggregation)** đơn giản nhưng hiệu quả. Tại mỗi tầng, biểu diễn (embedding) của một nút được cập nhật bằng cách tổng hợp biểu diễn của các nút láng giềng gần, sau đó áp dụng một hàm kích hoạt phi tuyến. Việc chia sẻ trọng số và bất biến hoán vị (permutation invariance) giúp mô hình học được đặc trưng từ cấu trúc đồ thị một cách hiệu quả, đồng thời tránh overfitting nhờ số lượng tham số nhỏ.

Chính những ưu điểm về tính ổn định, khả năng lan truyền cục bộ và bất biến hoán vị đã giúp GCN nhanh chóng trở thành nền tảng cho rất nhiều biến thể trong đa dạng các lĩnh vực, trong đó nổi bật nhất là bài toán hệ gợi ý dựa trên **lọc cộng tác (Collaborative Filtering – CF)**. Tuy nhiên, GCN gốc chưa được tối ưu cho bài toán này, do đó nhiều mô hình GCN-based CF ra đời như các mô hình: NGCF, LightGCN, UltraGCN,...để tận dụng tối ưu hơn GCN. Các phần sau của báo cáo trình bày sẽ trình bày cụ thể về các biến thể này.
Công thức tổng quát GCN-based CF

Với mỗi nút u bất kỳ ở tầng l, ta định nghĩa công thức chung cho 03 thao tác chính trong cơ chế ***lan truyền thông điệp (message passing)*** như sau:

$$\textbf{MESSAGE:} \quad \mathbf{m}_{u\leftarrow i}^{(l)} = \mathrm{MSG}^{(l)}\big(\mathbf{h}_u^{(l)}, \mathbf{h}_i^{(l)}, e_{ui}\big) \tag{3.1}$$

$$\textbf{AGGREGATE:} \quad \mathbf{n}_u^{(l)} = \mathrm{AGG}^{(l)}\Big(\big\{\mathbf{m}_{u\leftarrow i}^{(l)} \mid i \in \mathcal{N}_u\big\}\Big) \tag{3.2}$$

$$\textbf{UPDATE:} \quad \mathbf{h}_u^{(l+1)} = \mathrm{UPD}^{(l)}\big(\mathbf{h}_u^{(l)}, \mathbf{n}_u^{(l)}\big) \tag{3.3}$$

Trong đó:

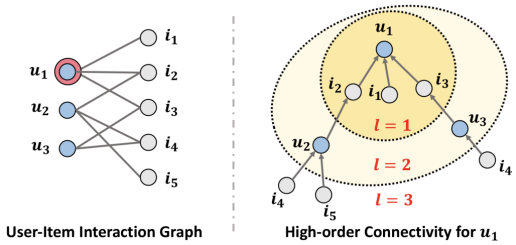
- $\mathbf{h}_u^{(l)} \in \mathbb{R}^d$ là biểu diễn ẩn (embedding) của node u tại layer l.
- $\mathbf{h}_i^{(l)} \in \mathbb{R}^d$ là biểu diễn ẩn (embedding) của node i tại layer l.
- $\mathcal{N}_u = \{i \mid (u,i) \in E\}$ là tập láng giềng của u.
- e_{ui} là trọng số hoặc đặc trưng cạnh (u,i) (nếu có).
- $\mathbf{m}_{u\leftarrow i}^{(l)}$ là thông điệp (message) được xây dựng từ láng giềng $i \to u$
- $\mathbf{n}_u^{(l)}$ là embedding đã được tổng hợp (aggregated representation) của các message đến u.
- $\mathrm{MSG}^{(l)}, \mathrm{AGG}^{(l)}, \mathrm{UPD}^{(l)}$ lần lượt là các hàm **Message Construction, Message Aggregation và Information Update** ở layer l. Các hàm này sẽ được tuỳ chỉnh theo một thiết kế của một mô hình cụ thể.

Ở các phần sau khi đi chi tiết vào cách triển khai của thể 03 thao tác trên của từng mô hình, để diễn giải một cách mạch lạc hơn theo hình vẽ kiến trúc mô hình, tôi sẽ dùng cả các ký hiệu khác nhau để trình bày về biểu diễn ẩn (embedding) của một nút v bất kỳ ở tầng thứ l, hai kí hiệu $\mathbf{h}_v^l$ và $\mathbf{e}_v^l$ có ý nghĩa tương tự nhau.

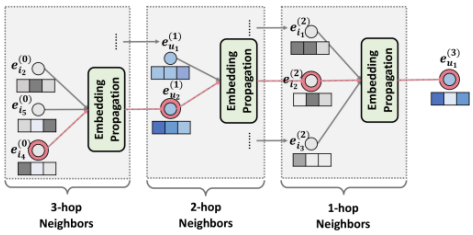
3.2.3 Neural Graph Collaborative Filtering (NGCF)

NGCF (Neural Graph Collaborative Filtering) là mô hình tiên phong kế thừa kiến trúc GCN cho bài toán hệ gợi ý dựa trên **lọc cộng tác (CF)**, kiến trúc của NGCF thể hiện ở hình vẽ 3.8 bao gồm 03 thành phần chính:

- **Lớp Embedding (Embedding Layer):** Lớp này đóng vai trò khởi tạo embedding ban đầu cho tất cả các nút (user, item).
- **Các lớp lan truyền embedding (Embedding Propagation Layers):** Các lớp này đóng vai trò tinh chỉnh (huấn luyện) embedding bằng cơ chế lan truyền thông điệp (message passing).
- **Lớp dự đoán (Prediction Layer):** Lớp này đóng vai trò kết hợp các embedding từ nhiều lớp lan truyền embedding khác nhau, và đưa ra dự đoán (predict) về độ tương đồng của user u và item i



Hình 3.6: Minh hoạ cách mà nút u_1 (nút người dùng mục tiêu (target node) cần cung cấp gợi ý, có thể khai thác thông tin kết nối bậc cao (high-order connectivity) để cập nhật biểu diễn ẩn (embedding) của chính nó.



Hình 3.7: Minh họa quá trình lan truyền nhúng (embedding propagation) bậc ba cho nút người dùng mục tiêu u_1.

Lớp Embedding (Embedding Layer)

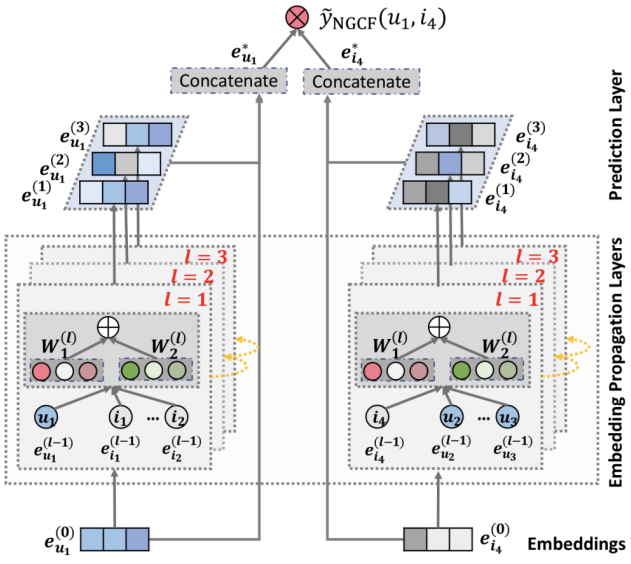
Mỗi user u và mỗi item i được khởi tạo bằng một véc-tơ nhúng (embedding) có kích thước d: $\mathbf{e}_u \in \mathbb{R}^d$, $\mathbf{e}_i \in \mathbb{R}^d$. Tất cả các embedding được lưu trong một bảng tra cứu (lookup table) $\mathbf{E} \in \mathbb{R}^{(N+M)\times d}$:

$$\mathbf{E} = \big[\underbrace{\mathbf{e}_{u_1}, \mathbf{e}_{u_2}, \ldots, \mathbf{e}_{u_N}}_{\text{user embeddings}} \;\big|\; \underbrace{\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \ldots, \mathbf{e}_{i_M}}_{\text{item embeddings}}\big].$$

Bảng tra cứu (lookup table) sẽ được tinh chỉnh dần dần qua các **Lớp lan truyền embedding (Embedding Propagation Layer)**.

Các lớp lan truyền embedding (Embedding Propagation Layers)

Đây là thành phần quan trọng nhất của **NGCF.** Thiết kế của các lớp này kế thừa và mở rộng kiến trúc của **GCN** ở 03 khía cạnh chính, thể hiện ở 03 thao tác **MESSAGE, AGGREGATE, UPDATE** như đã đề cập bên trên:



Hình 3.8: Minh họa kiến trúc tổng thể của mô hình **NGCF** (Các đường thông tin biểu thị luồng thông tin). Biểu diễn ẩn (embedding) của user u1 (trái) và item i4 (phải) được tinh chỉnh qua nhiều **Lớp lan truyền nhúng (embedding propagation layer)**, đầu ra của chúng được nối lại để đưa ra dự đoán cuối cùng.

- **NGCF** kế thừa thành phần *biến đổi tuyến tính* $\mathbf{W}_1^{(l)}\mathbf{h}_u^{(l)}$ trong **GCN**, đồng thời kết hợp *thêm thành phần tương tác phi tuyến* giữa user và item thông qua phép nhân Hadamard ($\mathbf{h}_u^l \odot \mathbf{h}_v^l$). Điều

này giúp mô hình hóa tốt hơn sự ảnh hưởng hai chiều trong tương tác giữa user và item.

$$\textbf{MESSAGE}: \quad \mathbf{m}_{u\leftarrow i}^{(l)} = \mathbf{W}_1^{(l)}\,\mathbf{h}_u^{(l)} + \mathbf{W}_2^{(l)}\left(\mathbf{h}_u^{(l)} \odot \mathbf{h}_i^{(l)}\right)$$

Trong đó: $\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}$: Ma trận trọng số có thể huấn luyện ở lớp l, với kích thước $\mathbb{R}^{d_l \times d_{l-1}}$, trong đó d_l là kích thước embedding ở lớp l.

- $\mathbf{W}_1^{(l)}$: *Biến đổi tuyến tính* embedding của nút nguồn $\mathbf{h}_u^{(l)}$.
- $\mathbf{W}_2^{(l)}$: Biến đổi phép nhân Hadamard $\mathbf{h}_u^{(l)} \odot \mathbf{h}_v^{(l)}$, giúp nắm bắt các *tương tác phi tuyến* giữa embedding của người dùng và mặt hàng.

- **NGCF** kế thừa GCN, vẫn dùng **tổng chuẩn hoá đối xứng (symmetrical normalization sum)** để tổng hợp thông điệp từ các láng giềng.

$$\textbf{AGGREGATE}: \quad \mathbf{n}_u^{(l)} = \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\,\mathbf{m}_{u\leftarrow i}^{(l)}$$

Trong đó **hệ số chuẩn hoá đối xứng** $\frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}$, đơn giản là giúp ổn định về việc xây dựng thông điệp, tránh tình huống những nút có bậc (degree) cao (nhiều láng giáng) "áp đảo" việc xây dựng thông điệp chung.

- **NGCF** kế thừa GCN, vẫn dùng hàm kích hoạt phi tuyến $\sigma(\cdot)$ để nâng cao khả năng biểu diễn. **Hàm kích hoạt (activation function)** thường dùng trong mô hình **NGCF** là **ReLU** hoặc **LeakyReLU**.

$$\textbf{UPDATE}: \quad \mathbf{h}_u^{(l+1)} = \sigma\big(\mathbf{n}_u^{(l)}\big)$$

Lớp dự đoán (Prediction Layer): Lớp này thực hiên hai thao tác chính sau đây:

- **Biểu diễn cuối cùng (Final Representation):** Embedding cuối cùng của mỗi nút (node) được tạo bằng cách nối tất cả các embedding từ các lớp:

$$\mathbf{e}_u^* = \mathbf{e}_u^{(1)} \,\|\, \mathbf{e}_u^{(2)} \,\|\, ... \,\|\, \mathbf{e}_u^{(l)}$$

$$\mathbf{e}_i^* = \mathbf{e}_i^{(1)} \,\|\, \mathbf{e}_i^{(2)} \,\|\, ... \,\|\, \mathbf{e}_i^{(l)}$$

- **Đưa ra dự đoán (Prediction)**: Dự đoán độ yêu thích giữa user u và item i được tính bằng tích vô hướng giữa hai embedding cuối:

$$\hat{y}(u,i) = {\mathbf{e}_u^*}^\top \cdot \mathbf{e}_i^*$$

Quy tắc lan truyền dưới dạng ma trận (Matrix–Form Propagation Rule): Toàn bộ cơ chế lan truyền thông điệp qua l lớp, với 03 thao tác chính ở mỗi tầng có thể được viết lại dưới dạng một công thức ma trận duy nhất:

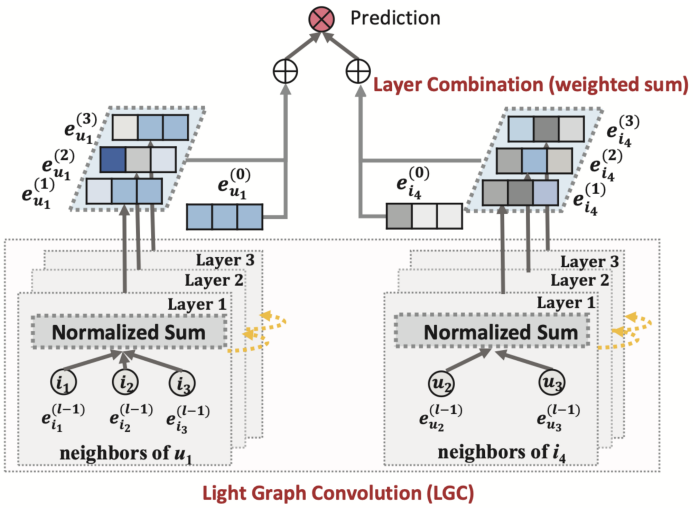
$$\mathbf{E}^{(l)} = \text{LeakyReLU}\Big(\big(\mathcal{L} + \mathbf{I}\big)\,\mathbf{E}^{(l-1)}\,\mathbf{W}_1^{(l)} \;+\; \mathcal{L}\,\mathbf{E}^{(l-1)} \;\odot\; \mathbf{E}^{(l-1)}\,\mathbf{W}_2^{(l)}\Big),$$

trong đó:

- $\mathbf{E}^{(l)} \in \mathbb{R}^{(N+M)\times d_l}$ là ma trận embedding của tất cả users và items sau l bước lan truyền.
- $\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)} \in \mathbb{R}^{d_{l-1}\times d_l}$ là các ma trận trọng số (tuyến tính và phi tuyến) ở tầng l.
- $\mathcal{L} = \mathbf{D}^{-\frac{1}{2}}\,\mathbf{A}\,\mathbf{D}^{-\frac{1}{2}}$ là normalized adjacency matrix của đồ thị hai phía user-item, với

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{R}^\top & \mathbf{0} \end{bmatrix}, \quad \mathbf{D}_{ii} = \sum_j A_{ij}$$

- $\mathbf{I}$ là ma trận đơn vị, tương đương self–loop.
- $\odot$ là phép nhân Hadamard giữa hai ma trận cùng kích thước, biểu diễn tương tác phi tuyến giữa embedding user và item.
- LeakyReLU$(\cdot)$ là hàm kích hoạt phi tuyến, giúp NGCF học được các biểu diễn phức tạp hơn.



Hình 3.9: Minh hoạ kiến trúc tổng thể của mô hình **LightGCN.** Trong **LGC của LightGCN (tương ứng với Embedding Propagation Layer của NGCF)**, chỉ phép tính **tổng chuẩn hóa (normalized sum)** của các embedding láng giềng được thực hiện để truyền tới lớp tiếp theo, điều này làm đơn giản hóa đáng kể các lớp tích chập GCN. Trong phần **Kết hợp lớp (Layer Combination)**, các embedding từ mỗi lớp để tổng hợp để thu được biểu diễn cuối cùng.

3.2.4 Light Graph Convolutional Network (LightGCN)

LightGCN (Light Graph Convolutional Network) là phiên bản nhẹ, gọn và đơn giản hơn rất nhiều so với mô hình NGCF truyền thống, LightGCN cải tiến NGCF bằng cách chỉ giữ lại đúng thành phần ***neighborhood aggregation***, trong khi **loại bỏ hoàn toàn tự kết nối (self-loops), biến đổi đặc trưng tuyến tính (feature transformation), biến đổi phi tuyến (non-linear activation).**

Do bản chất LightGCN là cải tiến của NGCF, hình vẽ 3.9 thể hiện kiến trúc của LightGCN vẫn gồm 03 thành phần chính giống NGCF, Paper của Xlanghe dùng tên gọi khác cho các Lớp lan truyền Embedding là **Light Graph Convolution (LGC)**, nhưng bản chất chúng là tương tự nhau.

Do đó ở phần này, tôi sẽ chỉ đưa ra sự khác biệt của LightGCN, và sự khác biệt lớn nhất nằm ở 03 thao tác chính: **MESSAGE, AGGREGATE, UPDATE** trong cơ chế l**an truyền thông điệp (message passing mechanism)** ở các **tầng lan truyền thông điệp (message passing layer**)

- **Loại bỏ kết nối vòng (self-loops)**, tức là mỗi bước **xây dựng thông điệp (message construction)**, mỗi nút trên đồ thị chỉ nhận thông điệp từ các nút lân cận để phục vụ cho việc cập nhật vector biểu diễn ẩn của chính mình.

$$\textbf{MESSAGE}: \quad \mathbf{m}_{u\leftarrow i}^{(l)} = \mathbf{h}_i^{(l)}$$

- **Loại bỏ biến đổi đặc trưng (feature transformation)**, tức là ở mối bước xây dựng thông điệp không đi kèm với phép nhân ma trận đặc trưng.

$$\textbf{AGGREGATE}: \quad \mathbf{n}_u^{(l)} = \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|\,|\mathcal{N}_i|}} \mathbf{m}_{u\leftarrow i}^{(l)}$$

- **Loại bỏ biến đổi phi tuyến (non-linear activation)**, tức là sau khi hoàn thành thu thập thông điệp (message aggregation), thì thông điệp này chính là biểu diễn ẩn (embedding) của nút mục tiêu (target node).

$$\textbf{UPDATE}: \quad \mathbf{h}_u^{(l+1)} = \mathbf{n}_u^{(l)}$$

Ba thao tác **MESSAGE, AGGREGATE, UPDATE** bên trên có thể gộp lại thành một công thức duy nhất, ở mỗi lớp lan truyền thông điệp như sau:

$$\mathbf{h}_u^{(l+1)} = \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|\,|\mathcal{N}_i|}} \mathbf{h}_i^{(l)}$$

Final Representation: Sự khác biệt thứ hai của LightGCN so với NGCF nằm ở **lớp dự đoán (Prediction Layer)**, cách kết hợp để để thu được **biểu diễn cuối cùng (final representation)** trong LightGCN là sử dụng **phép tổng có trọng số (weighted sum)** embedding học được từ Llớp lan truyền:

$$\mathbf{h}_u^* = \sum_{l=0}^{L} \alpha^{(l)}\, \mathbf{h}_u^{(l)}, \quad \alpha_l = \frac{1}{L+1},$$

Prediction: Sau khi có được **biểu diễn cuối cùng (final embedding)**, mô hình đưa ra dự đoán user u và item i bằng phép tính tích vô hướng, tương tự giống như NGCF:

$$\hat{y}(u,i) = \mathbf{e}_u^{*\top} \mathbf{e}_i^*.$$

Matrix-Form Propagation Rule in LightGCN: Embedding đầu ra cuối cùng được tính bằng trung bình có trọng số của các embedding từ tất cả các tầng lan truyền:

$$\mathbf{E}^* = \alpha_0\, \mathbf{E}^{(0)} + \alpha_1\, \mathbf{E}^{(1)} + \cdots + \alpha_L\, \mathbf{E}^{(L)} = \sum_{k=0}^{L} \alpha_l\, \widetilde{\mathbf{A}}^L\, \mathbf{E}^{(0)},$$

trong đó:

- $\mathbf{E}^{(0)} \in \mathbb{R}^{(N+M)\times d_0}$ là bảng embedding (look-up table) khởi tạo ban đầu.
- $\mathbf{E}^{(l)} \in \mathbb{R}^{(N+M)\times d_l}$ là ma trận embedding của tất cả users và items sau l bước lan truyền.
- $\widetilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ là ma trận kề được chuẩn hoá đối xứng.
- α_l là trọng số kết hợp (thường lấy $\alpha_l = \frac{1}{L+1}$).

Chương 4

Các chiến lược đánh giá và độ đo áp dụng

4.1 Các độ đo đánh giá hệ gợi ý

Trong các hệ thống gợi ý, các độ đo đánh giá mô hình thường tập trung vào nhóm các sản phẩm nằm trong **top-k kết quả được đề xuất** cho mỗi người dùng. Giá trị của k sẽ thay đổi tùy thuộc vào yêu cầu cụ thể của từng nền tảng ứng dụng, tuy nhiên trong các nghiên cứu học thuật, các giá trị phổ biến thường là $k = 10$ hoặc $k = 20$.

Để đánh giá mô hình trong top-k, mỗi sản phẩm trong danh sách gợi ý sẽ được gán nhãn nhị phân:

- Một item được xem là có liên quan (*relevant*, nhãn 1) nếu người dùng đã thực sự tương tác tích cực với item đó (ví dụ: mua, xem, đánh giá cao...).
- Ngược lại, item được xem là không liên quan (*irrelevant*, nhãn 0) nếu người dùng chưa từng tương tác, hoặc có hành vi phản hồi tiêu cực đối với sản phẩm đó.

Trong báo cáo này, tôi sử dụng 4 độ đo phổ biến nhất để đánh giá hiệu quả của các mô hình hệ gợi ý, bao gồm: **Precision@k, Recall@k, HitRate@k, và NDCG@k.** Các độ đo này đều được tính trên tập top 10 item mà mô hình gợi ý cho mỗi người dùng, phản ánh khả năng đưa ra những gợi ý chính xác và đúng thứ tự ưu tiên.

4.1.1 Precision at k (Precision@K)

Độ đo tính chính xác của đề xuất, xác định tỉ lệ phần *relevant* được đề xuất trên tổng số item được đề xuất

$$\text{Precision@k} = \frac{TP}{TP + FP} = \frac{|\text{relevant items recommended}|}{|k|}$$

Ý nghĩa: Precision cao cho thấy ít đề xuất "rác" (false positives).

4.1.2 Recall at k (Recall@K)

Thước đo tính đầy đủ, xác định tỉ lệ phần *relevant* được đề xuất trên tổng số item *relevant* trên toàn bộ dataset

$$\text{Recall@k} = \frac{TP}{TP + FN} = \frac{|\text{relevant items recommended}|}{|\text{all relevant items}|}$$

Ý nghĩa: Recall cao cho thấy mô hình không bỏ sót nhiều item quan trọng (false negatives).

4.1.3 NDCG at k (NDCG@K)

Normalized Discounted Cumulative Gain — thước đo độ hữu dụng của đề xuất giảm dần theo vị trí:

$$\text{DCG@k} = \sum_{i=1}^{k} \frac{2^{r_i} - 1}{\log_2(i+1)}, \quad \text{NDCG@k} = \frac{\text{DCG@k}}{\text{IDCG@k}},$$

trong đó $r_i = 1$ nếu item thứ i trong đề xuất đúng, ngược lại 0; IDCG là DCG lý tưởng (khi sắp xếp tất cả item đúng lên đầu). *Ý nghĩa:* NDCG cao khi các item đúng được xếp ở thứ hạng cao.

4.1.4 HitRate at k (HitRate@K)

Kiểm tra xem trong đề xuất có ít nhất một item đúng hay không:

$$\text{HitRate@k} = \begin{cases} 1, & |\text{relevant items recommended}| \geq 1, \\ 0, & \text{ngược lại.} \end{cases}$$

Ý nghĩa: Một độ đo nhẹ hơn Precision và Recall, thích hợp khi dataset có độ thưa rất lớn

4.2 Các chiến lược đánh giá

Dữ liệu trong bộ Steam Games chứa yếu tố thời gian thông qua các cột ***date*** và ***date_released***. Đây là yếu tố mang tính quyết định trong việc đánh giá mô hình một cách khách quan và thực tế. Nếu chia dữ liệu một cách ngẫu nhiên như các chiến lược phổ biến hiện nay (Hold-out, Cross-validation, Bootstrap Sampling), sẽ tiềm ẩn nguy cơ xảy ra hiện tượng rò rỉ dữ liệu thời gian (time leakage).

Time leakage là hiện tượng xảy ra khi thông tin từ tương lai (so với thời điểm dự đoán) vô tình xuất hiện trong tập huấn luyện. Điều này khiến mô hình học được các mẫu có liên hệ chặt chẽ với các sự kiện tương lai mà đáng lý ra không thể biết tại thời điểm dự đoán. Hậu quả là mô hình đạt hiệu suất rất cao trong giai đoạn đánh giá, nhưng lại không có khả năng tổng quát hóa khi triển khai trên dữ liệu thực tế – dẫn đến đánh giá sai lệch và tiềm ẩn rủi ro nghiêm trọng khi ứng dụng.

Lấy ví dụ cụ thể trong lĩnh vực hệ thống gợi ý, với phương pháp cơ bản là gợi ý theo độ phổ biến (popularity-based recommendation). Giả sử ta muốn dự đoán sản phẩm phù hợp cho người dùng tại thời điểm đầu tháng 4. Một item cụ thể trở nên cực kỳ phổ biến vào tháng 5 nhờ vào một xu hướng trên mạng xã hội. Nếu một phần dữ liệu tháng 5 bị trộn lẫn vào tập huấn luyện, mô hình sẽ học được rằng item đó có độ phổ biến cao và gợi ý cho người dùng, mặc dù tại thời điểm tháng 4 item này vẫn chưa có sức hút. Điều này cho thấy mô hình đã vô tình sử dụng thông tin từ tương lai để đưa ra quyết định – một vi phạm nghiêm trọng nguyên tắc nhân quả trong học máy.

Do đó, khi xử lý các bộ dữ liệu có yếu tố thời gian, bắt buộc phải chia tập dữ liệu theo trục thời gian nhằm đảm bảo tính khách quan, đúng ngữ cảnh và tránh hoàn toàn **hiện tượng time leakage.** Trong lĩnh vực hệ thống gợi ý, hai chiến lược đánh giá phổ biến tuân theo nguyên tắc này là **Full-corpus evaluation** và **Leave-one-last evaluation**, sẽ được trình bày chi tiết ở phần tiếp theo.

4.2.1 Full-corpus Evaluation

Một chiến lược đánh giá mô hình vừa tự nhiên về mặt ý tưởng, vừa gần sát thực tế triển khai là phương pháp **Full-corpus Evaluation**, còn được gọi là **Temporal Hold-out Split**. Trong phương pháp này, một mốc thời gian cố định được chọn để phân chia dữ liệu cho toàn bộ người dùng. Cụ thể, tất cả các tương tác xảy ra trước thời điểm này sẽ được sử dụng để huấn luyện mô hình, còn các tương tác sau thời điểm đó sẽ dùng để đánh giá.

Khi thực hiện đánh giá, mô hình sẽ đưa ra gợi ý cho mỗi người dùng bằng cách xếp hạng toàn bộ các item có trong tập dữ liệu, ngoại trừ những item mà người dùng đó đã tương tác trước đó.

- **Ưu điểm**: Phương pháp này phản ánh trung thực môi trường triển khai thực tế, nơi mà tại thời điểm dự đoán, hệ thống chỉ có thể tiếp cận thông tin từ quá khứ và cần đưa ra dự đoán cho tương lai. Do đó, nó đảm bảo không xảy ra rò rỉ thông tin theo thời gian và mô phỏng đúng quy trình sử dụng trong các hệ thống gợi ý thời gian thực.
- **Hạn chế**: Tuy nhiên, chiến lược này tồn tại hai nhược điểm chính khiến nó không phổ biến bằng phương pháp *Leave-one-last*:

- Thứ nhất, việc đánh giá dựa trên toàn bộ tập item trong mỗi lần gợi ý khiến chi phí tính toán trở nên rất lớn, đặc biệt khi dataset có số lượng item lớn.
- Thứ hai, phương pháp này thường làm gia tăng các trường hợp *cold-start* (user hoặc item chỉ mới xuất hiện trong tập test). Dù đây là hiện tượng phổ biến trong thực tế, nhưng nó yêu cầu các mô hình phải có chiến lược xử lý cold-start kèm theo – điều mà các phương pháp collaborative filtering thuần túy thường không hỗ trợ. Khi số lượng cold-start lớn, hiệu suất gợi ý của mô hình có thể không phản ánh đúng năng lực tổng thể mà bị ảnh hưởng nhiều bởi chiến lược xử lí kèm theo.

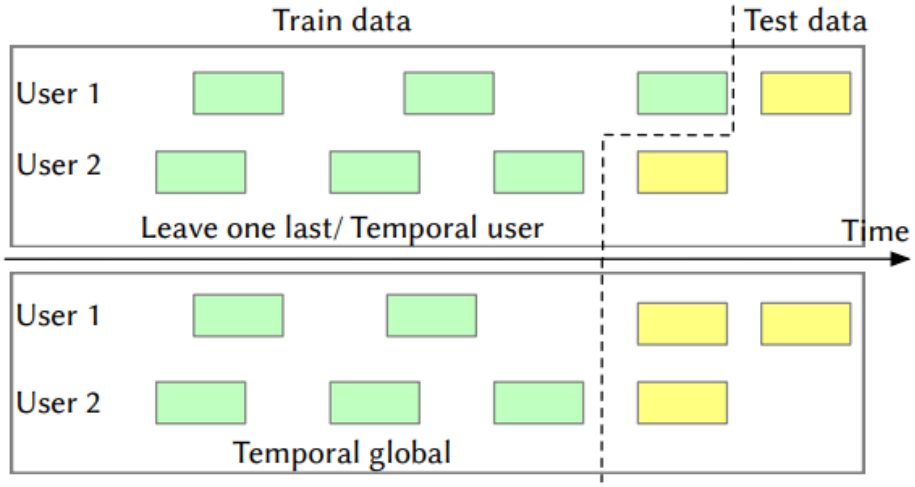
Tuy nhiên, với việc tập game trong bộ dữ liệu Steam Games có quy mô vừa phải (khoảng 4,600 trò chơi), nhược điểm liên quan đến chi phí tính toán của phương pháp Full-corpus Evaluation là không quá nghiêm trọng. Tổng số lượng item đủ nhỏ để việc đánh giá toàn tập vẫn khả thi về mặt tài nguyên và thời gian xử lý. Do đó, chúng tôi hoàn toàn có thể áp dụng chiến lược đánh giá này nhằm đảm bảo tính khách quan, sát thực tế triển khai, đồng thời phản ánh đúng năng lực tổng quát hóa của các mô hình gợi ý.

4.2.2 Leave-one-last Evaluation

Chiến lược đánh giá được sử dụng phổ biến nhất trong các nghiên cứu hệ gợi ý hiện nay là Leave-one-last. Đúng như tên gọi, phương pháp này giữ lại một tương tác tích cực cuối cùng của mỗi người dùng để đưa vào tập kiểm tra (test), trong khi toàn bộ các tương tác còn lại được sử dụng để huấn luyện mô hình.

Trong quá trình đánh giá, hệ thống sẽ thực hiện như sau: với mỗi người dùng, tương tác cuối cùng được xem là mục tiêu gợi ý cần dự đoán. Sau đó, item này sẽ được trộn lẫn với N item ngẫu nhiên mà người dùng đó chưa từng tương tác trước đó (nhưng xuất hiện trong tập huấn luyện). Tập hợp gồm $N+1$ item này sẽ được đưa vào mô hình để tính điểm và xếp hạng, từ đó đánh giá khả năng của mô hình trong việc phát hiện đúng item mục tiêu.

Giá trị của N có thể được lựa chọn tùy theo quy mô của tập dữ liệu — phổ biến nhất là $N = 100$, nhưng cũng có thể là 200, 500 hoặc 1000 trong các thiết lập khác nhau. Chiến lược này cho phép kiểm tra khả năng phân biệt (*discriminative ability*) của mô hình trong không gian gợi ý thực tế, nơi mà số lượng item tiềm năng thường rất lớn.



Hình 4.1: So sánh 2 chiến lược đánh giá Full-corpus và Leave-one-last

- **Ưu điểm**: Phương pháp này giải quyết hiệu quả vấn đề chi phí tính toán cao của chiến lược *Full-corpus* bằng cách chỉ đánh giá trên một tập hợp con các item (gồm 1 item mục tiêu và N item nhiễu). Điều này giúp giảm đáng kể độ phức tạp tính toán, đặc biệt với các tập dữ liệu có số lượng item lớn. Ngoài ra, do chỉ giữ lại một tương tác cuối cùng để đánh giá, gần như toàn bộ dataset còn lại đều có thể sử dụng cho huấn luyện, từ đó mô hình được tiếp cận tối đa lượng dữ liệu sẵn có để học các đặc trưng người dùng và sản phẩm.

- **Hạn chế**:
  - Một nhược điểm quan trọng của phương pháp *Leave-one-last* là không đảm bảo phân tách dữ liệu theo một mốc thời gian cố định trên toàn bộ người dùng. Thời điểm tương tác cuối cùng của mỗi user là khác nhau, dẫn đến nguy cơ **Time leakage** – khi mà dữ liệu từ tương lai (so với tương tác cuối cùng của một số user) có thể vô tình xuất hiện trong tập huấn luyện của user khác. Tuy nhiên, mức độ ảnh hưởng của hiện tượng này còn phụ thuộc vào đặc trưng của dataset, ví dụ như việc có tồn tại một bộ phận lớn người dùng không còn hoạt động trong giai đoạn cuối của tập dữ liệu hay không - và thông thường là không quá lớn vì khoảng thời gian trong test-set là không quá lớn.
  - Nhược điểm thứ hai là các chỉ số phổ biến như Precision@k và Recall@k sẽ mất đi phần lớn ý nghĩa trong ngữ cảnh này, do chỉ có duy nhất một item mục tiêu được coi là "relevant" trong quá trình đánh giá. Khi đó, Precision@k = HitRate@k / k, còn Recall@k = HitRate@k. Vì vậy, trong chiến lược này, các chỉ số như Hit Rate và NDCG thường được sử dụng thay thế để phản ánh chính xác hơn hiệu suất mô hình.

Trong báo cáo này, tôi sẽ thực hiện đồng thời đánh giá các mô hình trên cả 2 chiến lược này, nhằm so sánh và xác định sự ảnh hưởng đến từ các nhược điểm của mỗi phương pháp đến kết quả đánh giá.

Chương 5

Huấn luyện và đánh giá

Từ tập dữ liệu cuối cùng (đã qua bước tiền xử lý), việc huấn luyện mô hình và đánh giá mô hình tuân theo một quy trình chung như sau:

- **Xây dựng ma trận tương tác user-item.** Với mô hình dựa trên mạng nơ-ron đồ thị thì ta cần phải làm thêm một bước nữa là xây dựng **đồ thị hai phía (bipartite graph)**, bản chất là cài đặt một ma trận kề biểu diễn cấu trúc tương tác của đồ thị hai phía.
- **Chia tập dữ liệu theo một trong hai chiến lược full-corpus hoặc leave-one-last**, tuỳ vào từng loại chiến lược thì kết quả đánh giá sẽ khác nhau do các nguyên nhân ảnh hưởng như cold-start. Với cách chia theo kiểu full-corpus tồn tại khá nhiều trường hợp cold-start, còn với cách chia leave-one-last thì không có user cold-start và rất ít item cold-start. Với chiến lược **full-corpus** thì tỉ lệ tập **train/valid/test** sẽ là **60/20/20**, còn riêng với chiến lược **leave-one-last** có cách chia đặc biệt nên không thể chia theo tỉ lệ như **full-corpus,** điều quan trọng cần cẩn thận là vấn đề **data leakage.** Hình 5.1, 5.2 là thống kê cold-start.

```
==== Full-corpus evaluation strategy ====

=== Cold-start analysis in valid ===
Total users in valid: 45039
--> Cold-start users in valid: 903
Total items in valid: 4121
--> Cold-start games in valid: 524

=== Cold-start analysis in test ===
Total users in test: 47274
--> Cold-start users in test: 903
Total items in test: 4609
--> Cold-start games in test: 1029
```

Hình 5.1: Thống kê cold-start theo chiến lược full-corpus

```
==== Leave-one-last evalution strategy

=== Cold-start analysis in valid ===
Total users in valid: 47274
--> Cold-start users in valid: 0
Total items in valid: 4128
--> Cold-start games in valid: 1

=== Cold-start analysis in test ===
Total users in test: 47274
--> Cold-start users in test: 0
Total items in test: 3903
--> Cold-start games in test: 1
```

Hình 5.2: Thống kê cold-start theo chiến lược leave-one-last

- **Khởi tạo mô hình, thiết lập các siêu tham số, huấn luyện mô hình** theo hàm mục tiêu. Với bài toán hệ gợi ý có hai loại hàm mục tiêu chính là **pointwise loss** và **pairwise loss**. Mỗi hàm mục tiêu sẽ tối ưu cho các tác vụ khác nhau, với **pointwise loss**, điển hình là **BCE Loss** sẽ tối ưu cho các tác vụ về phân loại (classification), hay nói cách mô hình sẽ được huấn luyện để dự đoán đúng item liên quan (relevant) của user đó? Còn với **pairwise loss** sẽ tối ưu cho các tác vụ về xếp hạng, hay nói cách khác khác mô hình sẽ được huấn luyện để xếp hạng item i (item liên quan) trên item j (item không liên quan). Do ban đầu tôi đã xác định **mục tiêu của bài toàn gợi ý là học một danh sách top-K gợi ý** nên mô hình sẽ không được huấn luyện theo **pointwise loss**, đồng thời thực nghiệm cũng cho thấy **pointwise loss** có hiệu suất kém hơn **pairwise loss.**
- **Đánh giá mô hình** trên các độ đo: **Precision@K, Recall@K, NDCG@K, HitRate@K** như đã đề cập ở chương 4 và so sánh kết quả giữa các mô hình.

5.1 Huấn luyện mô hình

5.1.1 Thuật toán huấn luyện Adam

Thuật toán Adam (Adaptive Moment Estimation) là một phương pháp tối ưu hóa mạnh mẽ và hiệu quả thường được sử dụng rộng rãi trong các mô hình học sâu. Đây là một cải tiến dựa trên các phương pháp tối ưu hóa trước đó như **AdaGrad** và **RMSProp**, đồng thời kết hợp cả ý tưởng của **Momentum** nhằm tăng tốc độ hội tụ và ổn định quá trình huấn luyện mô hình. Trong thuật toán **Adam**, mỗi tham số sẽ được cập nhật với hệ số học tập riêng biệt, điều này giúp cho việc tối ưu hóa trở nên linh hoạt hơn và không yêu cầu người dùng phải tinh chỉnh nhiều siêu tham số như trong các phương pháp truyền thống.

Công thức cập nhật trọng số trong **Adam** bao gồm nhiều bước quan trọng. Trước tiên, tại mỗi bước lặp t, ta tính gradient của hàm mất mát theo các tham số hiện tại:

$$g_t = \nabla J(\theta_t)$$

Tiếp theo, thuật toán ước lượng mô-men bậc nhất, tức là trung bình trượt mũ của gradient, theo công thức:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Đồng thời, mô-men bậc hai, hay còn gọi là phương sai trượt mũ của gradient, cũng được cập nhật như sau:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Tuy nhiên, vì các giá trị ban đầu của m_t và v_t đều được khởi tạo bằng 0, nên ở những bước đầu tiên, các giá trị này có xu hướng bị lệch thấp so với thực tế. Để khắc phục vấn đề này, Adam áp dụng kỹ thuật hiệu chỉnh thiên lệch (bias correction):

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Sau khi có các giá trị mô-men đã được hiệu chỉnh, tham số của mô hình sẽ được cập nhật theo công thức sau:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

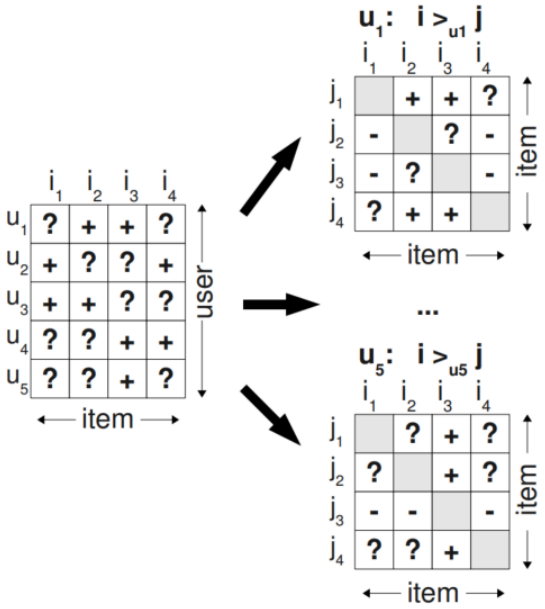
Trong đó, α là hệ số học tập (learning rate), β_1 và β_2 là các hệ số suy giảm mũ (thường được chọn lần lượt là 0.9 và 0.999), ϵ là một hằng số rất nhỏ (khoảng 10^{-8}) để tránh chia cho 0.

Ưu điểm nổi bật của thuật toán **Adam** là khả năng tự động điều chỉnh hệ số học tập cho từng tham số riêng biệt, nhờ đó giúp mô hình hội tụ nhanh hơn và ổn định hơn mà không đòi hỏi quá nhiều công sức trong việc lựa chọn và điều chỉnh thủ công hệ số học tập ban đầu. Ngoài ra, Adam còn tích hợp cả kỹ thuật **Momentum** thông qua mô-men bậc nhất và kỹ thuật chuẩn hóa gradient theo hướng tương tự **RMSProp**, giúp giảm dao động và tránh bị mắc kẹt tại các vùng phẳng hoặc cực trị địa phương. Thuật toán này đặc biệt hiệu quả khi áp dụng trên các mô hình có số lượng tham số lớn và dữ liệu đa dạng, phức tạp như mạng nơ-ron sâu.

So với thuật toán **Gradient Descent** truyền thống, **Adam** thể hiện rõ sự vượt trội về nhiều mặt. Trong **Gradient Descent**, hệ số học tập là cố định và nếu không được chọn phù hợp, mô hình có thể gặp khó khăn trong việc hội tụ hoặc mất rất nhiều thời gian để đạt đến nghiệm tối ưu. Ngược lại, **Adam** tự động thích nghi hệ số học tập theo từng bước huấn luyện và theo từng chiều của không gian tham số, mang lại tốc độ hội tụ nhanh hơn đáng kể. Đồng thời, nhờ vào việc sử dụng mô-men bậc nhất và bậc hai, **Adam** có khả năng điều chỉnh hướng di chuyển trong không gian tìm kiếm một cách linh hoạt hơn, giúp tránh xa các điểm cực trị địa phương hoặc vùng có độ dốc thấp. Nhờ những lợi thế này, **Adam** ngày càng được ứng dụng phổ biến trong các mô hình học máy hiện đại, đặc biệt là trong lĩnh vực trí tuệ nhân tạo và xử lý ngôn ngữ tự nhiên.

5.1.2 Hàm mất mát BPR

Khác với hàm mất mát BCE theo góc độ classification, BPR lại là một hàm mất mát theo góc độ ranking. Hàm mất mát này không sử dụng trực tiếp nhãn nhị phân của dữ liệu mà xây dựng từng cặp item đầu vào giữa 1 item có nhãn bằng 1 (positive item) và 1 item có nhãn bằng 0 (negative item) - do đó có thể xem là *self-supervised learning.*



Hình 5.3: Bayesian Personalized Ranking

Ý tưởng của hàm loss này là giả định rằng mọi item mà người dùng đã tương tác (positive) luôn vượt trội hơn các item mà người dùng chưa tương tác (negative). Khi đó, dù dữ liệu thu được là tường minh hay tiềm ẩn, ta luôn có thể thu được một **tập thứ tự một phần (*partial order*)** của các item. Và mục tiêu của BPR là cực đại hóa khả năng mở rộng tập thứ tự này thành tập thứ tự toàn phần.

$$\max_{\Theta} \prod_{u \in U} P(>_u | \Theta) = \max_{\Theta} \prod_{(u,i,j) \in U \times I \times I} P(i >_u j \mid \Theta)^{\delta_{(u,i,j) \in D_S}} \cdot (1 - P(i >_u j \mid \Theta))^{\delta_{(u,j,i) \in D_S}}$$

Với δ là hàm thuộc tính (indicator function):

$$\delta(u,i,j) = \begin{cases} 1 & \text{nếu } u \text{ thích } i \text{ hơn } j \\ 0 & \text{còn lại} \end{cases}$$

Vì tính *khả so sánh toàn phần* và *phi đối xứng*, ta có thể đơn giản công thức về:

$$\max_{\Theta} \prod_{u \in U} P(>_u | \Theta) = \max_{\Theta} \prod_{(w,i,j) \in \mathcal{D}_S} p(i >_w j \mid \Theta) \tag{5.1}$$

Hiện tại quan hệ thứ tự của chúng ta mới chỉ là một phần nên lúc này một quan hệ $i >_w j$ bất kì:

$$p(i >_w j \mid \Theta) = \begin{cases} 1 & \text{nếu } i >_w j \\ 0 & \text{còn lại} \end{cases}$$

Ta cần tổng quát hóa quan hệ thứ tự này thành quan hệ thứ tự toàn phần. Chọn $p(i >_w j \mid \Theta) = \sigma(\hat{y}_{u,i} - \hat{y}_{u,j})$, thay vào biểu thức (3.1) và lấy logarith phủ định ta thu được công thức *BPR Loss*:

$$\begin{aligned}\mathcal{L}_{\text{BPR}} &= -\ln p(\Theta \mid >_u) \\ &= -\sum_{(u,i,j)\in\mathcal{D}_S} \ln \sigma(\hat{y}_{u,i} - \hat{y}_{u,j}) - \lambda_\Theta \|\Theta\|^2 \\ &= -\sum_{(u,i,j)\in\mathcal{D}_S} \ln \sigma(\hat{y}_{u,i,j}) - \lambda_\Theta \|\Theta\|^2\end{aligned}$$

Vì số lượng cặp positive - negative là rất lớn, thông thường khi tính *BPR Loss* chỉ sample một số lượng nhỏ negative item cho mỗi positive item, điển hình là tỉ lệ 1 : 1, 1 : 3 hay 1 : 10 . Ngoài ra, hàm mất mát này vốn được thiết kế dành riêng cho *implicit feedback*, tuy nhiên cũng có thể tùy chỉnh cách định nghĩa negative item và chiến lược negative sampling cho phù hợp.

5.1.3 Tinh chỉnh siêu tham số (hyperparameter tuning)

Khi huấn luyện để kết quả so sánh phản ánh chính xác và đồng nhất. Tôi thiết lập các **siêu tham số (hyperparameter)** được thiết lập chung như sau: *batch size* = 8192, *num epochs* = 50, *embedding dim* = 64, *initial learning rate* = 0.01, *scheduler* = **CosineAnnealingLR** (*min learning rate* = 0.0005 sau *num epochs*), *num sample negative edges* = 3

- **Với mô hình NeuMF:** *num layer* = 3, *dropout* = 0.4, *latent_factor* = 32, *layers* = [128, 64, 32] (số lượng neural)
- **Với mô hình NGCF:** *num layer* = 3, *message dropout* = 0.4, *node dropout* = 0.6, *l2 regularization* = 1e-3.
- **Với mô hình LighGCN:** *num layer* = 3, *l2 regularization* = 1e-4

Trong quá trình huấn luyện, tôi tiếp tục chia tập train thành tập train và validate, sau đó tôi thực hiện tính toán các giá trị như **Train Loss, Valid Loss, NDCG@10, HitRate@10**. Việc tính toán **Train Loss** và **Valid Loss** với mục đích theo dõi mô hình có đang bị quá khớp hay không (overfitting), căn cứ vào đó để tinh chỉnh siêu tham số một cách phù hợp. Việc tính thêm cả **NDCG@10, HitRate@10** là một thao tác đáp ứng tính thực tiễn khi mô hình cần phải tối ưu các độ đo về xếp hạng, do đó chiến lược lựa chọn mô hình tốt nhất khi nó đáp ứng cả hai tiêu chí: **maximum NDCG@10@10**, **maximum HitRate@10.** Lưu ý rằng việc tính **NDCG@10, HitRate@10** trong quá trình huấn luyện đã tạm thời loại bỏ hoàn toàn các trường hợp **cold-start user** và các trường hợp **cold-start item** nên các các độ đo này thường cho ra kết quả vượt trội khi tính toán trên tập test, nơi xét đến cả trường hợp cold-start. Về vấn đề xử lý cold-start khi tính toán các độ đo trên tập test, với **từng cặp cold-start user-item** thì mô hình sẽ đưa ra dự đoán một cách trung lặp, đầu ra điểm dự đoán (score) sẽ là 0.0 (sau khi qua hàm sigmoid sẽ là 0.5) thể hiện rằng user này không thích, không ghét item đó.

5.2 Kết quả huấn luyện và đánh giá

Bảng 5.1: Kết quả đánh giá mô hình trên **Full-corpus** và **Leave-one-last**

Mô hình	Full-corpus				Leave-one-last	
	Precision@10	Recall@10	NDCG@10	HitRate@10	NDCG@10	HitRate@10
MF-BPR	0.0111	0.0123	0.0143	0.0927	0.1614	0.3151
NeuMF	0.0127	0.0140	0.0161	0.1029	0.1655	0.3146
NGCF	0.0244	0.0281	0.0315	0.1837	**0.2685**	**0.5165**
LightGCN	**0.0256**	**0.0298**	**0.0331**	**0.1906**	0.2681	0.5069

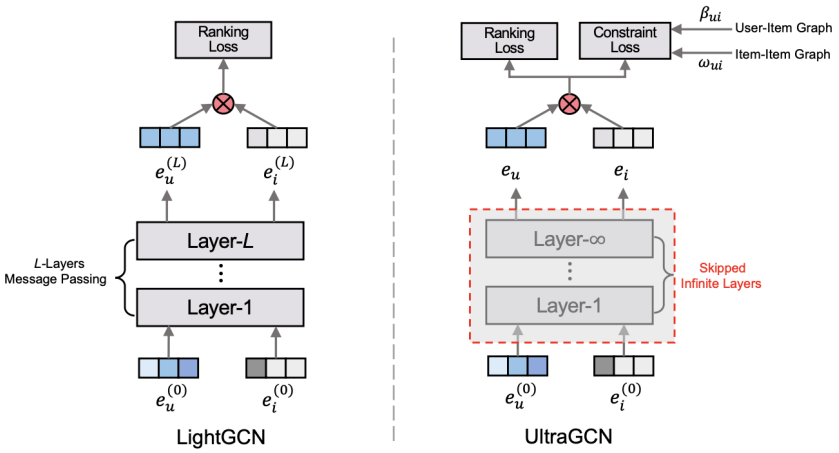
Nhận xét kết quả: Đúng như nhiều nghiên cứu, **NeuMF** vượt trội hơn so với **MF** vì nó kết hợp cả hai yếu tố tuyến tính và phi tuyến khi mã hoá tín hiệu cộng tác của user-item, trong khi **MF-BPR** chỉ xét đến yếu tố tuyến tính này. Các mô hình GNN cho hệ gợi ý dựa trên lọc cộng tác thực sự là một cách tiếp cận ***tiến tiến (state-of-the-art)*** nhất trong hệ gợi ý khi cho ra kết quả vượt xa các mô hình còn lại.

- Tuy nhiên nhìn thật kỹ ở chiến lược đánh giá ***Full-corpus***, mô hình **LightGCN** vượt trội hoàn so với **NGCF**, điều này phản ánh bản chất ưu điểm của chiến lược, tức là phản ánh tốt môi trường chiến khai thức tế, không bị ảnh hưởng bởi yếu tố ngẫu nhiên giống như chiến lược **Leave-one Last.**
- Còn với chiến lược đánh giá ***Leave-one-last***, có thể bị ảnh hưởng bởi yếu tố ngẫu nhiên trong việc lấy mẫu. Vì chiến lược lấy mẫu **tuân theo phân phối đều (uniformly negative sampling**, các **mẫu âm dễ (easy negative)** (nằm xa mẫu dương trong không gian ẩn) có thể được lấy nhiều hơn khi đánh giá **NGCF.** Điều này khiến cho **NGCF** dễ dàng phân biệt các **mẫu âm dễ** này và đưa ra kết quả **NDCG@10, Hitrate@10** tốt hơn so với **LightGCN**. **Giải pháp có thể áp dụng:** Chạy thêm nhiều lần đánh giá **NGCF, LightGCN** trên các độ đo và tính trung bình để có được kết luận đúng đắn hơn.

Phụ lục

Hướng phát triển tiếp trong tương lai

Ta có thể cải thiện chất lượng của hệ gợi ý bằng việc áp dụng một mô hình tiên tiến hơn mô hình **LightGCN** là mô hình **UltraGCN**. Ý tưởng đơn giản của mô hình **UltraGCN:** Thay vì huấn luyện các vector embedding của user và item thông qua cơ chế ***lan truyền thông điệp hữu hạn lớp (finite message propagation layer)***, thì **mô hình này sẽ huấn luyện trực tiếp các vector embedding này dựa trên sự kết hợp của hai hàm mất mát mục tiêu là: Ranking Loss và Constraint Loss** được thể hiện ở hình 5.4, nói cách khác nó loại bỏ hoàn toàn cơ chế lan truyền thông điệp hữu hạn tầng.



Hình 5.4: So sánh UltraGCN với LightGCN

Các công cụ, thư viện sử dụng

Để thực hiện quá trình tiền xử lý dữ liệu cho bộ dữ liệu **Game Recommendations on Steam**, các thư viện và công cụ sau đây trong hệ sinh thái Python đã được sử dụng:

- **Pandas:**
  - *Mục đích:* Là thư viện chính cho việc thao tác và phân tích dữ liệu dạng bảng.
  - *Ứng dụng cụ thể:* Đọc các tệp CSV **("games.csv", "users.csv", "recommendations.csv")** và JSON (nếu có metadata) vào cấu trúc DataFrame; thực hiện các thao tác làm sạch dữ liệu như xử lý giá trị thiếu, loại bỏ trùng lặp; biến đổi kiểu dữ liệu; hợp nhất (merge/join) các DataFrame từ các tệp khác nhau; thực hiện các phép tính tổng hợp và nhóm dữ liệu.
- **NumPy:**
  - *Mục đích:* Cung cấp hỗ trợ cho các mảng đa chiều và các phép toán số học hiệu suất cao.
  - *Ứng dụng cụ thể:* Thường được sử dụng ngầm bởi Pandas cho các thao tác tính toán trên cột dữ liệu số. Có thể được sử dụng trực tiếp cho các phép biến đổi toán học phức tạp hoặc khi cần tối ưu hóa hiệu suất.
- **Scikit-learn (sklearn):**

  - *Mục đích:* Thư viện học máy toàn diện, cung cấp nhiều công cụ hữu ích cho tiền xử lý.
  - *Ứng dụng cụ thể:* `sklearn.preprocessing`: Bao gồm các công cụ để chuẩn hóa/tiêu chuẩn hóa dữ liệu (ví dụ: `MinMaxScaler`)

- **Matplotlib & Seaborn:**
  - *Mục đích:* Các thư viện trực quan hóa dữ liệu.
  - *Ứng dụng cụ thể:* Tạo biểu đồ (histogram, box plot, scatter plot, heat matrix) để khám phá phân phối dữ liệu, xác định các giá trị ngoại lai, hiểu mối quan hệ giữa các biến, và kiểm tra kết quả của các bước tiền xử lý.

Ở giai đoạn **huấn luyện và đánh giá mô hình,** chúng tôi sử dụng thư viện **PyTorch & Pytorch Geometric**

- *Mục đích:* Cài đặt, huấn luyện và đánh giá các mô hình **học biểu diễn ẩn (latent representation learning).**
- *Ứng dụng cụ thể:* Sử dụng các module của PyTorch để:
  - Xây dựng kiến trúc mạng cho các mô hình **MF, NeuMF, NGCF, LightGCN,...**
  - Tạo lớp `Dataset` và `DataLoader` để xử lý batch và negative sampling.
  - Định nghĩa hàm mất mát (BCE, BPR) và thuật toán tối ưu Adam cho việc huấn luyện.
  - Tận dụng GPU để tăng tốc quá trình huấn luyện và đánh giá hiệu năng mô hình.

Tài liệu tham khảo

[1] Batra, S., Sharma, V., Sun, Y., Wang, X., & Wang, Y. (2023, May 3). *Steam Recommendation System*. arXiv preprint arXiv:2305.04890. https://arxiv.org/abs/2305.04890

[2] Tiep, V. H. (2017, May 17). *Bài 23: Content-based Recommendation Systems*. Machine Learning cơ bản. https://tinyurl.com/5x4ecx9f

[3] Tiep, V. H. (2017, May 24). *Bài 24: Neighborhood-Based Collaborative Filtering*. Machine Learning cơ bản. https://tinyurl.com/4y7bsfa9

[4] Tiep, V. H. (2017, May 31). *Bài 25: Matrix Factorization Collaborative Filtering*. Machine Learning cơ bản. https://tinyurl.com/mt3jxx6t

[5] Tiep, V. H. (2017, Jun 7). *Bài 26: Singular Value Decomposition*. Machine Learning cơ bản. https://tinyurl.com/y3eb5h3m

[6] Rendle, S. (2012, May 11). *BPR: Bayesian Personalized Ranking from Implicit Feedback*. arXiv preprint arXiv:1205.2618. https://arxiv.org/pdf/1205.2618

[7] Wu, H., Liu, X., Ma, C., & Tang, R. (2022). *Adapting Triplet Importance of Implicit Feedback for Personalized Recommendation. arXiv preprint arXiv:2208.01709.* https://arxiv.org/pdf/2208.01709

[8] Ma, H., Xie, R., Meng, L., Feng, F., Du, X., Sun, X., Kang, Z., & Meng, X. (2024). *Negative Sampling in Recommendation: A Survey and Future Directions*. arXiv preprint arXiv:2409.07237. https://arxiv.org/pdf/2409.07237

[9] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017, August 26). *Neural Collaborative Filtering*. arXiv preprint arXiv:1708.05031. https://arxiv.org/abs/1708.05031

[10] Wu, S., Sun, F., Zhang, W., Xie, X., & Cui, B. (2022, December 3). *Graph Neural Networks in Recommender Systems: A Survey*. ACM Computing Surveys, 55(5), 97. https://doi.org/10.1145/3535101

[11] Gao, C., Zheng, Y., Li, N., Li, Y., Qin, Y., Piao, J., Quan, Y., Chang, J., Jin, D., He, X., & Li, Y. (2023, March 3). A *Survey of Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions*. ACM Transactions on Recommender Systems, 1(1), 3. https://dl.acm.org/doi/10.1145/3568022

[12] Hamilton, W. L. (2020, September). *The Graph Neural Network Model*. In Graph Representation Learning Book. Retrieved from https://www.cs.mcgill.ca/~wlh/grl_book/

[13] Hamilton, W. L. (2020, September). *Graph Neural Networks in Practice*. In Graph Representation Learning Book. Retrieved from https://www.cs.mcgill.ca/~wlh/grl_book/

[14] Kipf, T. N., & Welling, M. (2017). *Semi-supervised classification with graph convolutional networks.* International Conference on Learning Representations (ICLR). https://arxiv.org/pdf/1609.02907

[15] Wang, X., He, X., Wang, M., Feng, F., & Chua, T.-S. (2020). *Neural Graph Collaborative Filtering.* arXiv preprint arXiv:1905.08108 https://arxiv.org/pdf/2002.0212

[16] He, X., Deng, K., Li, Y., & Zhang, Y. (2020). LightGCN: *Simplifying and Powering Graph Convolution Network for Recommendation.* arXiv preprint arXiv:2002.02126. https://arxiv.org/pdf/2002.02126

[17] Meng, Z., McCreadie, R., Macdonald, C., & Ounis, I. (2020, July 26). *Exploring Data Splitting Strategies for the Evaluation of Recommendation Models.* arXiv preprint arXiv:2007.13237. https://arxiv.org/pdf/2007.13237

[18] Campos, P. G., Díez, F. J., & Cantador, I. (2012). *Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols.* User Modeling and User-Adapted Interaction, 22(4-5), 421-455. https://doi.org/10.1007/s11257-012-9136-x

[19] Kula, P., Wróbel, Ł., & Szymański, J. (2020). *Modeling Online Behavior in Recommender Systems: The Importance of Temporal Context.* arXiv preprint arXiv:2007.13237. https://arxiv.org/pdf/2009.08978

[20] Cerqueira, V., Torgo, L., & Mozetic, I. (2019). *Evaluating time series forecasting models.* arXiv preprint arXiv:1905.11744. https://arxiv.org/pdf/1905.11744