ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA TOÁN - TIN HỌC

SEMINAR KHOA HỌC DỮ LIỆU

Đề tài: Recommendation system

Lớp: 20TTH

Giảng viên hướng dẫn: Ngô Minh Mẫn

◇ THÀNH VIÊN ◇

STT	Tên	MSSV
1	Hoàng Hải Đăng	20280011
2	Võ Thái Bình	20280007

Ngày 16 tháng 1 năm 2024

Mục lục

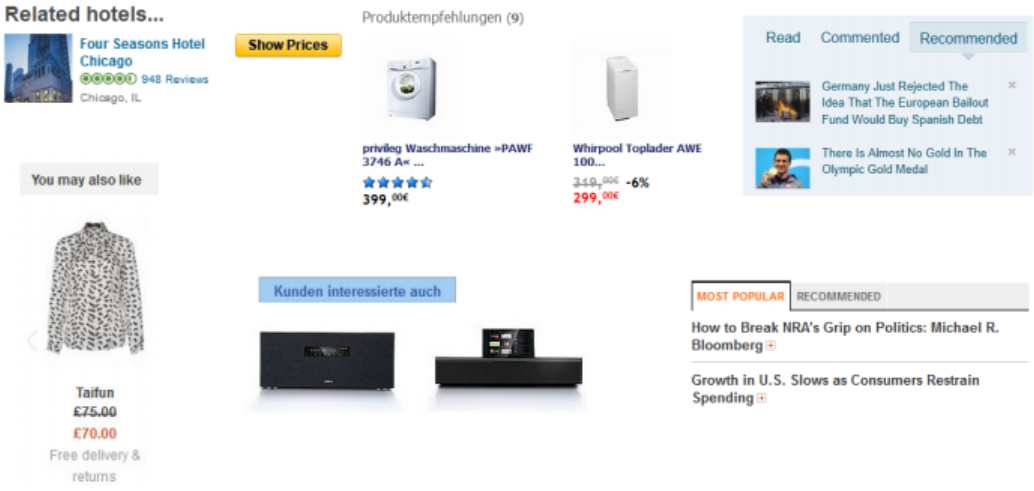
DEEP-LEARNING BASED RECOMMENDER SYSTEMS

Tóm lược

Với sự phát triển ngày càng mạnh mẽ của nội dung số và sự phức tạp ngày càng tăng của sở thích người dùng, các phương pháp đề xuất truyền thống đối mặt với những hạn chế trong việc cung cấp đề xuất chính xác và cá nhân hóa. Các kỹ thuật Deep learning đã nổi lên như là những công cụ mạnh mẽ để giải quyết những thách thức này, tận dụng khả năng của các mạng neural để trích xuất các mô hình phức tạp từ lượng lớn dữ liệu người dùng. Bài đánh giá này bao gồm các khái niệm chính, kiến trúc và thuật toán được sử dụng trong các hệ thống đề xuất dựa trên Deep learning, nhấn mạnh những điểm mạnh và hạn chế của chúng. Ngoài ra, bài đánh giá còn đàm phán về các xu hướng nghiên cứu hiện tại, bao gồm tích hợp tính minh bạch, công bằng và khả năng giải thích trong các hệ thống đề xuất. Thông qua phân tích toàn diện, bài đánh giá này nhằm cung cấp cho các nhà nghiên cứu và người thực hành cái nhìn sâu sắc về các kỹ thuật hiện đại, xu hướng mới nổi và các hướng nghiên cứu tiềm năng trong tương lai cho lĩnh vực hệ thống đề xuất dựa trên Deep Learning, đặc biệt là hệ thống đề xuất tuần tự (Sequential Recommendation)

1 Giới thiệu

1.1 Recommender System



Hệ thống đề xuất (Recommender System) là một dạng của hệ thống lọc thông tin, nó được sử dụng để dự đoán sở thích hay xếp hạng mà người dùng (user) có thể dành cho một mục thông tin (item) nào đó mà họ chưa xem xét tới trong quá khứ (item có thể là bài hát, bộ phim, đoạn video clip, sách, bài báo,..). Recommender System là một trong những ứng dụng phổ biến nhất của Khoa học dữ liệu ngày nay. Hầu hết mọi công ty công nghệ lớn đều đã áp dụng chúng dưới một số hình thức:

- Amazon sử dụng nó để đề xuất sản phẩm cho khách hàng
- Netflix sử dụng gợi ý phim cho người dùng
- YouTube sử dụng nó để đề xuất các video và quyết định video sẽ phát tiếp theo trên chế độ tự động phát.

Ứng dụng của Recommender System rất rộng rãi, từ các dịch vụ trực tuyến như Netflix, Spotify, Amazon đến các trang tin tức, trang web mua sắm, và nhiều lĩnh vực khác. Bằng cách sử dụng dữ liệu người dùng và phân tích thông tin về item, hệ thống đề xuất giúp cải thiện trải nghiệm người dùng, tăng doanh số bán hàng, và tối ưu hóa nội dung được hiển thị cho người dùng.

1.2 Deep Learning

Deep Learning đóng một vai trò quan trọng trong sự phát triển của Recommender System, mang lại những cải tiến đáng kể so với các phương pháp truyền thống. Deep Learning đã mang lại nhiều lợi ích cho việc phát triển Recommender System bằng cách cung cấp các phương tiện mạnh mẽ để mô hình hóa mối quan hệ phức tạp và tự động hóa quá trình học từ dữ liệu. Trong đó, nổi bật nhất có thể kể đến: Biểu diễn tuyến tính và phi tuyến tính, Embedding layers, End-to-End Learning, Học sâu từ dữ liệu thưa (Sparse data), Khả năng mở rộng,...

2 Tổng quan về Recommender System và Deep Learning

2.1 Các kĩ thuật đề xuất truyền thống trong Recommender System

Collaborative Filtering

- **Định nghĩa:** Collaborative Filtering là một phương pháp đề xuất dựa trên hành vi của cả nhóm user. Nó giả định rằng user có sở thích tương tự sẽ thích những item tương tự.
- **Loại hình:** Có hai dạng chính: User-based Collaborative Filtering (dựa trên sự giống nhau giữa các user) và Item-based Collaborative Filtering (dựa trên sự giống nhau giữa các item).
- **Ưu điểm:** Đơn giản triển khai và không đòi hỏi thông tin chi tiết về user và item.

Content-based Filtering

- **Định nghĩa:** Content-based Filtering tập trung vào đặc tính của các item và sở thích cá nhân của user. Nó đề xuất các item dựa trên mức độ tương quan giữa nội dung của item và sở thích của user.
- **Loại hình:** Xác định các đặc tính của item và tạo ra mô hình dựa trên sở thích của user với các đặc tính đó.

- **Ưu điểm:** Hoạt động tốt trong việc giải quyết vấn đề dữ liệu thưa và có khả năng giúp giải quyết cold start problem cho các item mới.

Hybrid Method

- **Định nghĩa:** Hybrid Method kết hợp Collaborative Filtering và Content-based Filtering để tận dụng ưu điểm của cả hai phương pháp.
- **Loại hình:** Có thể được chia thành hai loại chính: Model Fusion (kết hợp dự đoán từ các mô hình riêng biệt) và Feature Combination (kết hợp đặc tính từ cả hai loại phương pháp).
- **Ưu điểm:** Có thể giảm đi nhược điểm của từng phương pháp đơn lẻ và cung cấp độ chính xác cao hơn.

Giới hạn

- **Dữ Liệu Thưa (Data Sparsity):** Collaborative Filtering gặp khó khăn khi dữ liệu thưa, trong khi Content-based Filtering có thể không có đủ thông tin để tạo ra đề xuất chính xác.
- **Cold Start Problem:** Cả hai phương pháp đều gặp vấn đề khi phải đối mặt với user mới hoặc item mới mà chưa có đánh giá nào.
- **Over-Specialization:** Content-based Filtering có thể tạo ra đề xuất quá chuyên sâu và hạn chế đa dạng của các item.
- **Challenges in Combining Methods:** Hybrid Method đôi khi đòi hỏi các thuật toán phức tạp và không phải lúc nào cũng đem lại cải tiến đáng kể so với các phương pháp đơn lẻ.

2.2 Deep-learning based Recommender System

Deep-learning based Recommender System là những hệ thống đề xuất mà sử dụng các mô hình học sâu (Deep Learning) để hiểu và mô hình hóa mối quan hệ phức tạp giữa user và các item. Trong Deep-learning based Recommender System, các biểu diễn ẩn phức tạp của user và item được học từ dữ liệu để tạo ra các dự đoán chính xác về sự tương tác hoặc sự quan tâm của user đối với các item chưa được xem. Dưới đây là một số mô hình Deep-learning được sử dụng phổ biến trong Recommender System:

- Artificial Neural Networks (ANN): Mô hình ANN cố gắng mô phỏng cấu trúc của não người bằng cách sử dụng các tầng của các neural. Trong Recommender System, ANN thường được sử dụng để học các biểu diễn của user và item.
- Convolutional Neural Networks (CNN): CNN chủ yếu được thiết kế để xử lý dữ liệu không gian như hình ảnh. Trong Recommender System, chúng có thể được sử dụng để xử lý dữ liệu ảnh

hoặc mô tả hình ảnh của item để tạo ra các biểu diễn có ý nghĩa.

- Recurrent Neural Networks (RNN): RNN chủ yếu được sử dụng cho dữ liệu chuỗi, như thời gian hoặc trình tự. Trong Recommender System, chúng có thể được áp dụng cho dữ liệu mà có mối quan hệ thời gian, chẳng hạn như lịch sử tương tác của user.

- BERT (Bidirectional Encoder Representations from Transformers): BERT là một mô hình Transformer được đào tạo để hiểu ngôn ngữ tự nhiên. Trong Recommender System, BERT thường được sử dụng để tạo ra biểu diễn mạnh mẽ cho các văn bản, giúp hiểu rõ hơn về nội dung và ý định của user.

- Wide & Deep Learning: Mô hình Wide & Deep kết hợp giữa khả năng học các mối quan hệ phức tạp của mô hình Deep Learning và khả năng hiệu quả của mô hình tuyến tính để xử lý dữ liệu thưa, đặc biệt hữu ích trong môi trường Recommender System.

- Matrix Factorization with Neural Networks (MFNN): Kết hợp giữa Matrix Factorization và Neural Networks, mô hình này sử dụng đồng thời thông tin từ ma trận đánh giá và các biểu diễn học được từ mô hình Deep Learning.

Các mô hình trên đều có những đặc điểm riêng và có thể được sử dụng tùy thuộc vào loại dữ liệu và mục tiêu cụ thể của hệ thống đề xuất. Sự linh hoạt của Deep Learning giúp nó trở thành một công cụ mạnh mẽ trong việc xử lý và học từ các dạng dữ liệu đa dạng trong Recommender System.

2.3 Quá trình phát triển của Deep-learning based Recommender System

Deep Learning qua thời gian đã đem lại nhiều đổi mới và cải tiến đáng kể trong lĩnh vực Recommender System. Dưới đây là một số cột mốc quan trọng trong quá trình phát triển của Deep Learning trong hệ thống đề xuất:

- **2014 - Collaborative Deep Learning for Recommender Systems** [6]
  Ngày càng nhiều nghiên cứu bắt đầu kết hợp giữa Collaborative Filtering và Deep Learning để cải thiện độ chính xác của hệ thống đề xuất. Các mô hình như Restricted Boltzmann Machines (RBM) được sử dụng để học các biểu diễn tốt hơn cho user và item.

- **2016 - Wide & Deep Learning for Recommender Systems** [7]
  Mô hình Wide & Deep Learning của Google xuất hiện, là một bước tiến quan trọng trong việc kết hợp khả năng học sâu với khả năng xử lý tuyến tính. Mô hình này đã chứng minh hiệu suất tốt trong việc giải quyết vấn đề dữ liệu thưa và mang lại sự linh hoạt trong việc xử lý nhiều loại dữ liệu.

- **2018 - BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding** [9]

Sự ra đời của BERT đánh dấu một bước nhảy vọt trong việc hiểu và biểu diễn ngôn ngữ tự nhiên. BERT đã được tích hợp vào Recommender System để cải thiện khả năng hiểu nội dung và ý định user, đặc biệt trong các hệ thống đề xuất sử dụng văn bản và mô tả item.

- **2020 - Neural Collaborative Filtering vs. Matrix Factorization Revisited** [8]
  Xu hướng kết hợp giữa Deep Learning và Matrix Factorization trở nên phổ biến. Mô hình như Matrix Factorization with Neural Networks (MFNN) đã đưa ra cách tiếp cận mới trong việc sử dụng thông tin từ ma trận đánh giá và lợi ích của Deep Learning.

3 Phân loại các mô hình Deep Learning

Phân loại theo Architecture:

Collaborative Filtering with Deep Neural Networks (CF-DNN):

- Khái niệm: Kết hợp giữa Collaborative Filtering và Deep Neural Networks để học biểu diễn phức tạp cho user và item.
- Ví dụ: Sử dụng mạng neural để học các biểu diễn cho user và item, và sau đó kết hợp thông tin này để đưa ra dự đoán về sở thích của user cho các item chưa được đánh giá.
- Điểm mạnh: Có khả năng hiểu được mối quan hệ phức tạp giữa user và item dựa trên dữ liệu tương tác.
- Hạn chế: Đối mặt với thách thức của dữ liệu thưa và khó áp dụng cho user mới.

Content-Based Models with Embedding Layers (CBF-Embed):

- Khái niệm: Sử dụng lớp nhúng để biểu diễn đặc tính của item và sở thích của user.
- Ví dụ: Mô hình học biểu diễn cho các đặc tính như thể loại, diễn viên, hoặc từ khóa từ nội dung item, và sau đó sử dụng những biểu diễn này để đề xuất item cho user.
- Điểm mạnh: Hoạt động tốt trong việc giải quyết vấn đề dữ liệu thưa và có khả năng giải quyết cold start problem cho các item mới.
- Hạn chế: Giới hạn trong việc hiểu được sở thích phức tạp của user và không thể khám phá được những sở thích mới.

Sequential Recommendation with RNNs (Seq-RNN):

- Khái niệm: Sử dụng Recurrent Neural Networks (RNNs) để mô hình hoá mối quan hệ thời gian giữa các tương tác.

- Ví dụ: Đưa vào các item một user đã tương tác trước đó và sử dụng mô hình RNN để dự đoán item tiếp theo trong chuỗi tương tác của họ.
- Điểm mạnh: Hiệu quả trong việc đề xuất các item liên quan đến lịch sử tương tác của user.
- Hạn chế: Có thể đối mặt với vấn đề khi có quá nhiều tương tác và thời gian giữa chúng dài, gây khó khăn trong việc học mối quan hệ.

Phân loại theo Learning Methods:

Wide & Deep Learning:

- Khái niệm: Kết hợp khả năng học sâu với khả năng xử lý tuyến tính để cải thiện độ chính xác và linh hoạt.
- Ví dụ: Trong hệ thống đề xuất phim, mô hình có thể học cả những mối quan hệ tuyến tính như thời lượng phim và các mối quan hệ phi tuyến tính như sở thích đặc biệt của user trong thời gian gần đây.
- Điểm mạnh: Cung cấp sự linh hoạt trong việc kết hợp thông tin tuyến tính và phi tuyến tính. Đặc biệt hiệu quả khi có nhiều loại dữ liệu và đặc trưng.
- Hạn chế: Phức tạp hóa quá trình triển khai và đòi hỏi lượng dữ liệu lớn.

Matrix Factorization with Neural Networks (MFNN):

- Khái niệm: Kết hợp giữa Deep Learning và Matrix Factorization để sử dụng thông tin từ ma trận đánh giá và lợi ích của học sâu.
- Ví dụ: Trong hệ thống đề xuất sản phẩm, mô hình có thể học biểu diễn sở thích ẩn của user và các đặc trưng ẩn của sản phẩm để dự đoán xếp hạng.
- Điểm mạnh: Kết hợp ưu điểm của cả hai phương pháp, giúp mô hình hiệu quả hơn trong việc học biểu diễn. Đặc biệt hiệu quả khi có dữ liệu thưa.
- Hạn chế: Yêu cầu cấu hình tham số phức tạp và có thể đòi hỏi thời gian và nguồn lực lớn trong quá trình training.

Phân loại theo Phạm Vi Áp Dụng:

BERT (Bidirectional Encoder Representations from Transformers) for Recommender System:

- Khái niệm: Sử dụng mô hình ngôn ngữ BERT để cải thiện khả năng hiểu nội dung và ý định user trong hệ thống đề xuất

- Ví dụ: Nếu user thường xem các bộ phim hành động và phiêu lưu, hệ thống có thể sử dụng BERT để hiểu và tìm ra mối quan hệ giữa từ khóa như 'hành động,' 'phiêu lưu' trong mô tả bộ phim và sở thích đặc biệt của user.

- Điểm mạnh:
  - BERT có khả năng hiểu sâu sắc các mối quan hệ ngôn ngữ tự nhiên trong mô tả, giúp nâng cao khả năng đề xuất các item có nội dung tương tự với sở thích của user.
  - BERT có thể sử dụng được cho nhiều loại item, từ sách đến phim, do khả năng hiểu ngôn ngữ tự nhiên rộng lớn.

- Hạn chế:
  - Quá trình đào tạo và triển khai BERT đòi hỏi nguồn lực tính toán lớn, có thể là một thách thức về chi phí và hiệu suất.
  - BERT có thể quá mạnh và phức tạp cho một số ứng dụng nhỏ, và việc tùy chỉnh cho ngữ cảnh cụ thể có thể đòi hỏi kiến thức chuyên sâu về xử lý ngôn ngữ tự nhiên và Machine Learning.

Mỗi loại phương pháp có những điểm mạnh và hạn chế riêng, và sự lựa chọn giữa chúng thường phụ thuộc vào đặc điểm của dữ liệu, yêu cầu của ứng dụng cụ thể, và ngữ cảnh triển khai. Điều này giúp xây dựng một hệ thống đề xuất đa dạng và hiệu quả.

4 Thách Thức Và Cơ Hội

4.1 Thách Thức

Data Sparsity: Gặp khó khăn do số lượng đánh giá của user hạn chế, dẫn đến dữ liệu thưa. Để giải quyết vấn đề này, ta có thể sử dụng các kỹ thuật như lọc cộng tác, lọc dựa trên nội dung hoặc các phương pháp kết hợp để giảm thiểu ảnh hưởng của dữ liệu thưa.

Cold Start Problem: Cần một lượng lớn dữ liệu mới để có thể bắt đầu hoạt động hiệu quả sau một khoảng thời gian không hoạt động hoặc khi mới triển khai. Để giải quyết vấn đề này, ta có thể sử dụng thông tin bối cảnh, phân loại item, hoặc mô hình Deep Learning như BERT để hiểu và đề xuất trong tình huống 'Cold Start'.

Explainability: Các mô hình Deep Learning thường khó hiểu, giảm khả năng giải thích tại sao một đề xuất được thực hiện. Do đó, ta thường nghiên cứu và phát triển các phương pháp giải thích, sử dụng các mô hình có khả năng giải thích hoặc kết hợp với các kỹ thuật giải thích bên ngoài.

Scalability: Một số mô hình Deep Learning đòi hỏi nguồn lực tính toán đáng kể, tạo ra vấn đề về khả năng mở rộng với số lượng lớn user và item. Do đó, kiến trúc mô hình nên được tối ưu hoá, tận dụng các kỹ thuật như tính toán phân tán để xử lý bộ dữ liệu lớn.

4.2 Cơ Hội

Đa dạng và Cá nhân hóa: Deep Learning mang lại cơ hội tạo ra các mô hình phức tạp có khả năng đề xuất các item phù hợp với sở thích đa dạng của user.

Ứng dụng đa lĩnh vực: Deep Learning có thể được áp dụng không chỉ trong giải trí mà còn trong thương mại điện tử, giáo dục, chăm sóc sức khỏe và nhiều lĩnh vực khác.

Khả năng thích ứng: Deep Learning có thể được thiết kế để học liên tục, giúp chúng thích ứng với sự biến đổi của sở thích user theo thời gian.

Độ trễ thấp: Các kiến trúc và tối ưu hóa tiên tiến của Deep Learning cho phép hệ thống đề xuất theo thời gian thực.

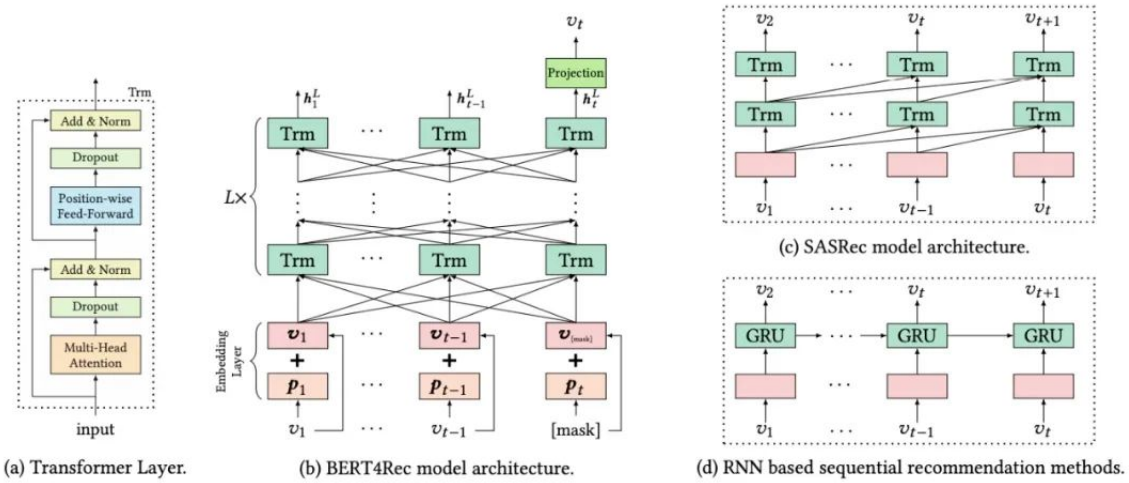
5 Mô Hình Đề Xuất Tuần Tự (Sequential Recommendation)

Mô hình đề xuất tuần tự là một hệ thống đề xuất được thiết kế để xử lý dữ liệu tuần tự, nơi mà thông tin thời gian có ý nghĩa quan trọng. Trong ngữ cảnh này, dữ liệu thường được tổ chức theo dạng chuỗi thời gian, chẳng hạn như lịch sử tương tác của user với sản phẩm hoặc nội dung trong một khoảng thời gian dài.
Các mô hình đề xuất tuần tự thường sử dụng thông tin từ các sự kiện trước đó để dự đoán và đề xuất các sự kiện tiếp theo một cách tối ưu, nhằm cung cấp trải nghiệm cá nhân hóa cho user. Các sự kiện trong chuỗi thời gian có thể là các tương tác như xem, mua, đánh giá hoặc bất kỳ hành động nào khác mà user thực hiện.

5.1 BERT4Rec (BERT for Recommender Systems)

BERT4Rec sử dụng mô hình ngôn ngữ BERT để cải thiện khả năng hiểu nội dung và ý định user trong hệ thống đề xuất. BERT4Rec giải quyết vấn đề sequential recommendation và có thể áp dụng cho nhiều ngữ cảnh.

Network Architecture



(a) Transformer Layer. (b) BERT4Rec model architecture. (c) SASRec model architecture. (d) RNN based sequential recommendation methods.

Bản chất, mô hình được đề xuất là sử dụng BERT cho một task mới là hệ thống đề xuất. Trong hình b, BERT4Rec được stack bởi L lớp bidirectional transformer. Tại mỗi layer, mô hình lặp đi lặp lại việc sửa đổi biểu diễn của mọi vị trí bằng cách trao đổi thông tin song song qua tất cả các vị trí ở lớp trước đó. Thay vì cách học truyền thông tin liên quan một cách step by step như RNN tại hình d thì cơ chế self attention mang lại cho BERT4Rec khả năng nắm bắt trực tiếp các phần phụ thuộc ở bất kì khoảng cách nào. Điểm hay của cơ chế này là cho ta một global receptive field, trong khi các phương pháp CNN thì thường cho một receptive field hạn chế. Ngoài ra, ngược lại với RNN, self attention có thể chạy song song.

Transformer Layer

Cho một chuỗi input có độ dài t, ta sẽ lặp lại việc tính toán đồng thời biểu diễn h_i^l tại mỗi layer l cho mỗi vị trí i bằng cách sử dụng transformer layer. Ta sẽ stack $h_i^l \in \mathbb{R}^d$ cùng nhau thành một ma trận $H^l \in \mathbb{R}^{t \times d}$. Việc này sẽ tận dụng khả năng tính toán đồng thời của GPU. Như trên hình a, Transformer layer Trm bao gồm 2 layer con, một multi-head self-attention sub-layer và một position-wise feed-forward network.

Multi-Head Self-Attention

Các cơ chế attention đã trở thành một phần không thể thiếu của việc mô hình hoá chuỗi trong nhiều nhiệm vụ khác nhau, nó cho phép nắm bắt sự phụ thuộc giữa các cặp biểu diễn mà không quan tâm đến khoảng cách của chúng trong chuỗi. Trong model, chúng tôi sử dụng multi-head self-attention để khai thác điểm mạnh của cơ chế này.

Đầu tiên, multi-head attention thực hiện chiếu tuyến tính H^l vào h không gian con bằng các phép chiếu tuyến tính khác nhau. Sau đó, sử dụng h hàm attention song song để cho ra biểu diễn output được nối lại và tiếp tục được chiếu:

$$\text{MH}(H^l) = [head_1; head_2; ...; head_h]W^O$$

$$head_i = \text{Attention}(H^l W_i^Q, H^l W_i^K, H^l W_i^V)$$

Trong đó, ma trận chiếu cho mỗi head là $W_i^Q \in \mathbb{R}^{d \times d/h}, W_i^K \in \mathbb{R}^{d \times d/h}, W_i^V \in \mathbb{R}^{d \times d/h}$ và $W_i^O \in \mathbb{R}^{d \times d}$ là các learnable parameter. Các tham số projection không được chia sẻ giữa các layer. Tại đây, Hàm attention là scaled dot-product:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d/h}}\right)V$$

Trong đó, truy vấn Q, khoá K, và giá trị V được chiếu từ cùng một ma trận H^l với các ma trận chiếu khác nhau. $\sqrt{d/h}$ được sử dụng để thu được phân phối attention đơn giản hơn nhằm hạn chế gradient quá nhỏ.

Position-wise Feed-Forward Network

Để ý rằng các self-attention sub-layer chủ yếu dựa trên các phép chiếu tuyến tính. Do đó, ta phải cung cấp tính chất phi tuyến tính và tương tác giữa các chiều khác nhau cho mô hình. Chúng tôi sử dụng position-wise feed-forward network cho các output của self-attention sub-layer riêng biệt và giống hệt nhau tại mỗi vị trí. Nó bao gồm 2 phép biến đổi cùng với hàm kích hoạt Gaussian Error Linear Unit (GELU) như sau:

$$\text{PFFN}(H^l) = [\text{FFN}(h_1^l)^\top; ...; \text{FFN}(h_t^l)^\top]^\top$$

$$\text{FFN}(x) = \text{GELU}(xW^{(1)} + b^{(1)})W^{(2)} + b^{(2)}$$

$$\text{GELU}(x) = x\phi(x)$$

Trong đó, $\phi(x)$ là hàm phân phối tích luỹ của phân phối chuẩn, $W^{(1)} \in \mathbb{R}^{d \times 4d}, W^{(2)} \in \mathbb{R}^{4d \times d}, b^{(1)} \in \mathbb{R}^{4d}$ và $b^{(2)} \in \mathbb{R}^d$ là các learnable parameter và được chia sẻ giữa các vị trí. Thực tế các tham số này khác nhau giữa các layer.

Stacking Transformer Layer

Bằng các thành phần ở trên, ta đã có thể ghi lại được tương tác giữa user và item sử dụng cơ chế attention. Ta cũng có thể ghi được các item transition pattern phức tạp hơn bằng cách stack các self-attention layer. Tuy nhiên, việc này làm cho mô hình sâu hơn và sẽ dẫn đến khó train hơn. Do vậy, chúng tôi sử dụng residual connection giữa mỗi 2 sublayer, tiếp sau đó là normalization layer. Ngoài ra, chúng tôi cũng áp dụng dropout cho đầu ra của mỗi sublayer, trước khi nó được chuẩn hóa. Công thức hóa như sau:

$$\text{LN}(x + \text{Dropout}(\text{sublayer}(x))$$

Tổng kết lại, BERT4Rec tinh chỉnh biểu diễn ẩn của mỗi layer như sau:

$$H^l = \text{Trm}(H^{l-1}),\ \forall i \in [1, ..., L]$$

$$\text{Trm}(H^{l-1}) = \text{LN}(A^{l-1} + \text{Dropout}(\text{PFFN}(A^{l-1})))$$

$$A^{l-1} = \text{LN}(H^{l-1} + \text{Dropout}(\text{MH}(H^{l-1})))$$

Embedding Layer

Để tận dụng thông tin vị trí của chuỗi input, chúng tôi tích hợp positional embedding vào input item embedding tại phía dưới của Transformer layer stack. Cho item v_i, biểu diễn input tương ứng h_i^0 được xây dựng bằng cách tính tổng embedding của item với positional embedding tương ứng.

$$h_i^0 = v_i + p_i$$

Trong đó, $v_i \in E$ là embedding d chiều cho item $v_i, p_i \in P$ là positional embedding d chiều cho vị trí i. chúng tôi sử dụng learnable positional embedding thay vì sinusoid embedding cố định để đạt hiệu suất tốt hơn. Ma trận positional embedding $P \in \mathbb{R}^{N \times d}$ cho phép mô hình xác định phần input mà nó đang xử lý. Tuy nhiên, nó cũng đặt ra giới hạn về độ dài câu tối đa N mà model có thể xử lý. Vì vậy, ta cần cắt bớt chuỗi input $[v_1, ..., v_t]$ thành N item cuối cùng $[vt - N + 1^u, ..., v_t]$ nếu $t > N$.

Output Layer

Sau L layer ta thu được output H^L cho tất cả item của chuỗi input. Mục tiêu tiếp theo là ta cần dự đoán item v_t tại time step t biết rằng item v_t đã bị mask và ta cần dựa vào h_t^L để dự đoán ra nó. Phân phối xác suất của các item được tính như sau:

$$P(v) = \text{softmax}(\text{GELU}(h_t^L W^P + b^P)E^\top + b^O)$$

Trong đó W^P là learnable projection matrix, b^P và b^O là các bias, $E \in \mathbb{R}^{|\mathcal{V}| \times d}$ là ma trận embedding cho tập item $\mathcal{V}$. Chúng tôi sử dụng ma trận shared item embedding trong lớp input và output để hạn chế overfitting và giảm kích thước mô hình.

Model Learning

- **Training**

  Mục tiêu của các mô hình là dự đoán item tiếp theo sẽ được tương tác. Cụ thể, trong các mô hình đề xuất đơn hướng, mục tiêu của chuỗi input $[v_1, ..., v_t]$ sẽ là $[v_2, ..., v_{t+1}]$. Tuy nhiên, vì mô hình hiện tại chúng tôi đề xuất là mô hình 2 chiều nên biểu diễn output cuối cùng của mỗi item có thể mang thông tin của item mục tiêu. Điều này làm cho việc dự đoán trở nên tầm thường và mô hình sẽ không học thêm được gì hữu ích. Một giải pháp đơn giản cho vấn đề này là ta sẽ tạo $t - 1$ mẫu từ chuỗi lịch sử hành vi user ban đầu có độ dài t, sau đó mã hoá mỗi chuỗi con bằng mô hình bidirectional và dùng để dự đoán item mục tiêu. Cách tiếp cận này có một nhược điểm là rất tốn thời gian và tài nguyên do ta cần tạo mẫu cho mỗi vị trí và thực hiện dự đoán một cách riêng biệt.

  Để cho việc training trở nên hiệu quả hơn, chúng tôi sử dụng Cloze task cho đề xuất tuần tự. Hiểu đơn giản là ta sẽ che đi một số từ và phải dự đoán từ đó là gì dựa vào các từ còn lại. Trong trường hợp này, tại mỗi bước training, chúng tôi sẽ ngẫu nhiên mask item trong chuỗi input với tỷ lệ là p và sau đó dự đoán id của item bị mask đó dựa vào bối cảnh bên trái và bên phải. Cụ thể như mô tả dưới:

Input: $[v_1, v_2, v_3, v_4, v_5] \xrightarrow{\text{randomly mask}} [v_1, [\text{mask}]_1, v_3, [\text{mask}]_2, v_5]$

Labels: $[\text{mask}]_1 = v_2, \quad [\text{mask}]_2 = v_4$

Hàm loss được sử dụng là negative log-likelihood

$$\mathcal{L} = \frac{1}{|\mathcal{S}_u^m|} \sum_{v_m \in \mathcal{S}_u^m} -\log P(v_m = v_m^* | \mathcal{S}_u^{'})$$

Trong đó $\mathcal{S}_u^{'}$ là phiên bản đã mask của lịch sử hành vi user, $\mathcal{S}_u$, $\mathcal{S}_u^m$ là các item được mask ngẫu nhiên trong đó v_m^* là label thực tế của item v_m bị mask.

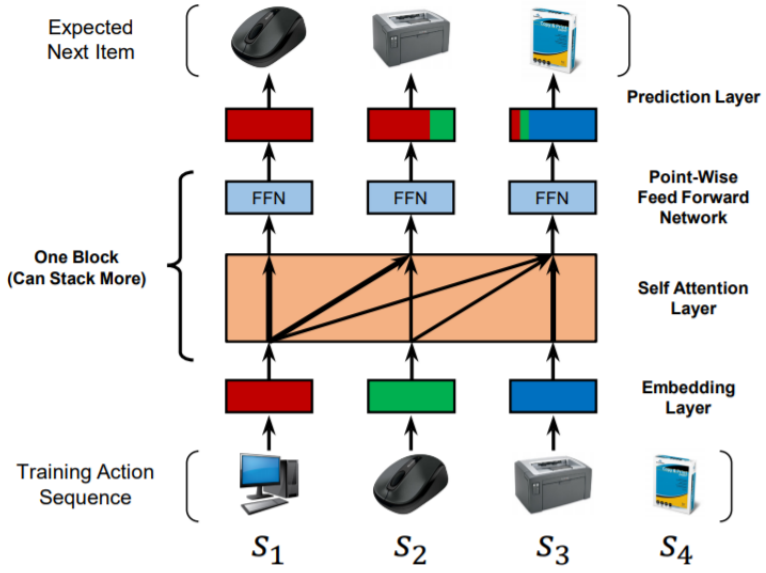
Một điểm lợi của Cloze task là nó có thể sinh nhiều mẫu để train mô hình. Giả sử với chuỗi có độ dài n, BERT4Rec có thể xác định $(n\,k)$ mẫu (trong đó ta mask ngẫu nhiên k item) tại nhiều epoch. Điều này cho phép việc train mô hình bidirectional hiệu quả hơn.

- **Testing**

  Cách làm trên bị phát sinh một thiếu sót là Cloze objective thực hiện dự đoán item đang bị mask trong khi hệ thống đề xuất cần dự đoán item ở tương lai. Để giải quyết vấn đề này, chúng tôi thêm một token '[mask]' vào cuối chuỗi hành vi của user và dự đoán item tiếp theo dựa vào biểu diễn ẩn cuối cùng của token này. Để phù hợp với nhiệm vụ đề xuất tuần tự (tức là dự đoán item cuối cùng), chúng tôi cũng tạo các sample chỉ mask item cuối cùng của chuỗi input trong quá trình train. Nó hoạt động như fine-tuning cho đề xuất tuần tự và có thể cải thiện hiệu suất của hệ thống.

5.2 SASRec (Self-Attentive Sequential Recommendation)

SASRec sử dụng cơ chế self-attention (self-attention) để hiệu quả hóa quá trình đề xuất theo chuỗi thời gian.



Embedding Layer

Chúng tôi biến đổi chuỗi đào tạo $\left(\mathcal{S}_1^u, \mathcal{S}_2^u, \ldots, \mathcal{S}_{|\mathcal{S}^u|-1}^u\right)$ thành một chuỗi có độ dài cố định $s = (s_1, s_2, \ldots, s_n)$, trong đó n đại diện cho độ dài tối đa mà mô hình của chúng tôi có thể xử lý. Nếu độ dài chuỗi lớn hơn n, chúng tôi xem xét n hành động gần đây nhất. Nếu độ dài chuỗi nhỏ hơn n, chúng tôi thêm một 'padding' item vào bên trái và lặp lại cho đến khi độ dài là n. Chúng tôi tạo một ma trận item embedding $\mathbf{M} \in \mathbb{R}^{|\mathcal{I}| \times d}$, trong đó d là số chiều ẩn, và tìm ma trận embedding đầu vào $\mathbf{E} \in \mathbb{R}^{n \times d}$, trong đó $\mathbf{E}_i = \mathbf{M}_{s_i}$. Một vector không đổi chứa các giá trị 0 làm vector embedding cho 'padding' item.

Positional Embedding: Như chúng ta sẽ thấy trong phần tiếp theo, vì mô hình self-attention không bao gồm bất kỳ module hồi quy hoặc tích chập nào, nó không thể nhận biết vị trí của các item trước đó. Vì vậy, chúng tôi thêm một learnable position embedding $\mathbf{P} \in \mathbb{R}^{n \times d}$ vào embedding đầu vào:

$$\widehat{\mathbf{E}} = \begin{bmatrix} M_{S1} + P_1 \\ M_{S2} + P_2 \\ \ldots \\ M_{Sn} + P_n \end{bmatrix}$$

Chúng tôi cũng đã thử nghiệm việc embedding vị trí cố định, nhưng phát hiện rằng điều này sẽ dẫn đến hiệu suất kém hơn trong trường hợp của chúng tôi. Chúng tôi phân tích hiệu ứng của embedding vị trí một cách tốt nhất trong các thử nghiệm của chúng tôi.

Self-Attention Block

Phương pháp chú ý dựa trên tích vô hướng có tỷ lệ (scaled dot-product attention) được định nghĩa là:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

Trong đó Q đại diện cho các truy vấn, K đại diện cho các khóa, và V đại diện cho các giá trị (mỗi hàng đại diện cho một item). Hiểu một cách trực quan, attention layer tính toán tổng có trọng số của tất cả các giá trị, trong đó trọng số giữa truy vấn i và giá trị j liên quan đến tương tác giữa truy vấn i và khóa j. Hệ số tỷ lệ $\sqrt{d}$ được sử dụng để tránh giá trị quá lớn, đặc biệt khi số chiều lớn.

Self-Attention layer: Trong các nhiệm vụ NLP như dịch máy, cơ chế chú ý thường được sử dụng với $K = V$ (ví dụ: sử dụng một bộ RNN encoder-decoder cho trình biên dịch: các trạng thái ẩn của bộ encoder là các khóa và giá trị, và các trạng thái ẩn của bộ decoder là các truy vấn). Gần đây, một phương pháp self-attention sử dụng các đối tượng giống nhau như truy vấn, khóa và giá trị đã được đề xuất. Trong trường hợp của chúng tôi, self-attention operation nhận embedding $\widehat{E}$ làm đầu vào, chuyển đổi nó thành ba ma trận thông qua các phép chiếu tuyến tính và đưa chúng vào một lớp chú ý (attention layer):

$$\mathbf{S} = \text{SA}(\widehat{\mathbf{E}}) = \text{Attention}\left(\widehat{\mathbf{E}}\mathbf{W}^Q, \widehat{\mathbf{E}}\mathbf{W}^K, \widehat{\mathbf{E}}\mathbf{W}^V\right)$$

Trong đó, các ma trận chiếu $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d \times d}$. Phép chiếu tuyến tính giúp cho mô hình linh

hoạt hơn. Ví dụ, mô hình có thể học được các tương tác không đối xứng (tức là <truy vấn i, khóa j> và <truy vấn j, khóa i> có thể có tương tác khác nhau).

Causality: Do tính chất của chuỗi, mô hình nên chỉ xem xét t item đầu tiên khi dự đoán item thứ $t+1$. Tuy nhiên, output thứ t của lớp self-attention (S_t) chứa embedding của các item tiếp theo, khiến cho mô hình không rõ ràng. Do đó, chúng tôi điều chỉnh attention bằng cách chặn tất cả các liên kết giữa Q_i và K_j $(j > i)$

Point-Wise Feed-Forward Network: Mặc dù hệ thống self-attention có khả năng tổng hợp tất cả các embedding của các item trước đó với trọng số thích ứng, nó vẫn là một mô hình tuyến tính. Để tạo nên tính phi tuyến cho mô hình và xem xét tương tác giữa các chiều ẩn khác nhau, chúng tôi áp dụng một point-wise two-layer feed-forward network cho tất cả các S_i một cách đồng nhất (chia sẻ tham số):

$$\mathbf{F}_i = \mathrm{FFN}\left(\mathbf{S}_i\right) = \mathrm{ReLU}\left(\mathbf{S}_i\mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right)\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

Trong đó $W^{(1)}, W^{(2)}$ là ma trận $d \times d$ và $b^{(1)}, b^{(2)}$ là vector d chiều. Lưu ý rằng không có tương tác giữa S_i và S_j $(i \neq j)$, nghĩa là chúng ta vẫn tránh được việc thông tin bị rò rỉ (từ sau ra trước)

Stacking Self-Attention Blocks

Sau khối self-attention đầu tiên, F_i về cơ bản tổng hợp tất cả embedding của các item trước đó (tức là $\widehat{E}_j, j \leq i$). Tuy nhiên sẽ hữu ích khi học các biến thể item phức tạp hơn thông qua một khối self-attention khác dựa trên F. Cụ thể, chúng tôi xếp chồng khối self-attention (tức là một self-attention layer và một feed-forward network), và khối thứ b $(b > 1)$ được định nghĩa như sau:

$$\mathbf{S}^{(b)} = \mathrm{SA}\left(\mathbf{F}^{(b-1)}\right),$$
$$\mathbf{F}_i^{(b)} = \mathrm{FFN}\left(\mathbf{S}_i^{(b)}\right), \quad \forall i \in \{1, 2, \ldots, n\},$$

và khối thứ nhất được định nghĩa là $S^{(1)} = S$ và $F^{(1)} = F$

Tuy nhiên, khi network đi sâu hơn, một số vấn đề trở nên trầm trọng: 1) Dung lượng mô hình tăng dẫn đến overfitting; 2) Quá trình training trở nên không ổn định (do mất mát đạo hàm,...); và 3) các mô hình nhiều tham số thường đòi hỏi thời gian train nhiều hơn. Chúng tôi thực hiện các thao tác sau để giảm nhẹ những vấn đề này:

$$g(x) = x + \mathrm{Dropout}(g(\mathrm{LayerNorm}(x))),$$

Trong đó, $g(x)$ đại diện cho self-attention layer hoặc feed-forward network. Nói cách khác, đối với lớp g trong mỗi khối, chúng tôi áp dụng chuẩn hóa lớp (layer normalization) trên input x trước khi đưa vào g, áp dụng dropout trên output của g và thêm input x vào output cuối cùng. Chúng tôi sẽ mô tả các phép toán này ở dưới.

Residual Connections: Trong một số trường hợp, các mạng neural đa tầng đã chứng minh khả năng học các đặc trưng có ý nghĩa theo cấp độ. Tuy nhiên, việc thêm nhiều lớp không đồng nghĩa với hiệu suất tốt hơn cho đến khi mô hình mạng dư (residual networks) được đề xuất. Ý tưởng cốt lõi

đằng sau mạng dư là truyền các đặc trưng ở các tầng thấp lên các tầng cao hơn thông qua kết nối dư. Do đó, nếu các đặc trưng ở tầng thấp có ý nghĩa, mô hình có thể dễ dàng truyền chúng lên tới tầng cuối cùng. Tương tự, chúng tôi giả định rằng kết nối dư cũng hữu ích trong trường hợp của chúng tôi. Ví dụ, các phương pháp đề xuất tuần tự hiện tại đã chỉ ra rằng item cuối cùng được tương tác đóng vai trò quan trọng trong việc dự đoán item tiếp theo. Tuy nhiên, sau một vài khối self-attention, embedding của item cuối cùng đã bị lan truyền vào tất cả các item trước đó; thêm kết nối dư để truyền embedding của item cuối cùng lên tới tầng cuối sẽ giúp mô hình tận dụng thông tin ở các tầng thấp hơn một cách dễ dàng.

Layer Normalization: Chuẩn hóa lớp (Layer normalization) được sử dụng để chuẩn hóa input theo từng đặc trưng (tức là có trung bình bằng 0 và phương sai bằng 1). Điều này giúp ổn định và tăng tốc quá trình đào tạo mạng neural. Khác với chuẩn hóa batch, các thống kê được sử dụng trong chuẩn hóa lớp là độc lập với các mẫu khác trong cùng một batch. Cụ thể, giả sử input là một vector x chứa tất cả các đặc trưng của một mẫu, phép toán được định nghĩa như sau:

$$\text{LayerNorm}(\mathbf{x}) = \boldsymbol{\alpha} \odot \frac{\mathbf{x}-\mu}{\sqrt{\sigma^2+\epsilon}} + \boldsymbol{\beta},$$

Trong đó, $\odot$ là phép nhân theo phần tử (tức là phép nhân Hadamard), μ và σ là giá trị trung bình và phương sai của x, α và β là learned scaling factors và bias terms.

Dropout: Để giảm nhẹ vấn đề overfitting trong các deep neural network, kỹ thuật 'Dropout' đã được chứng minh là hiệu quả trong nhiều kiến trúc mạng neural. Ý tưởng của dropout rất đơn giản: ngẫu nhiên 'tắt' các neural với xác suất p trong quá trình training, và sử dụng tất cả neural khi testing. Phân tích chi tiết hơn chỉ ra rằng dropout có thể được coi là một dạng học tổ hợp khi xem xét một lượng lớn mô hình chia sẻ tham số (là số mũ của số neural và đặc trưng input). Chúng tôi cũng áp dụng một lớp dropout trên embedding $\widehat{E}$.

Prediction Layer

Sau b khối self-attention, thông tin của các item trước đó đã được trích xuất một cách thích ứng và phân cấp, chúng tôi dự đoán item tiếp theo (với t item đầu tiên cho trước) dựa trên $F_t^{(b)}$. Cụ thể, chúng tôi sử dụng một lớp MF để dự đoán sự liên quan của item i:

$$r_{i,t} = \mathbf{F}_t^{(b)} \mathbf{N}_i^T$$

Trong đó $r_{i,t}$ là mức độ liên quan của item i trở thành item tiếp theo của t item đầu tiên cho trước (tức $s_1, s_2, ..., s_t$) và $\mathbf{N} \in \mathbb{R}^{|\mathcal{I}| \times d}$ là một ma trận item embedding. Do đó, $r_{i,t}$ cao đồng nghĩa với mức độ liên quan cao, và chúng tôi có thể tạo ra các đề xuất bằng cách xếp hạng các chúng.

Shared Item Embedding: Để giảm kích thước mô hình và overfitting, chúng tôi xem xét một kịch bản khác chỉ sử dụng một item embedding M:

$$r_{i,t} = \mathbf{F}_t^{(b)} \mathbf{M}_i^T$$

Lưu ý rằng $F_t^{(b)}$ có thể được biểu diễn dưới dạng một hàm phụ thuộc vào item embedding $\mathbf{M}$: $\mathbf{F}_t^{(b)} =$

$f\left(\mathbf{M}_{s_1}, \mathbf{M}_{s_2}, \ldots, \mathbf{M}_{s_t}\right)$. Một vấn đề tiềm ẩn khi sử dụng item embedding đồng nhất là tích vô hướng của chúng không thể biểu diễn được các biến đổi item không đối xứng (ví dụ: item i thường được mua sau item j, nhưng không có ngược lại), do đó các phương pháp hiện tại như FPMC thường sử dụng item embedding không đồng nhất. Tuy nhiên, mô hình của chúng tôi không có vấn đề này vì nó học một biến thể phi tuyến. Ví dụ, feed forward network có thể dễ dàng đạt được tính không đối xứng với cùng một item embedding: $\text{FFN}\left(\mathbf{M}_i\right)\mathbf{M}_j^T \neq \text{FFN}\left(\mathbf{M}_j\right)\mathbf{M}_i^T$. Thực nghiệm đã chứng minh rằng việc sử dụng chung một item embedding cải thiện đáng kể hiệu suất mô hình của chúng tôi.

Explicit User Modeling: Để cung cấp các đề xuất cá nhân hóa, các phương pháp hiện tại thường chọn một trong hai hướng tiếp cận: 1) Học một user embedding tường minh đại diện cho sở thích của user; 2) Xem xét các hành động trước đó của user và tạo ra một user embedding từ embedding của các item đã xem. Phương pháp của chúng tôi thuộc loại thứ hai, vì chúng tôi tạo ra một embedding $F_n^{(b)}$ bằng cách xem xét tất cả các hành động của user. Tuy nhiên, chúng tôi cũng có thể thêm một user embedding tường minh ở tầng cuối cùng, ví dụ thông qua phép cộng: $r_{u,i,t} = (U_u + \mathbf{F}_t^{(b)})\mathbf{M}_i^T$, trong đó U là ma trận user embedding. Tuy nhiên, thực nghiệm cho thấy việc thêm một user embedding tường minh không cải thiện hiệu suất (có lẽ vì mô hình đã xem xét tất cả các hành động của user).

Network Training

Hãy nhớ rằng chúng tôi đã chuyển đổi mỗi chuỗi hành động của user (loại bỏ hành động cuối cùng) $\left(\mathcal{S}_1^u, \mathcal{S}_2^u, \ldots, \mathcal{S}_{|\mathcal{S}^u|-1}^u\right)$ thành một chuỗi có độ dài cố định $s = \{s_1, s_2, \ldots, s_n\}$ thông qua việc lược bỏ hoặc padding item. Chúng tôi định nghĩa o_t là output dự kiến tại time step t:

$$o_t = \begin{cases} <\text{pad}> & \text{if } s_t \text{ is a padding item} \\ s_{t+1} & 1 \leq t < n \\ \mathcal{S}_{|\mathcal{S}^u|}^u & t = n \end{cases},$$

Trong đó <pad> chỉ định một padding item. Mô hình của chúng tôi nhận một chuỗi s làm input, tương ứng với chuỗi o làm output dự kiến, và chúng tôi sử dụng binary cross-entropy loss làm hàm mục tiêu:

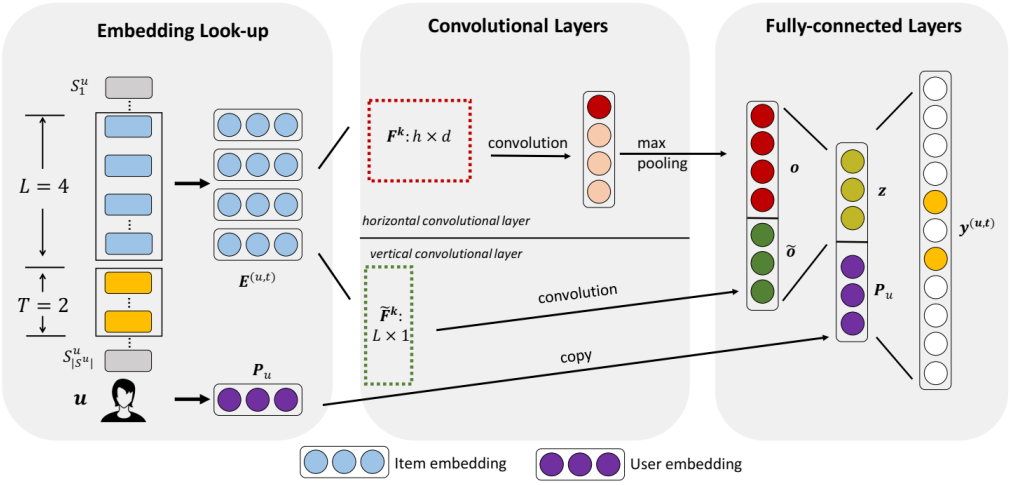
$$-\sum_{\mathcal{S}^u \in \mathcal{S}} \sum_{t \in [1,2,\ldots,n]} \left[\log\left(\sigma\left(r_{o_t,t}\right)\right) + \sum_{j \notin \mathcal{S}^u} \log\left(1 - \sigma\left(r_{j,t}\right)\right)\right].$$

Lưu ý rằng chúng tôi không chấp nhận $o_t = <\text{pad}>$.

5.3 Caser (Convolutional Sequence Embedding Recommendation Model)

Caser sử dụng các lớp convolutional để học các đặc trưng từ chuỗi thời gian của sự kiện. Điều này giúp mô hình hiểu được các mối quan hệ không chỉ giữa các sự kiện liên tiếp mà còn giữa các sự kiện không liên tiếp trong chuỗi.

Network Architecture



Các hộp chữ nhật đại diện cho các item $S_1^u, ..., S_{|S^u|}^u$ trong chuỗi hành vi của user, trong khi các hộp chữ nhật có vòng tròn bên trong đại diện cho một vector cụ thể, chẳng hạn như user embedding P_u. Các hộp chữ nhật đứt đoạn là convolutional filter với kích thước khác nhau. Các vòng tròn màu đỏ trong các convolutional filter đại diện cho giá trị lớn nhất trong mỗi kết quả convolutional. Trong hình trên, chúng tôi đang sử dụng 4 hành động trước đó ($L = 4$) để dự đoán user này sẽ tương tác với những item nào trong 2 bước tiếp theo ($T = 2$).

Embedding Look-up

Caser ghi lại các đặc tính của chuỗi trong không gian ẩn bằng cách đưa các embedding của L item trước đó vào mạng neural. Embedding $Q_i \in \mathbb{R}^d$ cho item i là một khái niệm tương tự với các hệ số ẩn của nó, trong đó d là số chiều ẩn. Thao tác embedding look-up lấy embedding của L item trước đó và xếp chồng chúng, tạo thành một ma trận $E^{(u,t)} \in \mathbb{R}^{L \times d}$ cho user u ở time step t:

$$E^{(u,t)} = \begin{bmatrix} Q_{S_{t-L}^u} \\ \vdots \\ Q_{S_{t-2}^u} \\ Q_{S_{t-1}^u} \end{bmatrix}.$$

Cùng với các item embedding, chúng ta cũng có một embedding $P_u \in \mathbb{R}^d$ cho user u, đại diện cho thói quen của user trong không gian ẩn.

Convolutional Layers

Phương pháp của chúng tôi tận dụng những thành công gần đây của các convolutional filter trong CNN để ghi lại đặc điểm cục bộ trong nhận diện hình ảnh và xử lý ngôn ngữ tự nhiên. Mượn ý tưởng sử dụng CNN trong phân loại văn bản, phương pháp của chúng tôi coi ma trận $E^{L \times d}$ như 'hình ảnh' của L item trước đó trong không gian ẩn và xem các mẫu tuần tự như đặc điểm cục bộ của 'hình ảnh'

này. Phương pháp này cho phép sử dụng convolutional filter để tìm kiếm các mẫu tuần tự. 'Horizontal filter' ghi lại các nhóm mẫu tuần tự. Các filter này được biểu diễn dưới dạng các ma trận $h \times d$, có chiều cao $h = 2$ và chiều rộng đầy đủ bằng d. Chúng thu thập tín hiệu cho các mẫu tuần tự bằng cách trượt qua các hàng của E. Tương tự, một 'vertical filter' là một ma trận $L \times 1$ và sẽ trượt qua các cột của E. Chi tiết được giải thích ở bên dưới. Không giống như việc nhận diện hình ảnh, 'hình ảnh' E không được cho trước vì embedding Q_i cho tất cả các item i phải được học đồng thời với tất cả các filter.

Horizontal Convolutional Layer: Lớp này có n horizontal filter $F^k \in \mathbb{R}^{h \times d}$, $1 \leq k \leq n$. $h \in \{1, ..., L\}$ là chiều cao của một filter. Ví dụ, nếu $L = 4$, ta có thể chọn $n = 8$ filter, 2 cho mỗi $h \in \{1, 2, 3, 4\}$. F^k sẽ trượt từ trên xuống dưới trên E và tương tác với tất cả các chiều ngang của E của các item i, $1 \leq i \leq L - h + 1$. Kết quả của sự tương tác là giá trị convolutional thứ i được cho bởi:

$$c_i^k = \phi_c \left(E_{i:t+h-1} \odot F^k \right).$$

Trong đó, kí hiệu $\odot$ biểu thị phép nhân vô hướng và $\phi_c(\cdot)$ là hàm kích hoạt cho các convolutional layer. Giá trị này là tích vô hướng giữa F^k và ma trận con được tạo ra từ hàng i đến hàng $i - h + 1$ của E, ký hiệu là $E_{i:i+h-1}$. Kết quả convolutional cuối cùng của F^k là vector:

$$c^k = \left[c_1^k c_2^k \cdots c_{L-h+1}^k \right].$$

Sau đó chúng tôi áp dụng một phép toán max pooling cho c^k để trích xuất giá trị lớn nhất từ tất cả các giá trị được tạo ra bởi filter cụ thể này. Giá trị lớn nhất giữ các đặc điểm quan trọng nhất được trích xuất bởi filter. Do đó, đối với n filter trong lớp này, giá trị output $o \in \mathbb{R}^n$ là:

$$o = \left\{ \max\left(c^1\right), \max\left(c^2\right), \cdots, \max\left(c^n\right) \right\}.$$

Horizontal filter tương tác với mỗi h item liên tiếp thông qua embedding của chúng trong ma trận E. Các embedding và các filter đều được học để giảm thiểu việc một hàm mục tiêu mã hóa sai số dự tính của các item mục tiêu (chi tiết trong phần Network Training). Bằng cách trượt các filter có chiều cao khác nhau, một tín hiệu quan trọng sẽ được thu nhận bất kể vị trí. Do đó, các horizontal filter có thể được huấn luyện để tìm các nhóm mẫu tuần tự với nhiều kích thước khác nhau.

Vertical Convolutional Layer: Chúng tôi sử dụng dấu ($\sim$) làm kí hiệu cho lớp này. Giả sử có $\widetilde{n}$ vertical filter $\widetilde{F}^k \in \mathbb{R}^{L \times 1}$, $1 \leq k \leq \widetilde{n}$. Mỗi filter $\widetilde{F}^k$ tương tác với các cột của E bằng cách trượt d lần từ trái qua phải trên E, tạo ra kết quả vertical convolutional $\widetilde{c}^k$:

$$\tilde{\boldsymbol{c}}^k = \begin{bmatrix} \overline{\boldsymbol{c}}_1^k & \overline{\boldsymbol{c}}_2^k & \cdots & \overline{\boldsymbol{c}}_d^k \end{bmatrix}.$$

Đối với tương tác tích vô hướng, ta dễ dàng thấy kết quả này bằng với tổng có trọng số qua L hàng của E với $\widetilde{F}^k$ làm trọng số:

$$\tilde{\boldsymbol{c}}^k = \textstyle\sum_{l=1}^{L} \tilde{\boldsymbol{F}}_l^k \cdot \boldsymbol{E}_l,$$

Trong đó E_l là hàng thứ l của E. Do đó, với các vertical filter, chúng ta có thể học cách tổng hợp các embedding của L item trước đó, tương tự như tổng có trọng số của Fossil để tổng hợp biểu diễn ẩn của L item trước đó. Khác ở chỗ mỗi filter $\widetilde{F}^k$ đang hoạt động như một công cụ tổng hợp khác nhau. Vì vậy, tương tự như Fossil, các vertical filter này đang ghi lại các điểm mẫu tuần tự thông qua tổng có trọng số trên biểu diễn ẩn của các item trước đó. Trong khi Fossil sử dụng một tổng có trọng số duy nhất cho mỗi user, chúng ta có thể sử dụng $\widetilde{n}$ global vertical filter để tạo ra $\widetilde{n}$ tổng có trọng số $\widetilde{o} \in \mathbb{R}^{d\,\widetilde{n}}$ cho tất cả user:

$$\tilde{o} = \left[\tilde{c}^1 \tilde{c}^2 \dots \tilde{c}^{\tilde{n}}\right].$$

Vì tác dụng của chúng là tổng hợp, vertical filter có một số khác biệt so với horizontal filter: (1) Kích thước của mỗi vertical filter được cố định là $L \times 1$. Do mỗi cột của E đều là ẩn, việc tương tác với nhiều cột liên tiếp cùng một lúc là vô nghĩa. (2) Không cần áp dụng phép toán max pooling trên kết quả vertical convolution, vì chúng tôi muốn giữ lại tổng hợp cho mỗi chiều ẩn. Vì vậy, output của layer này là $\widetilde{o}$.

Fully-connected Layers

Chúng tôi nối các output của hai lớp convolutional và đưa chúng vào một fully-connected neural network để tìm các đặc trưng cấp cao và trừu tượng hơn:

$$z = \phi_a\left(\boldsymbol{W}\begin{bmatrix}\boldsymbol{o}\\ \tilde{\boldsymbol{o}}\end{bmatrix} + \boldsymbol{b}\right),$$

Trong đó $W \in \mathbb{R}^{d\times(n+d\,\tilde{n})}$ là ma trận trọng số chiếu tầng nối lên một tầng ẩn d chiều, với $b \in \mathbb{R}^d$ là bias tương ứng và $\phi_a(\cdot)$ là hàm kích hoạt cho fully-connected layer. $z \in \mathbb{R}^d$ được gọi là convolutional sequence embedding, thứ mã hóa tất cả các loại đặc trưng tuần tự của L item trước đó.
Để ghi lại thói quen chung của user, chúng tôi cũng tra cứu user embedding P_u và nối hai vector d chiều, z và P_u lại với nhau và chiếu chúng lên một tầng output với $|\mathcal{I}|$ node, được viết là:

$$y^{(u,t)} = W'\begin{bmatrix}z\\ P_u\end{bmatrix} + \boldsymbol{b}',$$

Trong đó $b' \in \mathbb{R}^{|\mathcal{I}|}$ và $W' \in \mathbb{R}^{|\mathcal{I}|\times 2d}$ lần lượt là bias và ma trận trọng số cho output layer. Như giải thích trong phần Network Training, giá trị $y_i^{(u,t)}$ ở output layer liên quan đến xác suất user u sẽ tương tác với item i tại time step t. z ghi lại các mẫu tuần tự ngắn hạn, trong khi user embedding P_u ghi lại thói quen chung dài hạn của user. Ở đây, chúng tôi đặt user embedding P_u ở tầng ẩn cuối cùng vì vài lý do: (1) Nó có thể có khả năng tổng quát hóa cho các mô hình khác. (2) Chúng tôi có thể pre-train các tham số của mô hình chúng tôi với các tham số của các mô hình tổng quát khác. Việc pre-train như vậy có ảnh hưởng lớn đến hiệu suất mô hình.

Network Training

Để train network, chúng tôi biến đổi giá trị của output layer $y^{(u,t)}$ thành xác suất như sau:

$$p\left(\mathcal{S}_t^u \mid \mathcal{S}_{t-1}^u, \mathcal{S}_{t-2}^u, \cdots, \mathcal{S}_{t-L}^u\right) = \sigma\left(y_{\mathcal{S}_t^u}^{(u,t)}\right),$$

Trong đó, $\sigma(x) = 1/(1+e^{-x})$ là hàm sigmoid. Xem $\mathcal{C}^u = \{L+1, L+2, ..., |\mathcal{S}^u|\}$ là tập hợp các time step mà chúng tôi muốn đưa ra dự đoán cho user u. Xác suất của tất cả các chuỗi trong bộ dữ liệu là:

$$p(\mathcal{S}|\Theta) = \prod_u \prod_{t \in C^u} \sigma\left(y_{\mathcal{S}_t^u}^{(u,t)}\right) \prod_{j \neq \mathcal{S}_t^u} \left(1 - \sigma\left(y_j^{(u,t)}\right)\right).$$

Để ghi lại **skip behaviors** một cách chi tiết hơn, ta có thể xem xét T item mục tiêu tiếp theo $\mathcal{D}_t^u = \{\mathcal{S}_t^u, \mathcal{S}_{t+1}^u, ..., \mathcal{S}_{t+T}^u\}$ cùng lúc bằng cách thay thế item theo sau đó $\mathcal{S}_t^u$ trong phương trình trên bằng $\mathcal{D}_t^u$. Lấy âm logarit của xác suất chúng ta sẽ có hàm mục tiêu, còn được biết đến là binary cross-entropy loss:

$$\ell = \sum_u \sum_{t \in \mathcal{C}^u} \sum_{i \in D_t^u} -\log\left(\sigma\left(y_i^{(u,t)}\right)\right) + \sum_{j \neq i} -\log\left(1 - \sigma\left(y_j^{(u,t)}\right)\right).$$

6 Thực Nghiệm

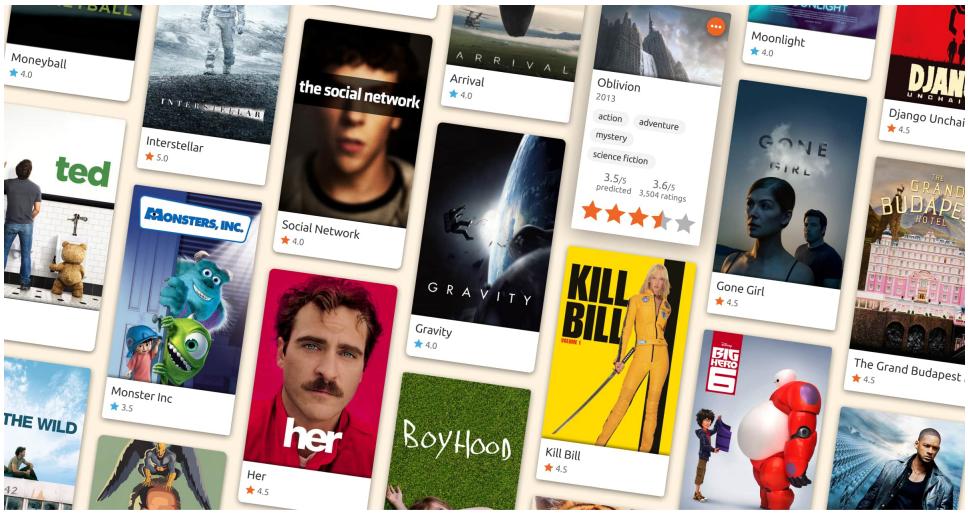
6.1 Mô tả bài toán

Ta đặt $\mathcal{U} = u_1, u_2, ..., u_{|\mathcal{U}|}$ là tập các user, $\mathcal{V} = v_1, v_2, ..., v_{|\mathcal{V}|}$ là tập các item và danh sách $\mathcal{S}_u = [v_1^{(u)}, ..., v_t^{(u)}, ..., v_{n_u}^{(u)}]$ là chuỗi tương tác theo thứ tự thời gian của user $u \in \mathcal{U}$, trong đó $v_t^{(u)} \in \mathcal{V}$ là item mà u đã tương tác tại time step t và n_u là độ dài của chuỗi tương tác của user u. Bài toán đặt ra là cho lịch sử tương tác $\mathcal{S}_u$, hệ thống đề xuất tuần tự sẽ dự đoán item mà user u sẽ tương tác tại time step $n_u + 1$. Bài toán có thể được công thức hoá như sau:

$$p(v_{n_u+1}^{(u)} = v|\mathcal{S}_u)$$

6.2 Benchmark Dataset

MovieLens



Bộ dữ liệu MovieLens là một bộ dữ liệu nổi tiếng trong lĩnh vực hệ thống đề xuất và nghiên cứu lọc cộng tác. Nó thường được sử dụng để kiểm thử và đánh giá hiệu suất của các thuật toán đề xuất khác nhau. Bộ dữ liệu này bao gồm các đánh giá phim được user thực hiện, cùng với thông tin bổ sung về các bộ phim và user.

Có nhiều phiên bản của bộ dữ liệu MovieLens với kích thước và chi tiết khác nhau. Trong bài này, chúng ta sẽ tập trung vào hai phiên bản: MovieLens 1m (**ML-1m**) and MovieLens 20m (**ML-20m**)

6.3 Input & Output

Tất cả đánh giá được chứa trong file `rating.csv`, mỗi dòng đại diện cho một đánh giá của một bộ phim bởi một người dùng và có định dạng sau:

```
userId,movieId,rating,timestamp
```

Mỗi dòng được sắp xếp theo thứ tự tăng dần của `userId`, sau đó là `movieId` theo mỗi user.

`rating` đại diện cho đánh giá trên thang điểm 5 sao, với độ chia 0.5 (0.5 - 5 sao).

`timestamp` đại diện cho số giây tính từ UTC 1970-01-01 00:00:00.

Đối với tiền xử lý dữ liệu, chúng tôi chuyển đổi tất cả các đánh giá thành 1 (tức là user có tương tác với item đó). Sau đó, chúng tôi nhóm các bản ghi tương tác theo user và xây dựng chuỗi tương tác cho mỗi user bằng cách sắp xếp các bản ghi tương tác này theo thời gian. Để đảm bảo chất lượng của bộ dữ liệu, chúng tôi chỉ giữ lại những user có ít nhất 5 đánh giá.

Do đó, input cho mô hình là một chuỗi tương tác của user bao gồm các item và timestamp tương ứng; output của mô hình sẽ là item có xác suất cao nhất mà người dùng sẽ tương tác ở time step tiếp theo.

6.4 Task Settings & Evaluation Metrics

Để đánh giá các mô hình đề xuất tuần tự, chúng tôi sử dụng phương pháp đánh giá leave-one-out (nghĩa là đề xuất item tiếp theo). Đối với mỗi user, chúng tôi giữ lại item cuối cùng của chuỗi hành vi làm dữ liệu kiểm thử, xem xét item ngay trước đó như là tập validation, và sử dụng các item còn lại để train. Để đánh giá một cách dễ dàng và công bằng, chúng tôi ghép cặp mỗi item thực tế trong tập test với 100 item ngẫu nhiên mà user chưa tương tác. Để đảm bảo việc lấy mẫu đáng tin cậy và có tính đại diện, 100 item này được lấy mẫu dựa trên độ phổ biến.

Để đánh giá hiệu suất của tất cả các mô hình, chúng tôi sử dụng nhiều thước đo đánh giá, bao gồm Hit Ratio (HR), Normalized Discounted Cumulative Gain (NDCG), và Mean Reciprocal Rank (MRR). Xem xét việc chúng ta chỉ có một item thực tế cho mỗi user, HR@k tương đương với Recall@k và tỷ lệ thuận với Precision@k; MRR tương đương với Mean Average Precision (MAP). Trong bài này, chúng tôi trình bày HR và NDCG với k = 1, 5, 10. Đối với tất cả các thước đo này, giá trị càng cao thì hiệu suất càng tốt.

6.5 Triển Khai Chi Tiết

Đối với SASRec[1] và Caser[2], chúng tôi sử dụng code được cung cấp bởi tác giả tương ứng. Đối với các siêu tham số chung trong tất cả các mô hình, chúng tôi xem xét kích thước chiều ẩn d từ $\{16, 32, 64, 128, 256\}$, hệ số điều chỉnh ℓ_2 từ $\{1, 0.1, 0.01, 0.001, 0.0001\}$ và tỷ lệ dropout từ $\{0, 0.1, 0.2, ..., 0.9\}$. Tất cả các siêu tham số khác (ví dụ như thứ tự Markov trong Caser) và các phương pháp khởi tạo đều tuân theo gợi ý từ các tác giả hoặc được điều chỉnh trên các tập validation. Chúng tôi thể hiện kết quả của mỗi mô hình dưới bộ siêu tham số tối ưu của nó.

Đối với BERT4Rec[3], chúng tôi thực hiện với TensorFlow. Tất cả tham số được khởi tạo bằng truncated normal distribution trong khoảng $[-0.02, 0.02]$. Chúng tôi train mô hình dùng Adam optimizer với learning rate $1e-4$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, ℓ_2 weight decay 0.01, và linear decay của learning rate. Đạo hàm bị cắt khi ℓ_2 norm đạt ngưỡng 5.

Để so sánh công bằng, chúng tôi cho số layer $L = 2$, head number $h = 2$ và chiều dài tối đa của chuỗi hành vi $N = 200$. Đối với cài đặt head, chúng tôi cho thực nghiệm số chiều của mỗi head là 32 (single head nếu $d < 32$). Chúng tôi điều chỉnh mask proportion ρ dùng tập validation, kết quả là $\rho = 0.2$ cho ML-1m và ML-20m. Tất cả mô hình đều được train từ đầu trên một GPU NVIDIA GeForce GTX 1080 Ti với batch size là 256.

6.6 So Sánh Hiệu Suất Tổng Thể

Bảng bên dưới tóm tắt kết quả tốt nhất của tất cả các mô hình trên hai bộ dữ liệu thử nghiệm. Chúng tôi bỏ qua kết quả nDCG@1 vì nó bằng với HR@1 trong các thực nghiệm.

[1] https://github.com/kang205/SASRec
[2] https://github.com/graytowne/caser_pytorch
[3] https://github.com/FeiSun/BERT4Rec

Datasets	Metric	Caser	SASRec	BERT4Rec
ML-1m	HR@1	0.2194	0.2351	0.2863
	HR@5	0.5353	0.5434	0.5876
	HR@10	0.6692	0.6629	0.6970
	NDCG@5	0.3832	0.3980	0.4454
	NDCG@10	0.4268	0.4368	0.4818
	MRR	0.3648	0.3790	0.4254
ML-20m	HR@1	0.1232	0.2544	0.3440
	HR@5	0.3804	0.5727	0.6323
	HR@10	0.5427	0.7136	0.7473
	NDCG@5	0.2538	0.4208	0.4967
	NDCG@10	0.3062	0.4665	0.5340
	MRR	0.2529	0.4026	0.4785

Theo kết quả, rõ ràng là BERT4Rec thể hiện tốt nhất trong 3 phương pháp trên hai bộ dữ liệu thử nghiệm đối với tất cả các evaluation metric.

7 Kết Luận Và Hướng Đi Trong Tương Lai

Việc khám phá về hệ thống đề xuất dựa trên Deep Learning, đặc biệt là trong bối cảnh đề xuất theo chuỗi, đã hé lộ những tiến bộ hứa hẹn trong việc nâng cao độ chính xác và cá nhân hóa của các hệ thống đề xuất. Bản chất chuỗi của tương tác người dùng đã được ghi lại một cách hiệu quả thông qua các mô hình như SASRec và Caser, cho phép hiểu biết sâu sắc hơn về sở thích và hành vi của người dùng theo thời gian.

Nghiên cứu đã làm nổi bật tầm quan trọng của việc sử dụng các mẫu tuần tự trong tương tác của người dùng để cải thiện độ chính xác của đề xuất, giải quyết những hạn chế của các phương pháp đề xuất truyền thống. Các mô hình Deep Learning đã chứng minh khả năng của chúng trong việc rút trích các biểu diễn ý nghĩa từ dữ liệu tuần tự, dẫn đến các đề xuất chính xác và nhạy cảm với ngữ cảnh. Việc tích hợp thời gian và sự phụ thuộc dài hạn trong quá trình đề xuất đã chứng minh được sự quan trọng trong việc thích ứng với sự thay đổi của sở thích của người dùng theo thời gian.

Mặc dù đã có những tiến triển đáng kể trong lĩnh vực các hệ thống đề xuất tuần tự dựa trên Deep Learning, vẫn còn nhiều hướng nghiên cứu và cải tiến trong tương lai. Đầu tiên, việc khám phá các mô hình kết hợp phương pháp lọc cộng tác dựa trên nội dung với Deep Learning có thể dẫn đến các hệ thống đề xuất mạnh mẽ hơn. Việc tích hợp thông tin ngữ cảnh bên ngoài, chẳng hạn như dữ liệu mạng xã hội hoặc chi tiết dân số, có thể tăng cường khả năng cá nhân hóa của các mô hình.

Ngoài ra, việc giải quyết các thách thức liên quan đến sự thiếu hụt dữ liệu và vấn đề cold start vẫn là một lĩnh vực chủ chốt cho sự nghiên cứu trong tương lai. Phát triển các kỹ thuật để xử lý các tình huống nơi tương tác của người dùng giới hạn hoặc không khả dụng là quan trọng để các hệ thống đề xuất này có thể ứng dụng thực tế. Hơn nữa, việc nghiên cứu về tính minh bạch và giải thích trong các mô hình đề xuất dựa trên Deep Learning là cần thiết để xây dựng sự tin tưởng của người dùng.

Khi công nghệ tiếp tục phát triển, khả năng mở rộng và hiệu suất của các mô hình Deep Learning trong các hệ thống đề xuất cần là một trọng tâm cho nghiên cứu trong tương lai. Tối ưu hóa quá

trình huấn luyện và triển khai mô hình để xử lý các bộ dữ liệu lớn và đề xuất trong thời gian thực sẽ đóng góp nhiều ứng dụng của hệ thống này trong thực tế.
Tổng kết lại, việc khám phá về hệ thống đề xuất dựa trên Deep Learning đã mở ra những cơ hội hứa hẹn để tạo ra các hệ thống đề xuất chính xác và cá nhân hóa hơn. Các nghiên cứu tương lai nên tập trung vào giải quyết những thách thức hiện tại, tích hợp thông tin bổ sung về ngữ cảnh, tăng cường tính minh bạch và đảm bảo khả năng mở rộng để làm cho những hệ thống này trở nên hiệu quả và áp dụng trong các tình huống thực tế.

8 Tài liệu tham khảo

[1] Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor. 2010. Recommender Systems Handbook.

[2] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer.

[3] Wang-Cheng Kang, Julian McAuley. 2018. Self-Attentive Sequential Recommendation.

[4] Jiaxi Tang, Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding.

[5] Shuai Zhang, Lina Yao, Aixin Sun, Yi Tay. 2019. Deep Learning based Recommender System: A Survey and New Perspectives.

[6] Hao Wang, Naiyan Wang, Dit-Yan Yeung. 2014. Collaborative Deep Learning for Recommender Systems.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems.

[8] Steffen Rendle, Walid Krichene, Li Zhang, John Anderson. 2020. Neural Collaborative Filtering vs. Matrix Factorization Revisited.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

[10] Diederik P. Kingma, Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization.

[11] Ruining He, Julian McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 1 (2014), 1929–1958

[13] Zhilu Zhang, Mert R. Sabuncu. 2018. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels.

[14] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

[15] Shoya Matsumori, Kohei Okuoka, Ryoichi Shibata, Minami Inoue, Yosuke Fukuchi, Michita Imai. 2022. Mask and Cloze: Automatic Open Cloze Question Generation using a Masked Language Model.