Federal State Autonomous Educational Institution for Higher Education
National Research University Higher School of Economics

Faculty of Computer Science
Educational Program
Applied Mathematics and Information Science

TERM PAPER

Programming project

"Sequential Recommender System"

Prepared by the student of group 171, 3rd year of study,
Fakanov Pavel Alexandrovich

Supervisor:
senior lecturer Sokolov Evgeny Andreevich

Co-Supervisor:
software engineer Filippovich Vadim Vyacheslavovich

Moscow 2020

Contents

1 Annotation

Recommender systems traditionally assume that user profiles and items' attributes are static. Unlike traditional recommender systems that include collaborative filtering and content-based filtering, sequential recommender systems attempt to understand and model the sequential user behaviors, the interactions between users and items, and the evolution of users' preferences and popularity of items over time. In this paper we discuss conventional approaches to building sequential recommender systems and implement some examples of them and then compare their perfomance on publicly available dataset and our internal data.

2 Аннотация

Рекомендательные системы традиционно предполагают, что профили пользователей и атрибуты объектов статичны. В отличие от традиционных рекомендательных систем, включающих коллаборативную фильтрацию и фильтрацию на основе контента, последовательные рекомендательные системы пытаются понять и смоделировать последовательное поведение пользователей, взаимодействие между пользователями и объектами рекоммендаций, а также эволюцию предпочтений пользователей и популярности элементов с течением времени. В этой статье мы обсуждаем традиционные подходы к построению последовательных рекомендательных систем и реализуем некоторые их них, а затем сравниваем их производительность на публичном наборе данных и на нашем внутреннем наборе данных.

3 Keywords

recommender systems, collaborative filtering, sequential recommendation, session-based recommendation, sequential data, deep learning

4 Introduction

As the world wide web continues to grow at an exponential rate, the size and complexity of many websites grow with it. It is becoming increasingly difficult and time-consuming for users to find the information they are looking for. User interfaces can help users find information that matches their interests by personalizing. This allows users to easily filter and find information in terms of their preferences, and also allows online platforms to widely publish the information they produce. Most traditional recommendation systems are based on content and collaborative filtering. They aim to model user preferences for elements based on explicit or implicit interactions between users and elements. In particular, they tend to use historical user interactions to study their static preferences, with the assumption that all user and element interactions in the historical sequence are equally important. However, this may not work in real-world scenarios, where the user's next behavior is not only dependent on a static long-term preference, but also largely depends on the current intent, which can probably be inferred and influenced by a small set of the most recent interactions. On the other hand, traditional approaches always ignore consideration of sequential dependencies between user interactions, which leads to inaccurate modeling of user preferences. In this case, the sequential recommendation (a. k.a. session-based) is becoming increasingly popular in academic research and practical applications. A sequential recommendation is also often referred to as a session-based,or sequence-based recommendation. For a sequential recommendation, in addition to taking into account the long-term preferences of users in different sessions, as a normal recommendation system does, it is also extremely important to simultaneously model the short-term interest of users in the session (or short sequence) for an accurate recommendation. In recent years, deep learning (DL) techniques such as recurrent neural network (RNN) and transformer architecture have made huge strides in natural language processing (NLP), demonstrating their effectiveness in processing sequential data. As such, they have attracted increasing interest in

sequential recommendation, and many DL-based models have achieved state-of-the-art results

5 Recommender systems

Over the past few decades, with the advent of Amazon, Netflix, Youtube and many other similar web services, recommendation systems have taken up more and more space in our lives. From e-Commerce (offering customers articles that might interest them) to online advertising (offering users the right content that matches their preferences), recommendation systems are now unavoidable in our daily online travel.

Huge conferences are hosted in different parts of the world annually, featuring many big companies. The RecSys conference [3] is a good example of this.

Recommendation systems are really important in some industries because they can generate huge revenue when they are effective, as well as being a way to stand out significantly from competitors. Here follows some of the most common general techniques used in recommender systems

5.1 Collaborative filtering

The main idea of collaborative filtering is to search for information or patterns using collaborative methods between multiple data sources. Data sources usually consist of users and elements such as movies, songs, etc. These data sources mainly consist of users and items that users can interact with (purchase, click, watch). The search for these patterns is performed by collecting and analyzing a huge amount of data about the behavior and activities of users, as well as items. Collaborative filtering [4] is one of the most common general methods for recommender systems being applicable within several big use-areas. Collaborative Filtering is frequently used in social networks such as Facebook, LinkedIn, mySpace, Twitter etc to effectively recommend new friends, pages, groups as well as who to follow or what to like. But also applications such as Youtube, Reddit,

Netflix etc make use of collaborative filtering. Collaborative filtering is viable in all applications where one can observe connections between a user and their registered friends or followers. But it is also widely used within e-commerce, where Amazon is a good example who popularised algorithms for item-to-item based collaborative filtering [5]

5.1.1 User-based Collaborative Filtering

The main idea of recommender systems built on user-based collaborative filtering consists in computing the similarity between users' $\{u, v\}$ profiles, s_{uv}. That is, computing a prediction for the probability of the user u liking a specific item i, consists in computing a rating based on all ratings made by users with similar profiles. All similar profiles contribute to this prediction depending on the similarity factor, s_{uv} [6, 7, 9]:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in U_i} \text{sim}(u, v) \, (r_{vi} - \bar{r}_v)}{\sum_{v \in U_i} \text{sim}(u, v)} \tag{1}$$

Where $\bar{r}_u$ is a mean rating user gave to items. More information regarding Similarity Computation and Prediction Generation can be found under 4.2.

5.1.2 Item-based Collaborative Filtering

The concept of item-based collaborative filtering applies the same idea as its User-based counterpart, but the similarity is computed between items instead of users, and is usually described as "Users who bought this also bought that"

More generally, taking an item-based approach means looking into the set, I, of items a specific user, u, has rated, using their context to compute the similarity of other items, $\{i_0, i_1, ..., i_n\}$, not in I. When the similarities and their corresponding items have been found, a prediction can be computed. [9, 10, 11]

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in I_u} \text{sim}(i, j) \, (r_{uj} - \bar{r}_j)}{\sum_{j \in I_u} \text{sim}(i, j)} \tag{2}$$

Item-based collaborative filtering is a very common approach, often used together with algorithms for matrix factorization.

5.1.3 Matrix Factorization

Matrix factorization [12] is an algebraic operation consisting in factorising a matrix M, meaning finding the matrices $\mathbf{M_1}, \mathbf{M_2}, ..., \mathbf{M_n}$ such that when they are multiplied, the resulting matrix is $\mathbf{M}$.

In collaborative filtering based recommender systems this can be used as a rather simple and very intuitive algorithm for discovering latent features. By constructing a User-Item matrix $\mathbf{M}$, where each index contains a rating r on an item i performed by a user u.

For example, by limiting the matrix to 4×4 and adding some fictitious values for ratings where $-$ symbolises yet unrated items, a matrix to be factorised might look something like:

$$M_{u,i} = \begin{pmatrix} 4 & 1 & - & 5 \\ - & - & 3 & 4 \\ 1 & 2 & - & 5 \\ 3 & - & - & 4 \end{pmatrix}$$

Matrix factorisation can be seen as the task of predicting the missing ratings meaning filling in the blanks $(-)$ in matrix. Latent features of items different users interacted with might be different types of items characteristics or might be a shared interest between some users. So for the matrix above, an assumption regarding how many latent features one wishes to find must be made. Let us assume we want to find k latent features and we have the sets U and I containing all users and all items respectively. This means finding two matrices, $\mathbf{N}(|U| \times k$ matrix) and $\mathbf{O}(k \times |I|$ matrix) which when multiplied approximating $\mathbf{M}$:

$$\mathbf{M} \approx \mathbf{N} \times \mathbf{O} = \hat{\mathbf{M}}$$

Further, using similarity and prediction computing techniques as described under 4.2, the gaps can be filled in and thereby provide really good recommendations [12]

5.2 Similarity Computation and Prediction Generation

Similarity and prediction computations compose vital parts of collaborative filtering-based recommender systems. Given a users' vector representations u and v, we can calculate their similarity by one of the ways described below and then we will be able to use this similarities for predictions generation, there are several ways of computing the similarity but some common methods are

1 Dot product.

$$s(u, v) = u \times v$$

2 Cosine-based Similarity.

$$s(u, v) = cos(u, v)$$

3 Euclidean similarity

$$s(u, v) = -1 \times ||u - v||_2$$

4 Manhattan similarity

$$s(u, v) = -1 \times ||u - v||_1$$

When the similarity computation is completed, it is time for the most important part in a collaborative filtering based recommender system; generating the output in terms of prediction. In collaborative methods to aggregate similarities such techniques as regression or the weighted average/weighted sum can be applied.

5.3 Content-Based filtering

Another well known method when implementing recommender systems is content-based filtering [13, 14]. Content-based filtering is usually used for movie recommendations or in other environments where, for example, keywords are used to describe the items of the system. Among the actors of recommender systems using content-based filtering one can find The Internet Movie Database and other popular services for movies. The methods of Content-based filtering usually use the correlation between item features and preferences from a user's profile, in contrast to the collaborative filtering approach that selects items based on the correlation between users with similar profiles. For the above to work, the system needs to deploy some learning technique [13, 14] such as Bayesian networks, clustering, decision trees, neural networks, reinforcement learning, Nearest Neighbour etc. The system uses these techniques when observing historical data of users to learn their preferences. The intention is that, after sufficient amounts of data have been observed, the system should be able to predict future behaviour of a specific user.

5.4 Hybrid Systems

A hybrid approach to implementing recommender system means designing the system so as to making use of several recommendation techniques such as for instance collaborative filtering techniques and content-based filtering, making predictions from the combined conclusions of the two. This can be done by unifying the two techniques, by adding features from one into the other or simply by running algorithms for both techniques separately and then combine the results in some way. The most common example of a hybrid based recommender system is the one used by Netflix. While an environment like Netflix is well suited for a hybrid recommender system, it does not fit everywhere. Why Netflix is considered a hybrid system:

- Collaborative Filtering: Observing the watching and browsing habits of

similar users.

- Content-Based Filtering: Observing users with equal preferences and how they rated certain movies.

In this work the results from our sequential recommender models will be further used to rank the candidates, so our system overall also can be considered as a hybrid one.

5.5 Evaluating Recommender Systems

As for any machine learning algorithm, we need to evaluate the performances of our recommender system in order to decide which algorithm fits the best. Evaluation methods for recommender systems can mainly be divided in two sets: evaluation based on well defined metrics and evaluation mainly based on human judgment and satisfaction estimation. Evaluation on well defined metrics can further be divided in two sets: online evaluation and offline evaluation. Consider these two approaches the best way to evaluate any recommender system is to test it out in the wild. In that case such metrics as high Customer Lifetime Values (CLV), CTR, CR, ROI, and QA can be considered. Techniques like A/B testing is the best since one can get actual feedback from real users. However, if that's not possible, then we have to resort to some offline evaluation. Here traditional machine learning approaches are used, we split our data for train and test subsets, it is important to note, that such type of splitting can be done in a variety of ways, for example, we can split our data by users or by time, the second one is definitely more preferable as it is the way our recommmender system is going to be used in production. Afterwards, we learn our model on training data and calculate metrics on test set. As examples of metrics that are usually used we can consider the following ones:

1 RMSE.

$$\sqrt{\frac{1}{|\mathcal{D}|}\sum_{(u,i)\in\mathcal{D}}\left(\hat{r}_{ui}-r_{ui}\right)^2} \tag{3}$$

2 MAE

$$\sqrt{\frac{1}{|\mathcal{D}|}\sum_{(u,i)\in\mathcal{D}}|\hat{r}_{ui}-r_{ui}|} \tag{4}$$

One obvious drawback of such metrics is that they have tiny correlation with the purpose of our recommender system as we are usually interested in a proper ranking of our items rather than in the prediction of rating, therefore the following ranking metrics can be used:

1 Precision

$$\frac{|R\cap P|}{|R|} \tag{5}$$

2 Recall

$$\frac{|R\cap P|}{|P|} \tag{6}$$

Meanwhile these metrics also have some disadvantages, for example, they do not care about the order of correctly predicted items, in order to deal with such a problem the following metrics are usually used: *precision at K, average precision at K, mean average precision at K, normalized discounted cumulative gain at K and hit ratio at K*. You can find further details in [8].

6 Overview of sequential recommendation

In this section, we provide a comprehensive overview of the sequential recommendation. First, we clarify the related concepts, and then formally describe the sequential recommendation tasks. Finally, we elaborate and compare the traditional ML and DL techniques for the sequential recommendation.

6.1 Concept definitions

To facilitate the understanding, we first formally define behavior object and behavior type to distinguish different user behaviors in sequential data.

Definition 2.1. behavior object refers to the items or services that a user chooses to interact with, which is usually presented as an ID of an item or a set of items. It may be also associated with other information including text descriptions, images and interaction time. For simplicity, we often use item(s) to describe behavior object(s) in the following sections.

Definition 2.2. behavior type refers to the way that a user interacts with items or services, including search, click, add-to-cart, buy, share, etc.

Given these concepts, a **behavior** can be considered as a combination of a behavior type and a behavior object, i.e., a user interacting with a behavior object by a behavior type. A **behavior trajectory** can be thus defined as a behavior sequence (or behavior session) consisting of multiple user behaviors. A typical behavior sequence is shown in Figure 1. Specifically, a behavior (a_i) is represented by a 2-tuple (c_i, o_i), i.e., a behavior type c_i and behavior object o_i. A user who generates the sequence can either be anonymous or identified by her ID. The behaviors in the sequence are sorted in time order. When a single behavior involves with multiple objects (e.g., items recorded in a shopping basket), objects within the basket may not be ordered by time, and then multiple baskets together form a behavior sequence. It should be noted that sequence and session are interchangeably used in this paper.

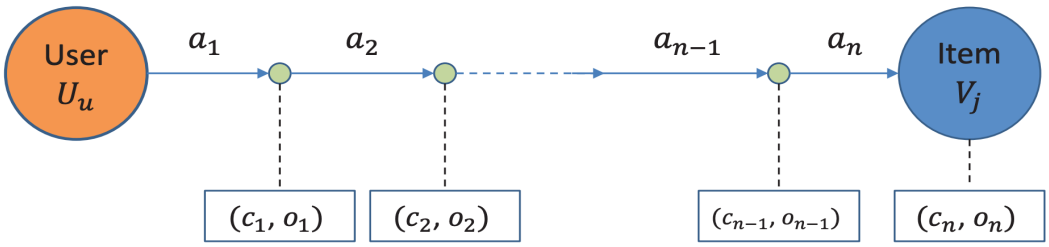

Figure 1: A schematic diagram of the sequential recommendation. c_i: behavior type, o_i: behavior object. A behavior a_i is represented by a 2-tuple, i.e., $a_i = (c_i, o_i)$. A behavior sequence (i.e., behavior trajectory) is a list of 2-tuples in the order of time.

Thus, a sequential recommender system is referred to a system which takes

a user's behavior trajectories as input, and then adopts recommendation algorithm to recommend appropriate items or services to the user. The input behavior sequence $\{a_1, a_2, a_3, ..., a_t\}$ is polymorphic, which can thus be divided into three types: experience-based, transaction-based and interaction-based behavior sequence, and the details are elaborated as follows:

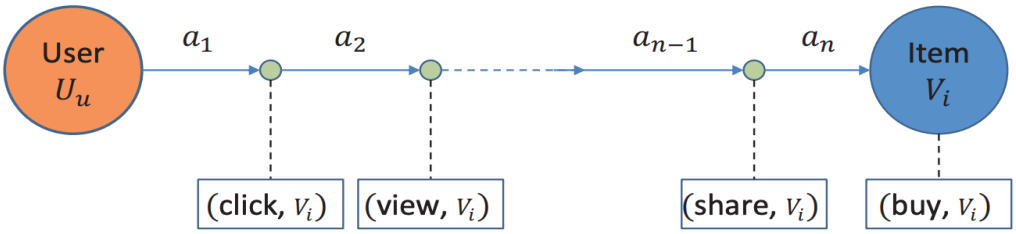

Figure 2: Experience-based behavior sequence.

Experience-based behavior sequence. In an experience-based behavior sequence, a user may interact with a same object (e.g., item $v_[i$) multiple times by different behavior types. For example, a user's interaction history with an item might be as follows: first searches related keywords, then clicks the item of interest on the result pages followed by viewing the details of the item. Finally, the user may share the item with her friends and add it to cart if she likes it. Different behavior types as well as their orders might indicate users' different intentions. For instance, click and view can only show a user's interest of a low degree, while share behavior appears before (or after) purchase might imply a user's strong desire (or satisfaction) to obtain (or have) the item. For this type of behavior sequence, a model is expected to capture a user's underlying intentions indicated by different behavior types. The goal here is to predict the next behavior type that the user will exert given an item.

Transaction-based behavior sequence. A transaction-based behavior sequence records a series of different behavior objects that a user interacts with, but with a same behavior type (i.e., buy). Therefore, with the transaction-based behavior sequence as input, the goal of a sequential recommender system is to recommend the next object (item) that a user will interact with in view of the

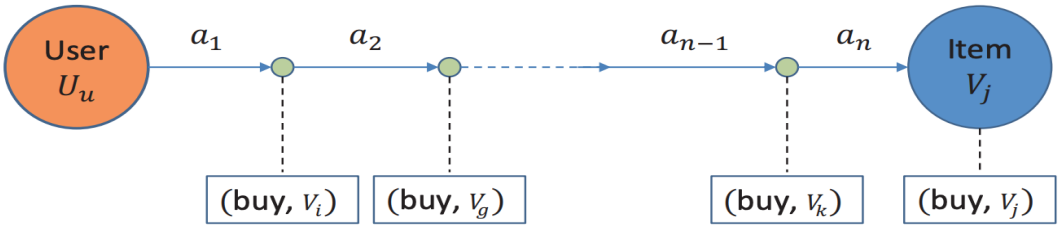

Figure 3: Transaction-based behavior sequence.

historical transactions of the user

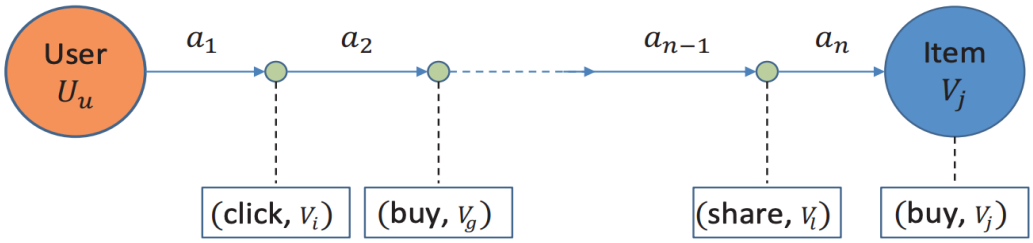

Figure 4: Interaction-based behavior sequence.

Interaction-based behavior sequence. An interaction-based behavior sequence could be viewed as a mixture of experience-based and transaction-based behavior sequences (see Fig. 4), i.e., a generalization of previous two types and much closer to the real scenarios. That is to say, it consists of different behavior objects and different behavior types simultaneously. In interaction-based behavioral sequence modeling, a recommender system is expected to understand user preferences more realistically, including different user intents expressed by different behavior types and preferences implied by different behavior objects. Its major goal is to predict the next behavior object that a user will interact with.

6.2 Sequential Recommender Tasks

According to the definition, and given the three types of behavior sequences, we thus divide the sequential recommendation tasks into three categories: experience-based sequential recommendation, transaction-based sequential recommendation, and interaction-based sequential recommendation. In this paper we will mostly consider transaction-based sequential recommendation and relevance prediction

tasks, that will be additionally described in subsections. Further information about other types of tasks is provided at [1]

6.2.1 Relevance Prediction

In some cases we model the recommendation task as Relevance prediction problem, which can be defined as follows: let $\mathcal{U} = \left\{u_1, u_2, \ldots, u_{|\mathcal{U}|}\right\}$ denote a set of users, $\mathcal{V} = \left\{v_1, v_2, \ldots, v_{|\mathcal{V}|}\right\}$ be a set of items, and list $\mathcal{S}_u = \left[v_1^{(u)}, \ldots, v_t^{(u)}, \ldots, v_{n_u}^{(u)}\right]$ denote the interaction sequence in chronological order for user $u \in \mathcal{U}$, where $v_t^{(u)} \in \mathcal{V}$ is the item that u has interacted with at time step t and n_u is the the length of interaction sequence for user u. We therefore need to learn a function, $\mathcal{F}$, to predict the probability of u liking the target item $v_{n_u+1}^{(u)}$ i.e., the candidate one.

6.2.2 Transaction-based Sequential Recommendation

In transaction-based sequential recommendation, there is only a single behavior type (transaction-related, e.g., like), and recommendation models generally consider the sequential dependency relationships between different objects (items) as well as user preferences. More formally, given the interaction history $\mathcal{S}_u$ sequential recommendation aims to predict the item that user u will interact with at time step $n_u + 1$. It can be formalized as modeling the probability over all possible items for user u at time step $n_u + 1$

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

For simplicity, we will further address this task as "Next Item Prediction Task". Additionally, as there is a substantial amount of DL-based models for this task, we further summarize the existing models in terms of the employed specific DL techniques.

6.3 Related Models

In this subsection, we first review the traditional ML methods applied to the sequential recommendation and also briefly discuss their advantages and disadvantages. Second, we summarize related DL techniques for the sequential recommendation and elaborate how they overcome the issues involved in traditional methods.

6.3.1 Traditional Methods

Conventional popular methods for the sequential recommendation include frequent pattern mining, K-nearest neighbors, Markov chains, matrix factorization, and reinforcement learning [15]. They generally adopt matrix factorization for addressing users' longterm preferences across different sequences, whilst use first-order Markov Chains for capturing users' short-term interest within a sequence [16]. We next introduce the traditional methods as well as the representative algorithms for the sequential recommendation.

Frequent pattern mining. As we know, association rule strives to use frequent pattern mining to mine frequent patterns with sufficient support and confidence. In the sequential recommendation, patterns refer to the sets of items which are frequently co-occurred within a sequence, and then are deployed to make recommendations. Although these approaches are easy to implement, and relatively explicable for users, they suffer from the limited scalability problem as matching patterns for recommendation is extremely strict and time-consuming. Besides, determining suitable thresholds for support and confidence is also challenging, where a low minimum support or confidence value will lead to too many identified patterns, while a large value will merely mine co-occurred items with very high frequency, resulting in that only few items can be recommended or few users could get effective recommendation.

K-nearest neighbors (KNN). It includes item-based KNN and session-based KNN for the sequential recommendation. Item-based KNN only considers

the last behavior in a given session and recommends items that are most similar to its behavior object (item), where the similarities are usually calculated via the cosine similarity or other advanced measurements. In contrast, session-based KNN compares the entire existing session with all the past sessions to recommend items via calculating similarities using Jaccard index or cosine similarity on binary vectors over the item space. KNN methods can generate highly explainable recommendation. Besides, as the similarities can be pre-calculated, KNN-based recommender systems could generate recommendations promptly. However, this kind of algorithms generally fails to consider the sequential dependency among items.

Markov chains (MC). In the sequential recommendation, Markov models assume that future user behaviors only depend on the last or last few behaviors. For example, merely considered the last behavior with first-order MC, while adopted high-order MCs, which take the dependencies with more previous behaviors into account. Considering only the last behavior or several behaviors makes the MC-based models unable to leverage the dependencies among behaviors in a relatively long sequence and thus fails to capture intricate dynamics of more complex scenarios. Besides, they might also suffer from data sparsity problems.

Factorization-based methods. Matrix factorization (MF) tries to decompose the user-item interaction matrix into two low-rank matrices. For example, BPR-MF optimizes a pairwise ranking objective function via stochastic gradient descent (SGD). Twardowski proposed a MF-based sequential recommender system (a simplified version of Factorization Machines), where only the interaction between a session and a candidate item is considered for generating recommendations. FPMC is a representative baseline for next-basket recommendation, which integrates the MF with first-order MCs. FISM conducts matrix factorization on an item-item matrix, and thus no explicit user representation is learned. On the basis of FISM, FOSSIL tackles the sequential recommendation task by combining similarity-based methods and highorder Markov Chains. It performs better on sparse datasets in comparison with the traditional MC methods and FPMC. The

main drawbacks of MF-based methods lie in: 1) most of them only consider the low-order interactions (i.e., first-order and second-order) among latent factors, but ignore the possible high-order interactions; and 2) excepts for a handful of algorithms considering temporal information (e.g., TimeSVD++), they generally ignore the time dependency among behaviors both within a session and across different sessions.

Reinforcement learning (RL). The essence of RL methods is to update recommendations according to the interactions between users and the recommender systems. When a system recommends an item to a user, a positive reward is assigned if the user expresses her interest on the item (via behaviors such as click or view). It is usually formulated as a Markov decision process (MDP) with the goal of maximizing the cumulative rewards in a set of interactions. With RL frameworks, sequential recommender systems can dynamically adapt to users (changing) preferences. However, similar to DL-based approaches, this kind of works is also lack of interpretability. Besides, more importantly, there is few appropriate platforms or resources for developing and testing RL-based methods in academia.

6.3.2 Deep Learning Techniques

In this subsection, we summarize the DL models (e.g., RNN and CNN) that have been adopted in the sequential recommendation in the literature.

Recurrent neural networks (RNNs). Recurrent Neural Networks have been devised to model variable-length sequence data. The main difference between RNNs and conventional feedforward deep models is the existence of an internal hidden state in the units that compose the network. Standard RNNs update their hidden state h using the following update function:

$$\mathbf{h_t} = g\left(W\mathbf{x_t} + U\mathbf{h_{t-1}}\right)$$

The effectiveness of RNNs in sequence modeling have been widely demon-

strated in the field of natural language processing (NLP). In the sequential recommendation, RNN-based models are in the majority of DL-based models. In comparison with the traditional models, RNN-based sequential recommendation models can well capture the dependencies among items within a session or across different sessions. The main limitation of RNNs for the sequential recommendation is that it is relatively difficult to model dependencies in a longer sequence (although could be somehow mitigated by other techniques), and training is burdened with the high cost especially with the increase of sequence length.

Convolutional neural networks (CNNs). CNN is commonly applied to process time series data (e.g., signals) and image data, where a typical structure consists of convolution layers, pooling layers, and feed-forward full-connected layers. It is suitable to capture the dependent relationship across local information (e.g., the correlation between pixels in a certain part of an image or the dependencies between several adjacent words in a sentence). In the sequential recommendation, CNN-based models can well capture local features within a session, and also could take the time information into consideration in the input layer.

Multi-layer perceptrons (MLPs). MLPs refer to feed-forward neural networks with multiple hidden layers, which can thus well learn the nonlinear relationship between the input and output via nonlinear activation functions (e.g., tanh and ReLU). Therefore, MLP-based sequential recommendation models are expected to well capture the complex and nonlinear relationships among users behaviors. Attention mechanisms. Attention mechanism in deep learning is intuited from visual attentions of human-beings (incline to be attracted by more important parts of a target object). It is originated from the work of Bahdanau et al., which proposes an attention mechanism in neural machine translation task to focus on modeling the importance of different parts of the input sentence on the output word. Grounded on the work, vanilla attention is proposed by applying the work as a decoder of the RNN, and has been widely used in the sequential recommendation. On the other hand, self-attention mechanism (originated in transformer for neural machine translation by Google 2017) has also been deployed in the sequen-

tial recommendation. In contrast with vanilla attention, it does not include RNN structures, but performs much better than RNN-based models in recommender systems.

7 Recommendations with RNNs

We used the LSTM-based RNN as our baseline model for our recommendations. The core of the network is the LSTM layer(s) and additional feed forward layers can be added between the last layer and the output. When multiple LSTM layers are used, the hidden state of the previous layer is the input of the next one. We use different architectures depending on the type of task we are trying to solve. In one case, we try to predict the relevance of a given candidate item to a user, given it's interactions history, while this task is being considered, the input of the network is the actual state of the user and a candidate item. The state of the user can either be the item of the actual event or the events user has interacted so far. In our case we limit the number of events and consider only last N items user has interacted with. In another case, we deal with transaction-based sequential recommendation, where we try to predict the next item user will interact with. In next subsections we will in details describe the differences of these two approaches.

7.1 Relevance prediction

In this section we describe how our model can be applied for solving Relevance Prediction Task. Generally, we use LSTM-based RNN to build user embedding, given its state (list of items user previously interacted with). That can afterwards be used to calculate the similarity with a candidate item embedding by any of similarity functions described at 4.2.

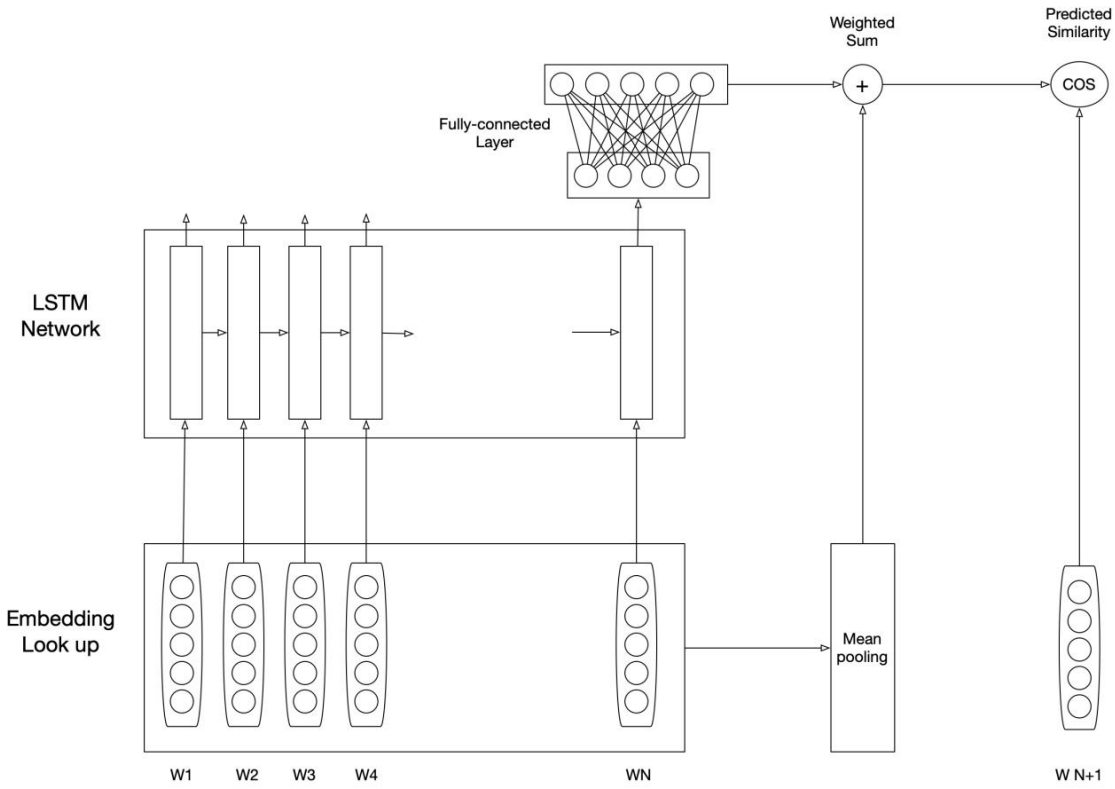

Figure 5: LSTM-based model architecture for relevance prediction

7.1.1 Model architecture

Given a sequence of items from user interaction history we process them with one layer LSTM network and then process the last hidden state with fully-connected layer, additionally we calculate the mean of history embeddings, therefore emb_{user} is a weighted sum of mean of history embeddings and linear layer over the last hidden state from LSTM.

$$emb_{user} = \alpha(LSTM(w_1, w_2, \cdots, w_N)W^P + b^P) + \beta mean(w_1, w_2, \cdots, w_N) \quad (7)$$

where W^P is a learnable projection matrix, b^P is a bias term, α, β are also learnable parameters, by $w_1, w_2, \cdots, w_N$ we mean the embeddings of corresponding items

7.1.2 Model Learning

Given a set of users with their interactions history, we can create a set of positive and negative examples. There is a variety of options that can be used to sample positive and negative examples. We will discuss them in depth in section 8.

We model relevance prediction as a binary-classification problem. We train our model in a supervised manner by minimizeing logloss error of the relevance classification. Additionaly, to prevent overfitting we regularize outputs from Embedding layer. The loss function is given below:

$$\mathcal{L} = -\frac{1}{|\mathcal{D}|} \sum_{x,y \in |D|} (y \log p(x) + (1-y) \log(1-p(x))) + \lambda \sum_{i=1}^{N} ||x_i||^2 \qquad (8)$$

where $\mathcal{D}$ represent all the samples, x_i is the embedding of i-th item in input sequence and $y \in 0, 1$ is the label representing whether the item is relevant to user or not, $p(x)$ is the output of the network after the cosine similarity function, representing the predicted probability of item being relevant.

7.2 Next item prediction

7.2.1 Model architecture

The overall architecture is quite simillar to a previous one, the only difference is that in order to get probability distribution over vocabulary tokens we apply an additional fully-connected layer with softmax activation, therefore the output of the model is the distribution over all tokens:

$$softmax((\alpha(LSTM(w_1, w_2, \cdots, w_N)W^P + b^P) + \beta mean(w_1, w_2, \cdots, w_N))W^{P'} + b^{P'}) \qquad (9)$$

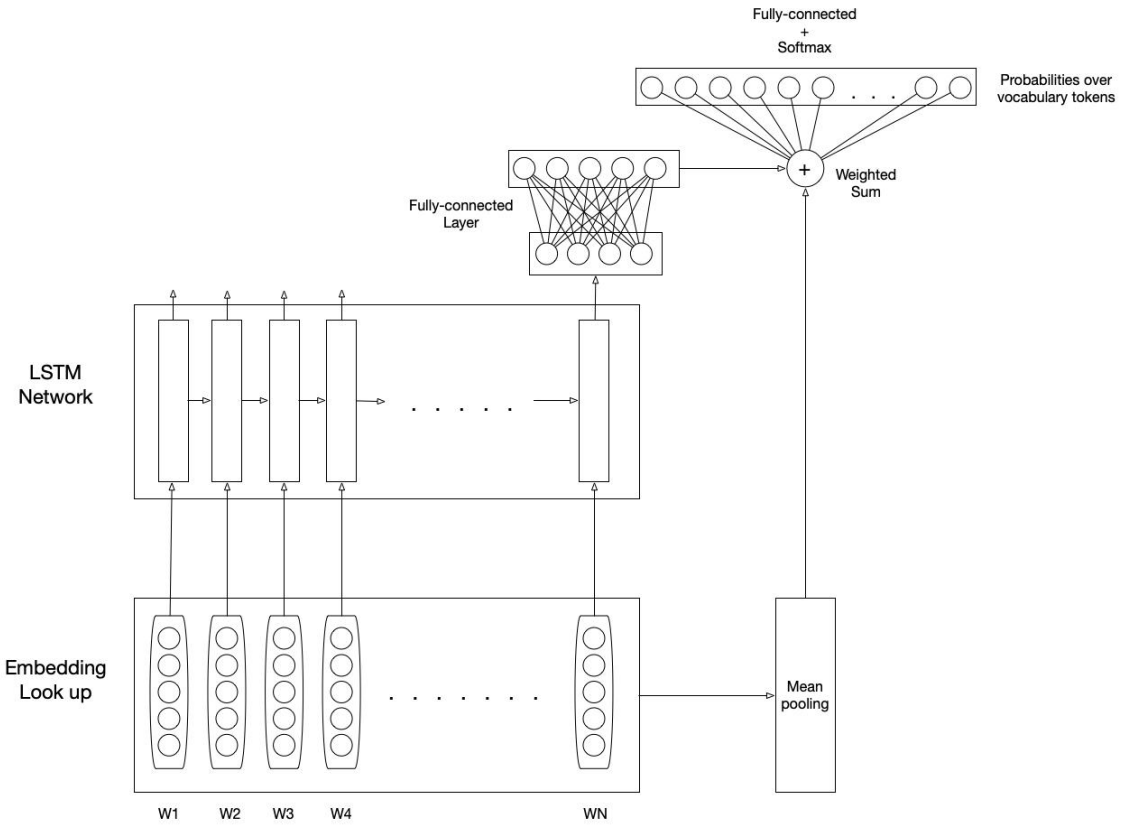

Figure 6: LSTM-based model architecture for next item prediction

7.2.2 Model Learning

Conventional unidirectional sequential recommendation models usually train the model by predicting the next item for each position in the input sequence. For simplicity we consider a model that predicts only the last item in a user interaction history.

Oppositely to relevance prediction task, we model next item prediction as a multi-class classification task. We train our model by minimizing the cross-entropy error. Similarly to previous task we add regularization to prevent overfitting. The loss function is given below:

$$\mathcal{L} = -\frac{1}{|\mathcal{D}|} \sum_{x, v^x_{n_x+1} \in |D|} \sum_{v \in \mathcal{V}} [v = v^x_{n_x+1}] \log p(v|x) + \lambda \sum_{i=1}^{N} ||x_i||^2 \tag{10}$$

where $\mathcal{D}$ represent all the samples, and $v_{v^x_{n_x+1}}$ is the last item user has interacted with, $p(v|x)$ is the output of the network after the softmax layer, representing the predicted probability of item v being relevant.

8 BERT4Rec

Here we describe a sequential recommender model BERT4Rec [2], which adopts Bidirectional Encoder Representations from Transformers to a new task, sequential Recommendation. It is built upon the popular self-attention layer, "Transformer layer"

8.1 Model Architecture

As illustrated in Figure 7, BERT4Rec is stacked by L bidirectional Transformer layers. At each layer, it iteratively revises the representation of every position by exchanging information across all positions at the previous layer in parallel with the Transformer layer. Instead of learning to pass relevant information forward step by step as RNN based methods did, self-attention mechanism endows BERT4Rec with the capability to directly capture the dependencies in any distances. This mechanism results in a global receptive field, while CNN based methods like Caser usually have a limited receptive field. In addition, in contrast to RNN based methods, self-attention is straightforward to parallelize.

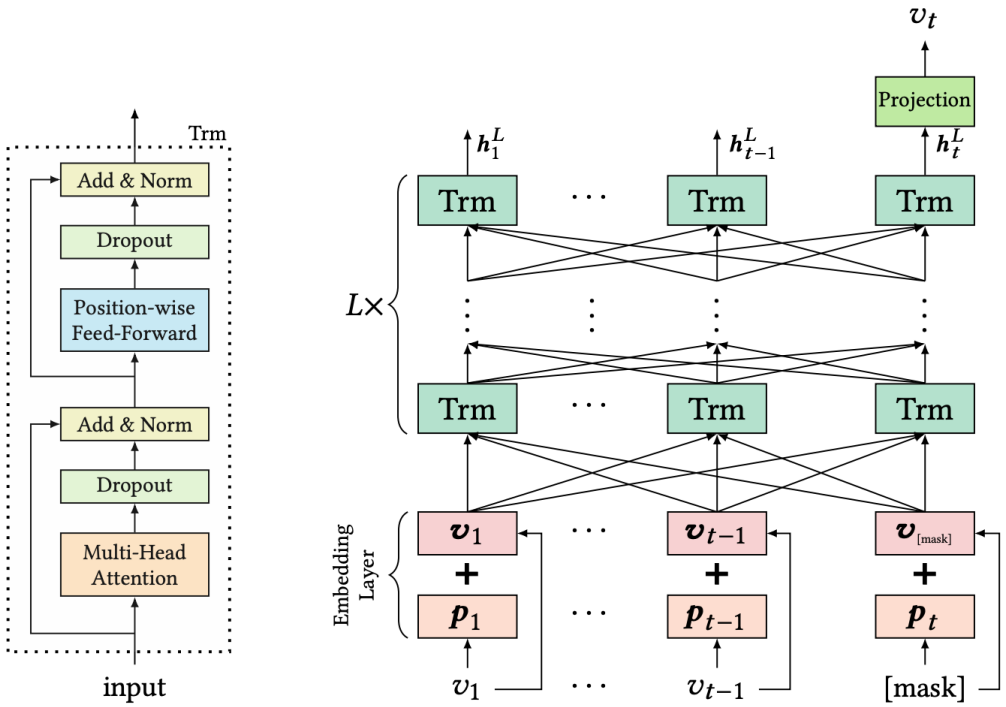

Figure 7: Transformer layer and BERT4Rec model architecture

8.2 Transformer Layer

As illustrated in Figure 7, given an input sequence of length t, we iteratively compute hidden representations h_i^l at each layer l for each position i simultaneously by applying the Transformer layer from [17]. Here, we stack $h_i^l \in \mathbb{R}^d$ together into matrix $H^l \in \mathbb{R}^{t \times d}$ since we compute attention function on all positions simultaneously in practice. As shown in Figure 7, the Transformer layer `Trm` contains two sub-layers, a *Multi-Head Self-Attention* sub-layer and a *Position-wise Feed-Forward Network*.

Multi-Head Self-Attention. Attention mechanisms have become an integral part of sequence modeling in a variety of tasks, allowing capturing the dependencies between representation pairs without regard to their distance in the sequences. Previous work has shown that it is beneficial to jointly attend to information from different representation subspaces at different positions [17, 19, 18]. Thus, we here adopt the multi-head self-attention instead of performing a single attention function. Specifically, multi-head attention first linearly projects H^l into h subspaces, with different, learnable linear projections, and then apply h attention functions in parallel to produce the output representations which are concatenated and once again projected:

$$\begin{aligned} \texttt{MH}(H^l) &= [\text{head}_1; \text{head}_2; \ldots; \text{head}_h] W^O \\ \text{head}_i &= \texttt{Attention}\big(H^l W_i^Q, H^l W_i^K, H^l W_i^V\big) \end{aligned} \tag{11}$$

where the projections matrices for each head $W_i^Q \in \mathbb{R}^{d \times d/h}$, $W_i^K \in \mathbb{R}^{d \times d/h}$, $W_i^V \in \mathbb{R}^{d \times d/h}$, and $W_i^O \in \mathbb{R}^{d \times d}$ are learnable parameters. Here, we omit the layer subscript l for the sake of simplicity. In fact, these projection parameters are not shared across the layers. Here, the `Attention` function is *Scaled Dot-Product Attention*:

$$\texttt{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d/h}}\right)V \tag{12}$$

where *query* Q, *key* K, and *value* V are projected from the same matrix H^l with

different learned projection matrices as in Equation 11. The *temperature* $\sqrt{d/h}$ is introduced to produce a softer attention distribution for avoiding extremely small gradients [20, 17].

Position-wise Feed-Forward Network. As described above, the self-attention sub-layer is mainly based on linear projections. To endow the model with nonlinearity and interactions between different dimensions, we apply a *Position-wise Feed-Forward Network* to the outputs of the self-attention sub-layer, separately and identically at each position. It consists of two affine transformations with a Gaussian Error Linear Unit (`GELU`) activation in between:

$$\begin{aligned} \texttt{PFFN}(H^l) &= \left[\texttt{FFN}(h_1^l)^\top; \ldots; \texttt{FFN}(h_t^l)^\top\right]^\top \\ \texttt{FFN}(x) &= \texttt{GELU}\big(xW^{(1)} + b^{(1)}\big)W^{(2)} + b^{(2)} \\ \texttt{GELU}(x) &= x\Phi(x) \end{aligned} \tag{13}$$

where $\Phi(x)$ is the cumulative distribution function of the standard gaussian distribution, $W^{(1)} \in \mathbb{R}^{d\times 4d}$, $W^{(2)} \in \mathbb{R}^{4d\times d}$, $b^{(1)} \in \mathbb{R}^{4d}$ and $b^{(2)} \in \mathbb{R}^{d}$ are learnable parameters and shared across all positions. We omit the layer subscript l for convenience. In fact, these parameters are different from layer to layer. In this work, following OpenAI GPT and BERT [18], we use a smoother `GELU` activation rather than the standard `ReLu` activation

Stacking Transformer Layer. As elaborated above, we can easily capture item-item interactions across the entire user behavior sequence using self-attention mechanism. Nevertheless, it is usually beneficial to learn more complex item transition patterns by stacking the self-attention layers. However, the network becomes more difficult to train as it goes deeper. Therefore, we employ a residual connection around each of the two sub-layers as in Figure 7, followed by layer normalization. Moreover, we also apply dropout to the output of each sub-layer, before it is normalized. That is, the output of each sub-layer is $\texttt{LN}(x + \texttt{Dropout}(\text{sublayer}(x)))$, where $\text{sublayer}(\cdot)$ is the function implemented by the sub-layer itself, `LN` is the layer normalization function defined in. We use `LN`

to normalize the inputs over all the hidden units in the same layer for stabilizing and accelerating the network training.

In summary, BERT4Rec refines the hidden representations of each layer as follows:

$$H^l = \texttt{Trm}\big(H^{l-1}\big), \quad \forall i \in [1, \ldots, L] \tag{14}$$

$$\texttt{Trm}(H^{l-1}) = \texttt{LN}\Big(A^{l-1} + \texttt{Dropout}\big(\texttt{PFFN}(A^{l-1})\big)\Big) \tag{15}$$

$$A^{l-1} = \texttt{LN}\Big(H^{l-1} + \texttt{Dropout}\big(\texttt{MH}(H^{l-1})\big)\Big) \tag{16}$$

8.3 Embedding Layer

As elaborated above, without any recurrence or convolution module, the Transformer layer `Trm` is not aware of the order of the input sequence. In order to make use of the sequential information of the input, we inject *Positional Embeddings* into the input item embeddings at the bottoms of the Transformer layer stacks. For a given item v_i, its input representation h_i^0 is constructed by summing the corresponding item and positional embedding:

$$h_i^0 = v_i + p_i$$

where $v_i \in E$ is the $d-$dimensional embedding for item v_i, $p_i \in P$ is the $d-$dimensional positional embedding for position index i. In this work, we use the learnable positional embeddings instead of the fixed sinusoid embeddings in [17] for better performances. The positional embedding matrix $P \in \mathbb{R}^{N \times d}$ allows our model to identify which portion of the input it is dealing with. However, it also imposes a restriction on the maximum sentence length N that our model can handle. Thus, we need to truncate the the input sequence $[v_1, \ldots, v_t]$ to the last N items $[v_{t-N+1}^u, \ldots, v_t]$ if $t > N$.

8.4 Output Layer

Depending on what type of problem we are dealing with we use different output layers. Here we describe output layers architectures for "Relevance prediction" and "Next item prediction" problems.

8.4.1 Relevance prediction

Similarly to RNN-based model, we want to build a user embedding given its history $[v^u_{t-N+1}, \ldots, v^u_t]$. After L layers that hierarchically exchange information across all positions in the previous layer, we get the final output H^L for all items of the input sequence, then we take the average of the final hidden state of encoding layer on the time axis and apply a two-layer feed-forward network with RELU activation in between, resulting in a user embedding:

$$emb_{user} = RELU(mean(H^L)W^P + b^P)W^{P'} + b^{P'} \tag{17}$$

where $W^P, W^{P'}$ are learnable projection matrixes, b^P, and $b^{P'}$ are bias terms. After that the relevance is calculated as simillarity between emb_{user} and emb_{item}, that is taken from embedding matrix $E \in \mathbb{R}^{|\mathcal{V}| \times d}$ for the item set $\mathcal{V}$:

$$score = cos(emb_{user}, emb_{item}) \tag{18}$$

Other simillarity metrics described at **4.2** also can be used for relevance prediction

8.4.2 Next item prediction

While solving a problem of next item prediction we produce the probability distribution over set of items the following way: assuming that we mask the item v_t at time step t, we then predict the masked items v_t base on h^L_t as shown in Figure 7. Specifically, we apply a two-layer feed-forward network with `GELU`

activation in between to produce an output distribution over target items:

$$P(v) = \text{softmax}\big(\texttt{GELU}(h_t^L W^P + b^P)E^\top + b^O\big) \tag{19}$$

where W^P is the learnable projection matrix, b^P, and b^O are bias terms, $E \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the embedding matrix for the item set $\mathcal{V}$. We use the shared item embedding matrix in the input and output layer for alleviating overfitting and reducing model size.

8.5 Model Learning

8.5.1 Relevance prediction

For relevance prediction we used similar approach as for LSTM-based model, we randomly sampled negative candidates and solved this task as binary classification problem. We tried both approaches with and without regularization, unfortunately, none of them resulted in competitive quality.

8.5.2 Next item prediction

Training. Conventional unidirectional sequential recommendation models usually train the model by predicting the next item for each position in the input sequence. Specifically, the target of the input sequence $[v_1, \ldots, v_t]$ is a shifted version $[v_2, \ldots, v_{t+1}]$. However, as shown in Figure 7, jointly conditioning on both left and right context in a bidirectional model would cause the final output representation of each item to contain the information of the target item. This makes predicting the future become trivial and the network would not learn anything useful. A simple solution for this issue is to create $t - 1$ samples (subsequences with next items like ($[v_1]$, v_2) and ($[v_1, v_2]$, v_3)) from the original length t behavior sequence and then encode each historical subsequence with the bidirectional model to predict the target item. However, this approach is very time and resources consuming since we need to create a new sample for each position in the

sequence and predict them separately.

In order to efficiently train our proposed model, we apply a new objective: *Cloze* task (also known as "Masked Language Model" in [18]) to sequential recommendation. It is a test consisting of a portion of language with some words removed, where the participant is asked to fill the missing words. In our case, for each training step, we randomly mask ρ proportion of all items in the input sequence (*i.e.*, replace with special token "`[mask]`"), and then predict the original ids of the masked items based solely on its left and right context. For example:

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

The final hidden vectors corresponding to "`[mask]`" are fed into an output softmax over the item set, as in conventional sequential recommendation. Eventually, we define the loss for each masked input $\mathcal{S}'_u$ as the negative log-likelihood of the masked targets:

$$\mathcal{L} = \frac{1}{|\mathcal{S}_u^m|} \sum_{v_m \in \mathcal{S}_u^m} -\log P(v_m = v_m^* | \mathcal{S}'_u) \tag{20}$$

where $\mathcal{S}'_u$ is the masked version for user behavior history $\mathcal{S}_u$, $\mathcal{S}_u^m$ is the random masked items in it, v_m^* is the true item for the masked item v_m, and the probability $P(\cdot)$ is defined in Equation 19.

An additional advantage for Cloze task is that it can generate more samples to train the model. Assuming a sequence of length n, conventional sequential predictions produce n unique samples for training, while BERT4Rec can obtain $\binom{n}{k}$ samples (if we randomly mask k items) in multiple epochs. It allows us to train a more powerful bidirectional representation model.

9 Experiments

9.1 Datasets

We evaluate our models on a subset of Yandex internal dataset of movies, additionally we provide results of some of our models for MovieLens dataset, that is a popular benchmark dataset for evaluating recommendation algorithms. In this work, we adopt MovieLens 1m (**ML-1m**) for simplicity and due to limited resources.

For ML-1m dataset we follow the common practice. We convert all numeric ratings to implicit feedback of 1 (*i.e.*, the user interacted with the item). After that, we group the interaction records by users and build the interaction sequence for each user by sorting these interaction records according to the timestamps. To ensure the quality of the dataset, following the common practice, we keep users with at least five feedbacks. It is important to note, that we used ML-1m dataset to compare the perfomance of our LSTM-based model with more advanced techniques and provide the information about the applicability of our model on third-party data.

In contrast, for our internal dataset we ended up with an approach, where we provide our models only interactions that can be considered as positive ones, in other words, we use only data where the rating of film exceeds 3.5 out of 5, additionally we filter the history of user by leaving only films user rated as good ones. Therefore, our models given a sequence of items a user liked will try to predict a next item user will like with a high probability, that highly correlates with business purposes of our service.

The further preprocessing depends on what type of task we are trying to solve. In one case (relevance prediction) we also have to provide our model with negative examples. This can be done in a variety of ways, we used a straightforward approach: given a user and its interactions history we sample a random item $v \in \mathcal{V}$ that due to sparcity of our data with a high probability can be considered as a negative candidate. Finally, we have a balanced dataset of positive and

Table 1: Statistics of datasets.

Datasets	#users	#items	#actions	Avg. length	Density
ML-1m	6040	3416	1.0m	163.5	4.79%
Yandex films	63727	25654	0.94m	14.7	0.0573%

negative examples.

The statistics of the processed datasets are summarized in Table 1.

9.2 Task Settings And Evaluation Metrics

Relevance Prediction. Such type of model can be applied in two various ways. In one case, we can score all the items and rank them according to the predicted score given by our model, unfortunately, this approach is not really applicable, as the inference phase takes to long. In another case, we can use scores of our models as some extra information only for candidate items that have been choosen by a simpler and less time-consuming model (usually collaborative), that will allow us to make use of sequential requlatities in our data.

Next item prediction To evaluate the sequential recommendation models, we adopted the *leave-one-out* evaluation (*i.e.*, next item recommendation) task, which has been widely used in. For each user, we hold out the last item of the behavior sequence as the test data, and utilize the remaining items for training. For easy and fair evaluation, we follow the common strategy, pairing each ground truth item in the test set with 100 randomly sampled *negative* items that the user has not interacted with. To make the sampling reliable and representative, these 100 negative items are sampled according to their popularity. Hence, the task becomes to rank these negative items with the ground truth item for each user.

Evaluation Metrics. For relevance prediction model we use a standard classification metric, such as accuracy due to balance in classes. To evaluate the ranking list of models for next item prediction, we employ a variety of evaluation metrics, including *Hit Ratio* (HR) and *Normalized Discounted Cumulative Gain* (NDCG). In this work, we report HR and NDCG with $k = 10$. For all these

metrics, the higher the value, the better the performance.

9.3 Implementation details

9.3.1 LSTM-based model

We trained our model using Adam with learning rate of 3e-3, $\beta_1 = 0.9, \beta_2 = 0.999$ with regularization coefficient of 0.01. We trained all of our models for 5000 steps with a batch size of 5120. Also we experimented with more layers of Long short-term memory network, that didn't reflect in any metrics boost, but caused longer inference time.

9.3.2 BERT4Rec

We used BERT4Rec[1] implemented with `TensorFlow`. All parameters are initialized using truncated normal distribution in the range $[-0.02, 0.02]$. We train the model using Adam with learning rate of 1e-4, $\beta_1 = 0.9, \beta_2 = 0.999$, ℓ_2 weight decay of 0.01, and linear decay of the learning rate. The gradient is clipped when its ℓ_2 norm exceeds a threshold of 5. We set the layer number $L = 2$ and head number $h = 2$ and experimented with two values of maximum sequence length $N = 10$, that can be applied for session predictons, and $N = 50$, for getting more information about user previous history of interactions. For head setting, we empirically set the dimensionality of each head as 32 (single head if $d < 32$). All the models were trained from scratch without any pre-training with a batch size of 256

9.4 Impact of hidden dimensionality

We now study how the hidden dimensionality d affect the recommendation performance for next item recommendation task. Figure 8 shows NDCG@10 and HR@10 for neural sequential methods with the hidden dimensionality d varying from 64 to 256 while keeping other optimal hyper-parameters unchanged.

[1] https://github.com/FeiSun/BERT4Rec

9.5 LSTM-based model

For LSTM-based model we provide the results for configuration with sequence length N equals to 5, that ended up giving the best perfomance, sllightly different from models with N equals to 10, 50. Additionally we provide results of our model for ML-1m dataset:

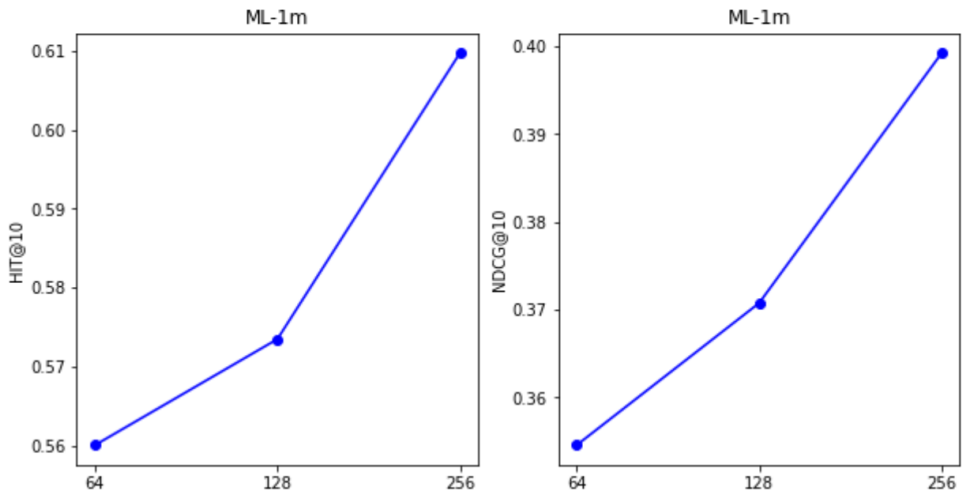

Figure 8: **Effect of the hidden dimensionality d on HR@10 and NDCG@10 for ML-1m dataset for LSTM-based model.**

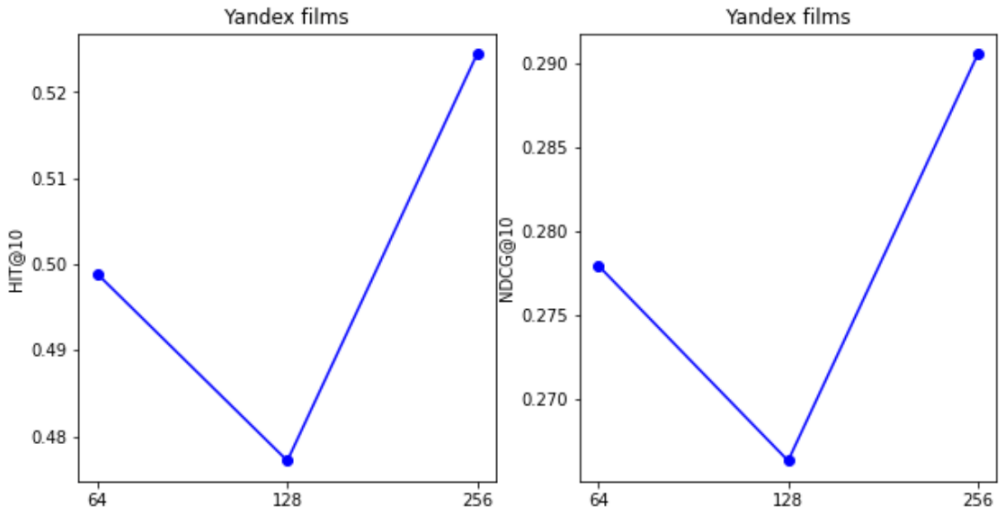

Figure 9: **Effect of the hidden dimensionality d on HR@10 and NDCG@10 for Yandex films dataset for LSTM-based model.**

We can see from these figures that our model provides competitive results with simillar architecture GRU4Rec [21] for Next Item prediction task. Moreover, we can see that for MovieLens dataset an increase in hidden dimensionality causes

significant increase in ranking metrics, in contrast, for our dataset we can't notice a monotonous dependency of ranking metrics on dimension size.

9.6 BERT4Rec

For BERT4Rec arhcitecture we additionally provide the dependency on sequence length N, oppositely to LSTM-based model we can see some noticeble differences.

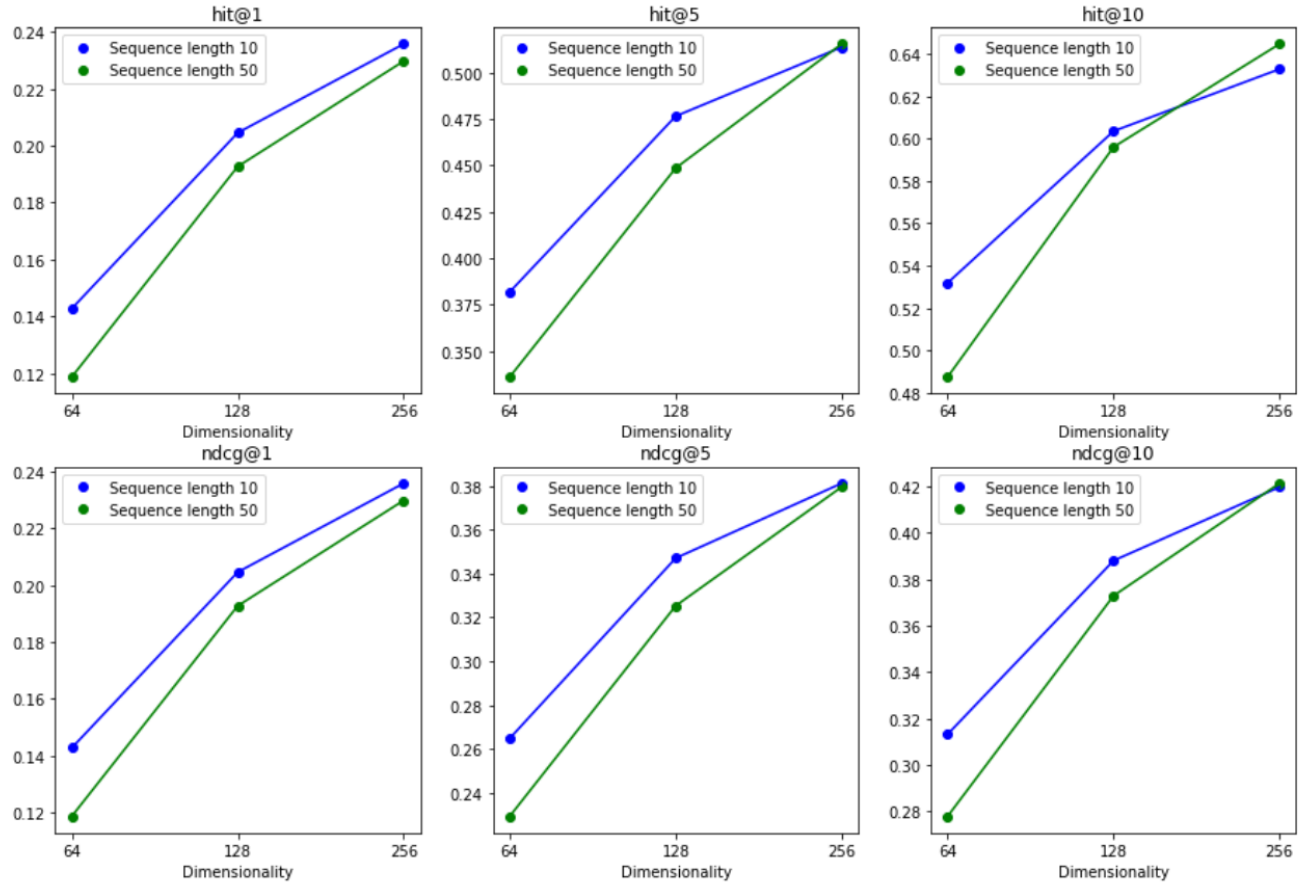

Figure 10: **Effect of the hidden dimensionality d on HR@10 and NDCG@10 for for Yandex films dataset for BERT4Rec.**

We make some observations from this figure. The most obvious observation from these sub-figures is that the performance of each model tends to converge as the dimensionality increases. Comparing to the results authors of BERT4Rec provided for ML-1m dataset, we can see that for our data increasing hidden size from 64 to 256 gives more significant quality boost. Additionally, we can see that in most cases configuration with sequence length equals to 10 outperforms the

configuration with sequence length equals to 50.

9.7 Perfomance comparison

Here we will compare the perfomance of our models on two tasks.

9.7.1 Relevance prediction

For LSTM-based model we reached the accuracy of **0.908** with our best configuration, this configuration also was further used in production and lead to **0.5%** improvement in the final metrics in our service. Unfortunately, we were not able to achieve comparing results with BERT4Rec model due to facing with overfitting problems, to deal with it we simillarly to LSTM-based approach tried additional embeddings regularization that didn't result in competitive results. In our best BERT4Rec configuration we managed to achieve **0.841** accuracy. Nevertheless, it is important to note that such type of models (transformers) require a lot of data due to their complexity, that being said, after learning our model on the whole dataset we achieved the accuracy of **0.912**, compared to **0.906** for LSTM-based model.

9.7.2 Next item prediction

Oppositely, for next item prediction problem BERT4Rec architecture provided outstanding results, achieving the *hit*@10 of **0.64** in its best configuration, while the best value for LSTM-based model was *hit*@10 of **0.52**. As it was mentioned before, this type of task is not really applicable due to the usage of softmax layer, that is 9x times more time-consuming compared to 'relevance prediction' approach.

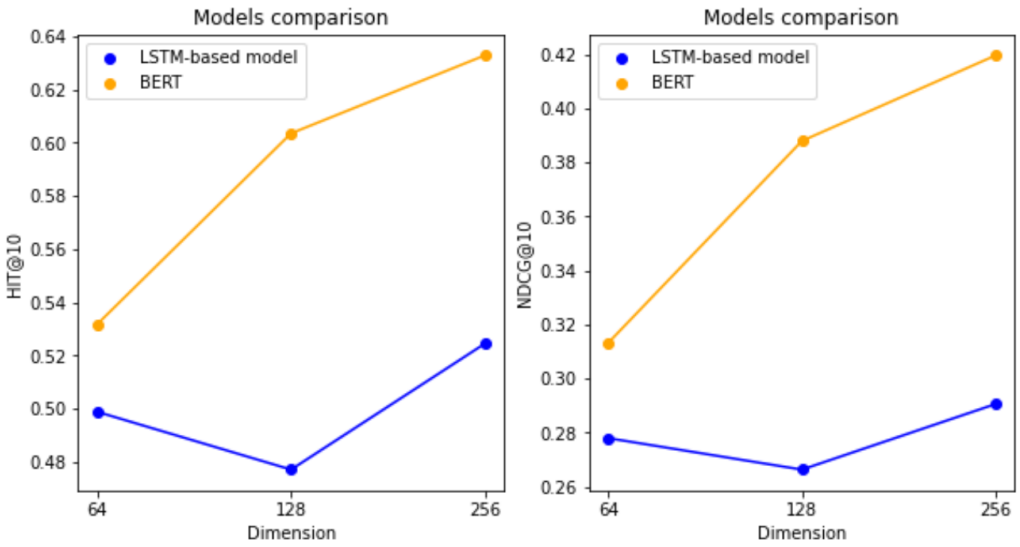

Figure 11: **Models comparison for Next item prediction task for Yandex films Dataset**

10 Conclusion And Future Work

Sequential recommender models can understand and model the sequential user behaviors, the interactions between users and items, and the evolution of users' preferences and item popularity over time, that is especially important for session-based recommendations. In this article we experimented with two sequential architectures for two different tasks, that to lead to **0.5 %** quality boost in the final metrics. Additionally it is important to note, that currently these models are used for calculating features just for candidate items, but in fact they also can be used as candidate generators.

11 References

1 Hui Fang, Guibing Guo, Danning Zhang, and Yiheng Shu. 2019. Deep Learning-Based Sequential Recommender Systems: Concepts, Algorithms, and Evaluations. In International Conference on Web Engineering. Springer, 574–577.

2 Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional

Encoder Representations from Transformer.

3 The ACM conference series on recommender systems

4 J. L. Herlocker, J. A. Konstan, A. Borchers, J. Riedl, An algorithmic framework for performing collaborative filtering, in: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, ACM, 1999, pp. 230–237.

5 G. Linden, B. Smith, J. York, Amazon. com recommendations: Item-to-item collaborative filtering, Internet Computing, IEEE 7 (1) (2003) 76–80.

6 Z.-D. Zhao, M.-S. Shang, User-based collaborative-filtering recommendation algorithms on hadoop, in: Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on, IEEE, 2010, pp. 478–481.

7 J. Wang, A. P. De Vries, M. J. Reinders, A user-item relevance model for log-based collaborative filtering, in: Advances in Information Retrieval, Springer, 2006, pp. 37–48.

8 Li, H., 2011b. A short introduction to learning to rank. IEICE Transactions on Information and Systems 94 (10), 1854–1862.

9 J. Wang, A. P. De Vries, M. J. Reinders, Unifying user-based and item-based collaborative filtering approaches by similarity fusion, in: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, 2006, pp. 501–508.

10 M. Deshpande, G. Karypis, Item-based top-n recommendation algorithms, ACM Transactions on Information Systems (TOIS) 22 (1) (2004) 143–177.

11 B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, in: Proceedings of the 10th international conference on World Wide Web, ACM, 2001, pp. 285–295.

12 Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer (8) (2009) 30–37.

13 R. Van Meteren, M. Van Someren, Using content-based filtering for recommendation, in: Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop, 2000, pp. 47–56.

14 RM. J. Pazzani, D. Billsus, Content-based recommendation systems, in: The adaptive web, Springer, 2007, pp. 325–341.

15 Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-aware recommender systems. arXiv preprint arXiv:1802.08452 (2018).

16 Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In ICDM. 191–200.

17 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In NIPS. Curran Associates, Inc., 5998–6008.

18 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. CoRR abs/1810.04805 (2018).

19 Jian Li, Zhaopeng Tu, Baosong Yang, Michael R. Lyu, and Tong Zhang. 2018. MultiHead Attention with Disagreement Regularization. In Proceedings of tEMNLP. 2897–2903.

20 Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. In Deep Learning and Representation Learning Workshop.

21 Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In Proceedings of ICLR.