BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang
Alibaba Group, Beijing, China
{ofey.sf,yanhan.lj,joshuawu.wujian,changhua.pch,hc.lx,santong.oww,jiangpeng.jp}@alibaba-inc.com

ABSTRACT

Modeling users' dynamic preferences from their historical behaviors is challenging and crucial for recommendation systems. Previous methods employ sequential neural networks to encode users' historical interactions from left to right into hidden representations for making recommendations. Despite their effectiveness, we argue that such left-to-right unidirectional models are sub-optimal due to the limitations including: ***a)*** unidirectional architectures restrict the power of hidden representation in users' behavior sequences; ***b)*** they often assume a rigidly ordered sequence which is not always practical. To address these limitations, we proposed a sequential recommendation model called **BERT4Rec**, which employs the deep bidirectional self-attention to model user behavior sequences. To avoid the information leakage and efficiently train the bidirectional model, we adopt the Cloze objective to sequential recommendation, predicting the random masked items in the sequence by jointly conditioning on their left and right context. In this way, we learn a bidirectional representation model to make recommendations by allowing each item in user historical behaviors to fuse information from both left and right sides. Extensive experiments on four benchmark datasets show that our model outperforms various state-of-the-art sequential models consistently.

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

Sequential Recommendation; Bidirectional Sequential Model; Cloze

ACM Reference Format:
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. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3357384.3357895

1 INTRODUCTION

Accurately characterizing users' interests lives at the heart of an effective recommendation system. In many real-world applications, users' current interests are intrinsically dynamic and evolving, influenced by their historical behaviors. For example, one may purchase accessories (*e.g.*, Joy-Con controllers) soon after buying a Nintendo Switch, though she/he will not buy console accessories under normal circumstances.

To model such sequential dynamics in user behaviors, various methods have been proposed to make *sequential recommendations* based on users' historical interactions [15, 22, 40]. They aim to predict the successive item(s) that a user is likely to interact with given her/his past interactions. Recently, a surge of works employ sequential neural networks, *e.g.*, Recurrent Neural Network (RNN), for sequential recommendation and obtain promising results [7, 14, 15, 56, 58]. The basic paradigm of previous work is to encode a user's historical interactions into a vector (*i.e.*, representation of user's preference) using a left-to-right sequential model and make recommendations based on this hidden representation.

Despite their prevalence and effectiveness, we argue that such left-to-right unidirectional models are not sufficient to learn optimal representations for user behavior sequences. The major limitation, as illustrated in Figure 1c and 1d, is that such unidirectional models restrict the power of hidden representation for items in the historical sequences, where each item can only encode the information from previous items. Another limitation is that previous unidirectional models are originally introduced for sequential data with natural order, *e.g.*, text and time series data. They often assume a rigidly ordered sequence over data which is not always true for user behaviors in real-world applications. In fact, the choices of items in a user's historical interactions may not follow a rigid order assumption [18, 54] due to various unobservable external factors [5]. In such a situation, it is crucial to incorporate context from both directions in user behavior sequence modeling.

To address the limitations mentioned above, we seek to use a bidirectional model to learn the representations for users' historical behavior sequences. Specifically, inspired by the success of BERT [6] in text understanding, we propose to apply the deep bidirectional self-attention model to sequential recommendation, as illustrated in Figure 1b. For representation power, the superior results for deep bidirectional models on text sequence modeling tasks show that it is beneficial to incorporate context from both sides for sequence representations learning [6]. For rigid order assumption, our model is more suitable than unidirectional models in modeling user behavior sequences since all items in the bidirectional model can leverage the contexts from both left and right side.

However, it is not straightforward and intuitive to train the bidirectional model for sequential recommendation. Conventional sequential recommendation models are usually trained left-to-right by predicting the next item for each position in the input sequence. As shown in Figure 1, jointly conditioning on both left and right

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357895

context in a deep bidirectional model would cause information leakage, *i.e.*, allowing each item to indirectly "see the target item". This could make predicting the future become trivial and the network would not learn anything useful.

To tackle this problem, we introduce the Cloze task [6, 50] to take the place of the objective in unidirectional models (*i.e.*, sequentially predicting the next item). Specifically, we randomly mask some items (*i.e.*, replace them with a special token `[mask]`) in the input sequences, and then predict the ids of those masked items based on their surrounding context. In this way, we avoid the information leakage and learn a bidirectional representation model by allowing the representation of each item in the input sequence to fuse both the left and right context. In addition to training a bidirectional model, another advantage of the Cloze objective is that it can produce more samples to train a more powerful model in multiple epochs. However, a downside of the Cloze task is that it is not consistent with the final task (*i.e.*, sequential recommendation). To fix this, during the test, we append the special token "`[mask]`" at the end of the input sequence to indicate the item that we need to predict, and then make recommendations base on its final hidden vector. Extensive experiments on four datasets show that our model outperforms various state-of-the-art baselines consistently.

The contributions of our paper are as follows:

- We propose to model user behavior sequences with a bidirectional self-attention network through Cloze task. To the best of our knowledge, this is the first study to introduce deep bidirectional sequential model and Cloze objective into the field of recommendation systems.
- We compare our model with state-of-the-art methods and demonstrate the effectiveness of both bidirectional architecture and the Cloze objective through quantitative analysis on four benchmark datasets.
- We conduct a comprehensive ablation study to analyze the contributions of key components in the proposed model.

2 RELATED WORK

In this section, we will briefly review several lines of works closely related to ours, including general recommendation, sequential recommendation, and attention mechanism.

2.1 General Recommendation

Early works on recommendation systems typically use Collaborative Filtering (CF) to model users' preferences based on their interaction histories [26, 43]. Among various CF methods, Matrix Factorization (MF) is the most popular one, which projects users and items into a shared vector space and estimate a user's preference on an item by the inner product between their vectors [26, 27, 41]. Another line of work is item-based neighborhood methods [20, 25, 31, 43]. They estimate a user's preference on an item via measuring its similarities with the items in her/his interaction history using a precomputed item-to-item similarity matrix.

Recently, deep learning has been revolutionizing the recommendation systems dramatically. The early pioneer work is a two-layer Restricted Boltzmann Machines (RBM) for collaborative filtering, proposed by Salakhutdinov et al. [42] in Netflix Prize[1].

[1]https://www.netflixprize.com

One line of deep learning based methods seeks to improve the recommendation performance by integrating the distributed item representations learned from auxiliary information, *e.g.*, text [23, 53], images [21, 55], and acoustic features [51] into CF models. Another line of work seeks to take the place of conventional matrix factorization. For example, Neural Collaborative Filtering (NCF) [12] estimates user preferences via Multi-Layer Perceptions (MLP) instead of inner product, while AutoRec [44] and CDAE [57] predict users' ratings using Auto-encoder framework.

2.2 Sequential Recommendation

Unfortunately, none of the above methods is for sequential recommendation since they all ignore the order in users' behaviors.

Early works on sequential recommendation usually capture sequential patterns from user historical interactions using Markov chains (MCs). For example, Shani et al. [45] formalized recommendation generation as a sequential optimization problem and employ Markov Decision Processes (MDPs) to address it. Later, Rendle et al. [40] combine the power of MCs and MF to model both sequential behaviors and general interests by Factorizing Personalized Markov Chains (FPMC). Besides the first-order MCs, high-order MCs are also adopted to consider more previous items [10, 11].

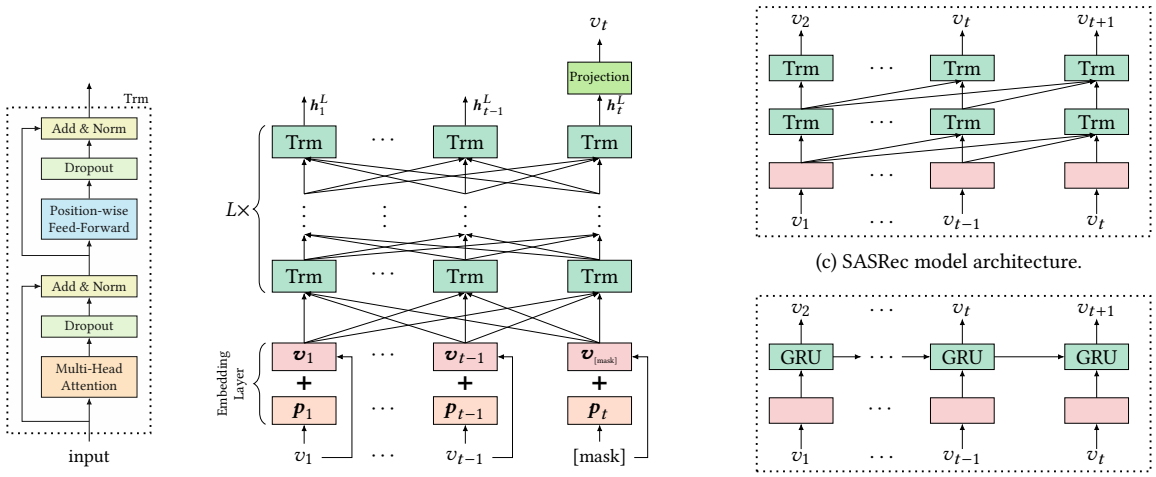
Recently, RNN and its variants, Gated Recurrent Unit (GRU) [4] and Long Short-Term Memory (LSTM) [17], are becoming more and more popular for modeling user behavior sequences [7, 14, 15, 28, 37, 56, 58]. The basic idea of these methods is to encode user's previous records into a vector (*i.e.*, representation of user's preference which is used to make predictions) with various recurrent architectures and loss functions, including session-based GRU with ranking loss (GRU4Rec) [15], Dynamic REcurrent bAsket Model (DREAM) [58], user-based GRU [7], attention-based GRU (NARM) [28], and improved GRU4Rec with new loss function (*i.e.*, `BPR-max` and `TOP1-max`) and an improved sampling strategy [14].

Other than recurrent neural networks, various deep learning models are also introduced for sequential recommendation [3, 22, 33, 49]. For example, Tang and Wang [49] propose a Convolutional Sequence Model (Caser) to learn sequential patterns using both horizontal and vertical convolutional filters. Chen et al. [3] and Huang et al. [19] employ Memory Network to improve sequential recommendation. STAMP captures both users' general interests and current interests using an MLP network with attention [33].

2.3 Attention Mechanism

Attention mechanism has shown promising potential in modeling sequential data, *e.g.*, machine translation [2, 52] and text classification [**?**]. Recently, some works try to employ the attention mechanism to improve recommendation performances and interpretability [28, 33]. For example, Li et al. [28] incorporate an attention mechanism into GRU to capture both the user's sequential behavior and main purpose in session-based recommendation.

The works mentioned above basically treat attention mechanism as an additional component to the original models. In contrast, Transformer [52] and BERT [6] are built solely on multi-head self-attention and achieve state-of-the-art results on text sequence modeling. Recently, there is a rising enthusiasm for applying purely attention-based neural networks to model sequential data for their



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

Figure 1: Differences in sequential recommendation model architectures. BERT4Rec learns a bidirectional model via Cloze task, while SASRec and RNN based methods are all left-to-right unidirectional model which predict next item sequentially.

effectiveness and efficiency [30, 32, 38, 46**?**]. For sequential recommendation, Kang and McAuley [22] introduce a two-layer *Transformer decoder* (*i.e.*, Transformer language model) called SASRec to capture user's sequential behaviors and achieve state-of-the-art results on several public datasets. SASRec is closely related to our work. However, it is still a unidirectional model using a casual attention mask. While we use a bidirectional model to encode users' behavior sequences with the help of Cloze task.

3 BERT4REC

Before going into the details, we first introduce the research problem, the basic concepts, and the notations in this paper.

3.1 Problem Statement

In sequential recommendation, let $\mathcal{U}$=$\{u_1, u_2, \ldots, u_{|\mathcal{U}|}\}$ denote a set of users, $\mathcal{V}$=$\{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$ be a set of items, and list $\mathcal{S}_u$=$[v_1^{(u)}, \ldots, v_t^{(u)}, \ldots, v_{n_u}^{(u)}]$ 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[2] t and n_u is the the length of interaction sequence for user u. 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:

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

3.2 Model Architecture

Here, we introduce a new sequential recommendation model called **BERT4Rec**, which adopts **B**idirectional **E**ncoder **R**epresentations from **T**ransformers to a new task, sequential **R**ecommendation. It is built upon the popular self-attention layer, "Transformer layer".

As illustrated in Figure 1b, 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 in Figure 1d, 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.

Comparing Figure 1b, 1c, and 1d, the most noticeable difference is that SASRec and RNN based methods are all left-to-right unidirectional architecture, while our BERT4Rec uses bidirectional self-attention to model users' behavior sequences. In this way, our proposed model can obtain more powerful representations of users' behavior sequences to improve recommendation performances.

3.3 Transformer Layer

As illustrated in Figure 1b, given an input sequence of length t, we iteratively compute hidden representations $\boldsymbol{h}_i^l$ at each layer l for each position i simultaneously by applying the Transformer layer from [52]. Here, we stack $\boldsymbol{h}_i^l \in \mathbb{R}^d$ together into matrix $\boldsymbol{H}^l \in \mathbb{R}^{t \times d}$ since we compute attention function on all positions simultaneously in practice. As shown in Figure 1a, 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 [6, 29, 52]. Thus, we here adopt the multi-head self-attention instead of performing a single attention function. Specifically, multi-head attention first linearly projects $\boldsymbol{H}^l$ into h subspaces, with different, learnable linear projections, and then apply h attention functions

[2]Here, following [22, 40], we use the relative time index instead of absolute time index for numbering interaction records.

in parallel to produce the output representations which are concatenated and once again projected:

$$\begin{aligned} \texttt{MH}(\boldsymbol{H}^l) &= [\text{head}_1; \text{head}_2; \ldots; \text{head}_h]\boldsymbol{W}^O \\ \text{head}_i &= \texttt{Attention}\big(\boldsymbol{H}^l\boldsymbol{W}_i^Q, \boldsymbol{H}^l\boldsymbol{W}_i^K, \boldsymbol{H}^l\boldsymbol{W}_i^V\big) \end{aligned} \tag{1}$$

where the projections matrices for each head $\boldsymbol{W}_i^Q \in \mathbb{R}^{d\times d/h}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{d\times d/h}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{d\times d/h}$, and $\boldsymbol{W}_i^O \in \mathbb{R}^{d\times d}$ 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*:

$$\texttt{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d/h}}\right)\boldsymbol{V} \tag{2}$$

where *query* $\boldsymbol{Q}$, *key* $\boldsymbol{K}$, and *value* $\boldsymbol{V}$ are projected from the same matrix $\boldsymbol{H}^l$ with different learned projection matrices as in Equation 1. The *temperature* $\sqrt{d/h}$ is introduced to produce a softer attention distribution for avoiding extremely small gradients [16, 52].

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:

$$\begin{aligned} \texttt{PFFN}(\boldsymbol{H}^l) &= \left[\texttt{FFN}(\boldsymbol{h}_1^l)^\top; \ldots; \texttt{FFN}(\boldsymbol{h}_t^l)^\top\right]^\top \\ \texttt{FFN}(\boldsymbol{x}) &= \texttt{GELU}\big(\boldsymbol{x}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}\big)\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)} \\ \texttt{GELU}(x) &= x\Phi(x) \end{aligned} \tag{3}$$

where $\Phi(x)$ is the cumulative distribution function of the standard gaussian distribution, $\boldsymbol{W}^{(1)} \in \mathbb{R}^{d\times 4d}$, $\boldsymbol{W}^{(2)} \in \mathbb{R}^{4d\times d}$, $\boldsymbol{b}^{(1)} \in \mathbb{R}^{4d}$ and $\boldsymbol{b}^{(2)} \in \mathbb{R}^d$ 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 [38] and BERT [6], we use a smoother `GELU` [13] 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 [9] around each of the two sub-layers as in Figure 1a, followed by layer normalization [1]. Moreover, we also apply dropout [47] to the output of each sub-layer, before it is normalized. That is, the output of each sub-layer is $\texttt{LN}(\boldsymbol{x} + \texttt{Dropout}(\text{sublayer}(\boldsymbol{x})))$, where $\text{sublayer}(\cdot)$ is the function implemented by the sub-layer itself, `LN` is the layer normalization function defined in [1]. 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:

$$\boldsymbol{H}^l = \texttt{Trm}\big(\boldsymbol{H}^{l-1}\big), \quad \forall i \in [1, \ldots, L] \tag{4}$$

$$\texttt{Trm}(\boldsymbol{H}^{l-1}) = \texttt{LN}\Big(\boldsymbol{A}^{l-1} + \texttt{Dropout}\big(\texttt{PFFN}(\boldsymbol{A}^{l-1})\big)\Big) \tag{5}$$

$$\boldsymbol{A}^{l-1} = \texttt{LN}\Big(\boldsymbol{H}^{l-1} + \texttt{Dropout}\big(\texttt{MH}(\boldsymbol{H}^{l-1})\big)\Big) \tag{6}$$

3.4 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 $\boldsymbol{h}_i^0$ is constructed by summing the corresponding item and positional embedding:

$$\boldsymbol{h}_i^0 = \boldsymbol{v}_i + \boldsymbol{p}_i$$

where $\boldsymbol{v}_i \in \boldsymbol{E}$ is the $d-$dimensional embedding for item v_i, $\boldsymbol{p}_i \in \boldsymbol{P}$ 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 [52] for better performances. The positional embedding matrix $\boldsymbol{P} \in \mathbb{R}^{N\times d}$ 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 $t > N$.

3.5 Output Layer

After L layers that hierarchically exchange information across all positions in the previous layer, we get the final output $\boldsymbol{H}^L$ for all items of the input sequence. Assuming that we mask the item v_t at time step t, we then predict the masked items v_t base on $\boldsymbol{h}_t^L$ as shown in Figure 1b. Specifically, we apply a two-layer feed-forward network with `GELU` activation in between to produce an output distribution over target items:

$$P(v) = \text{softmax}\big(\texttt{GELU}(\boldsymbol{h}_t^L\boldsymbol{W}^P + \boldsymbol{b}^P)\boldsymbol{E}^\top + \boldsymbol{b}^O\big) \tag{7}$$

where $\boldsymbol{W}^P$ is the learnable projection matrix, $\boldsymbol{b}^P$, and $\boldsymbol{b}^O$ are bias terms, $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{V}|\times d}$ 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.

3.6 Model Learning

Training. Conventional unidirectional sequential recommendation models usually train the model by predicting the next item for each position in the input sequence as illustrated in Figure 1c and 1d. 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 1b, 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 [50] (also known as "Masked Language Model" in [6]) 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:

$$\mathcal{L} = \frac{1}{|\mathcal{S}_u^m|} \sum_{v_m \in \mathcal{S}_u^m} -\log P(v_m = v_m^* | \mathcal{S}'_u) \tag{8}$$

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 7.

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 in Figure 1c and 1d 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.

Test. As described above, we create a mismatch between the training and the final sequential recommendation task since the Cloze objective is to predict the current masked items while sequential recommendation aims to predict the future. To address this, we append the special token "[mask]" to the end of user's behavior sequence, and then predict the next item based on the final hidden representation of this token. To better match the sequential recommendation task (*i.e.*, predict the last item), we also produce samples that only mask the last item in the input sequences during training. It works like fine-tuning for sequential recommendation and can further improve the recommendation performances.

3.7 Discussion

Here, we discuss the relation of our model with previous related work.

SASRec. Obviously, SASRec is a left-to-right unidirectional version of our BERT4Rec with single head attention and causal attention mask. Different architectures lead to different training methods. SASRec predicts the next item for each position in a sequence, while BERT4Rec predicts the masked items in the sequence using Cloze objective.

CBOW & SG. Another very similar work is Continuous Bag-of-Words (CBOW) and Skip-Gram (SG) [35]. CBOW predicts a target word using the average of all the word vectors in its context (both left and right). It can be seen as a simplified case of BERT4Rec, if we use one self-attention layer in BERT4Rec with uniform attention weights on items, unshare item embeddings, remove the positional embedding, and only mask the central item. Similar to CBOW, SG can also be seen as a simplified case of BERT4Rec following similar reduction operations (mask all items except only one). From this point of view, Cloze can be seen as a general form for the objective of CBOW and SG. Besides, CBOW uses a simple aggregator to model word sequences since its goal is to learn good word representations, not sentence representations. On the contrary, we seek to learn a powerful behavior sequence representation model (deep self-attention network in this work) for making recommendations.

BERT. Although our BERT4Rec is inspired by the BERT in NLP, it still has several differences from BERT: ***a)*** The most critical difference is that BERT4Rec is an end-to-end model for sequential recommendation, while BERT is a pre-training model for sentence representation. BERT leverages large-scale task-independent corpora to pre-train the sentence representation model for various text sequence tasks since these tasks share the same background knowledge about the language. However, this assumption does not hold in the recommendation tasks. Thus we train BERT4Rec end-to-end for different sequential recommendation datasets. ***b)*** Different from BERT, we remove the next sentence loss and segment embeddings since BERT4Rec models a user's historical behaviors as only one sequence in sequential recommendation task.

Table 1: Statistics of datasets.

Datasets	#users	#items	#actions	Avg. length	Density
Beauty	40,226	54,542	0.35m	8.8	0.02%
Steam	281,428	13,044	3.5m	12.4	0.10%
ML-1m	6040	3416	1.0m	163.5	4.79%
ML-20m	138,493	26,744	20m	144.4	0.54%

4 EXPERIMENTS

4.1 Datasets

We evaluate the proposed model on four real-world representative datasets which vary significantly in domains and sparsity.

- Amazon **Beauty**[3]: This is a series of product review datasets crawled from Amazon.com by McAuley et al. [34]. They split the data into separate datasets according to the top-level product categories on Amazon. In this work, we adopt the "Beauty" category.
- **Steam**[4]: This is a dataset collected from Steam, a large online video game distribution platform, by Kang and McAuley [22].
- **MovieLens** [8]: This is a popular benchmark dataset for evaluating recommendation algorithms. In this work, we adopt two well-established versions, MovieLens 1m (**ML-1m**)[5] and MovieLens 20m (**ML-20m**)[6].

For dataset preprocessing, we follow the common practice in [22, 40, 49]. For all datasets, we convert all numeric ratings or the presence of a review to implicit feedback of 1 (*i.e.*, the user interacted

[3] http://jmcauley.ucsd.edu/data/amazon/
[4] https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data
[5] https://grouplens.org/datasets/movielens/1m/
[6] https://grouplens.org/datasets/movielens/20m/

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 [12, 22, 40, 49], we keep users with at least five feedbacks. The statistics of the processed datasets are summarized in Table 1.

4.2 Task Settings & Evaluation Metrics

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 [12, 22, 49]. For each user, we hold out the last item of the behavior sequence as the test data, treat the item just before the last as the validation set, and utilize the remaining items for training. For easy and fair evaluation, we follow the common strategy in [12, 22, 49], 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 [19], 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. To evaluate the ranking list of all the models, we employ a variety of evaluation metrics, including *Hit Ratio* (HR), *Normalized Discounted Cumulative Gain* (NDCG), and *Mean Reciprocal Rank* (MRR). Considering we only have one ground truth item for each user, HR@k is equivalent to Recall@k and proportional to Precision@k; MRR is equivalent to Mean Average Precision (MAP). In this work, we report HR and NDCG with $k = 1, 5, 10$. For all these metrics, the higher the value, the better the performance.

4.3 Baselines & Implementation Details

To verify the effectiveness of our method, we compare it with the following representative baselines:

- **POP**: It is the simplest baseline that ranks items according to their popularity judged by the number of interactions.
- **BPR-MF** [39]: It optimizes the matrix factorization with implicit feedback using a pairwise ranking loss.
- **NCF** [12]: It models userâĂŞitem interactions with a MLP instead of the inner product in matrix factorization.
- **FPMC** [40]: It captures users' general taste as well as their sequential behaviors by combing MF with first-order MCs.
- **GRU4Rec** [15]: It uses GRU with ranking based loss to model user sequences for session based recommendation.
- **GRU4Rec$^+$** [14]: It is an improved version of GRU4Rec with a new class of loss functions and sampling strategy.
- **Caser** [49]: It employs CNN in both horizontal and vertical way to model high-order MCs for sequential recommendation.
- **SASRec** [22]: It uses a left-to-right Transformer language model to capture users' sequential behaviors, and achieves state-of-the-art performance on sequential recommendation.

For NCF[7], GRU4Rec[8], GRU4Rec$^+$[8], Caser[9], and SASRec[10], we use code provided by the corresponding authors. For BPR-MF and FPMC, we implement them using `TensorFlow`. For common hyper-parameters in all models, we consider the hidden dimension size d from $\{16, 32, 64, 128, 256\}$, the ℓ_2 regularizer from {1, 0.1, 0.01, 0.001, 0.0001}, and dropout rate from $\{0, 0.1, 0.2, \cdots, 0.9\}$. All other hyper-parameters (*e.g.*, Markov order in Caser) and initialization strategies are either followed the suggestion from the methods' authors or tuned on the validation sets. We report the results of each baseline under its optimal hyper-parameter settings.

We implement BERT4Rec[11] with `TensorFlow`. All parameters are initialized using truncated normal distribution in the range $[-0.02, 0.02]$. We train the model using Adam [24] 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. For fair comparison, we set the layer number $L = 2$ and head number $h = 2$ and use the same maximum sequence length as in [22], $N = 200$ for ML-1m and ML-20m, $N = 50$ for Beauty and Steam datasets. For head setting, we empirically set the dimensionality of each head as 32 (single head if $d < 32$). We tune the mask proportion ρ using the validation set, resulting in $\rho = 0.6$ for Beauty, $\rho = 0.4$ for Steam, and $\rho = 0.2$ for ML-1m and ML-20m. All the models are trained from scratch without any pre-training on a single NVIDIA GeForce GTX 1080 Ti GPU with a batch size of 256.

4.4 Overall Performance Comparison

Table 2 summarized the best results of all models on four benchmark datasets. The last column is the improvements of BERT4Rec relative to the best baseline. We omit the NDCG@1 results since it is equal to HR@1 in our experiments. It can be observed that:

The non-personalized POP method gives the worst performance[12] on all datasets since it does not model user's personalized preference using the historical records. Among all the baseline methods, sequential methods (*e.g.*, FPMC and GRU4Rec$^+$) outperforms non-sequential methods (*e.g.*, BPR-MF and NCF) on all datasets consistently. Compared with BPR-MF, the main improvement of FPMC is that it models users' historical records in a sequential way. This observation verifies that considering sequential information is beneficial for improving performances in recommendation systems.

Among sequential recommendation baselines, Caser outperforms FPMC on all datasets especially for the dense dataset ML-1m, suggesting that high-order MCs is beneficial for sequential recommendation. However, high-order MCs usually use very small order L since they do not scale well with the order L. This causes Caser to perform worse than GRU4Rec$^+$ and SASRec, especially on sparse datasets. Furthermore, SASRec performs distinctly better than GRU4Rec and GRU4Rec$^+$, suggesting that self-attention mechanism is a more powerful tool for sequential recommendation.

According to the results, it is obvious that BERT4Rec performs best among all methods on four datasets in terms of all evaluation metrics. It gains **7.24%** HR@10, **11.03%** NDCG@10, and **11.46%** MRR improvements (on average) against the strongest baselines.

Question 1: *Do the gains come from the bidirectional self-attention model or from the Cloze objective?*

[7] https://github.com/hexiangnan/neural_collaborative_filtering
[8] https://github.com/hidasib/GRU4Rec
[9] https://github.com/graytowne/caser_pytorch
[10] https://github.com/kang205/SASRec
[11] https://github.com/FeiSun/BERT4Rec
[12] What needs to be pointed out is that such low scores for POP are because the negative samples are sampled according to the items' popularity.

Table 2: Performance comparison of different methods on next-item prediction. Bold scores are the best in each row, while underlined scores are the second best. Improvements over baselines are statistically significant with $p < 0.01$.

Datasets	Metric	POP	BPR-MF	NCF	FPMC	GRU4Rec	GRU4Rec$^+$	Caser	SASRec	BERT4Rec	Improv.
Beauty	HR@1	0.0077	0.0415	0.0407	0.0435	0.0402	0.0551	0.0475	0.0906	**0.0953**	5.19%
	HR@5	0.0392	0.1209	0.1305	0.1387	0.1315	0.1781	0.1625	0.1934	**0.2207**	14.12%
	HR@10	0.0762	0.1992	0.2142	0.2401	0.2343	0.2654	0.2590	0.2653	**0.3025**	14.02%
	NDCG@5	0.0230	0.0814	0.0855	0.0902	0.0812	0.1172	0.1050	0.1436	**0.1599**	11.35%
	NDCG@10	0.0349	0.1064	0.1124	0.1211	0.1074	0.1453	0.1360	0.1633	**0.1862**	14.02%
	MRR	0.0437	0.1006	0.1043	0.1056	0.1023	0.1299	0.1205	0.1536	**0.1701**	10.74%
Steam	HR@1	0.0159	0.0314	0.0246	0.0358	0.0574	0.0812	0.0495	0.0885	**0.0957**	8.14%
	HR@5	0.0805	0.1177	0.1203	0.1517	0.2171	0.2391	0.1766	0.2559	**0.2710**	5.90%
	HR@10	0.1389	0.1993	0.2169	0.2551	0.3313	0.3594	0.2870	0.3783	**0.4013**	6.08%
	NDCG@5	0.0477	0.0744	0.0717	0.0945	0.1370	0.1613	0.1131	0.1727	**0.1842**	6.66%
	NDCG@10	0.0665	0.1005	0.1026	0.1283	0.1802	0.2053	0.1484	0.2147	**0.2261**	5.31%
	MRR	0.0669	0.0942	0.0932	0.1139	0.1420	0.1757	0.1305	0.1874	**0.1949**	4.00%
ML-1m	HR@1	0.0141	0.0914	0.0397	0.1386	0.1583	0.2092	0.2194	0.2351	**0.2863**	21.78%
	HR@5	0.0715	0.2866	0.1932	0.4297	0.4673	0.5103	0.5353	0.5434	**0.5876**	8.13%
	HR@10	0.1358	0.4301	0.3477	0.5946	0.6207	0.6351	0.6692	0.6629	**0.6970**	4.15%
	NDCG@5	0.0416	0.1903	0.1146	0.2885	0.3196	0.3705	0.3832	0.3980	**0.4454**	11.91%
	NDCG@10	0.0621	0.2365	0.1640	0.3439	0.3627	0.4064	0.4268	0.4368	**0.4818**	10.32%
	MRR	0.0627	0.2009	0.1358	0.2891	0.3041	0.3462	0.3648	0.3790	**0.4254**	12.24%
ML-20m	HR@1	0.0221	0.0553	0.0231	0.1079	0.1459	0.2021	0.1232	0.2544	**0.3440**	35.22%
	HR@5	0.0805	0.2128	0.1358	0.3601	0.4657	0.5118	0.3804	0.5727	**0.6323**	10.41%
	HR@10	0.1378	0.3538	0.2922	0.5201	0.5844	0.6524	0.5427	0.7136	**0.7473**	4.72%
	NDCG@5	0.0511	0.1332	0.0771	0.2239	0.3090	0.3630	0.2538	0.4208	**0.4967**	18.04%
	NDCG@10	0.0695	0.1786	0.1271	0.2895	0.3637	0.4087	0.3062	0.4665	**0.5340**	14.47%
	MRR	0.0709	0.1503	0.1072	0.2273	0.2967	0.3476	0.2529	0.4026	**0.4785**	18.85%

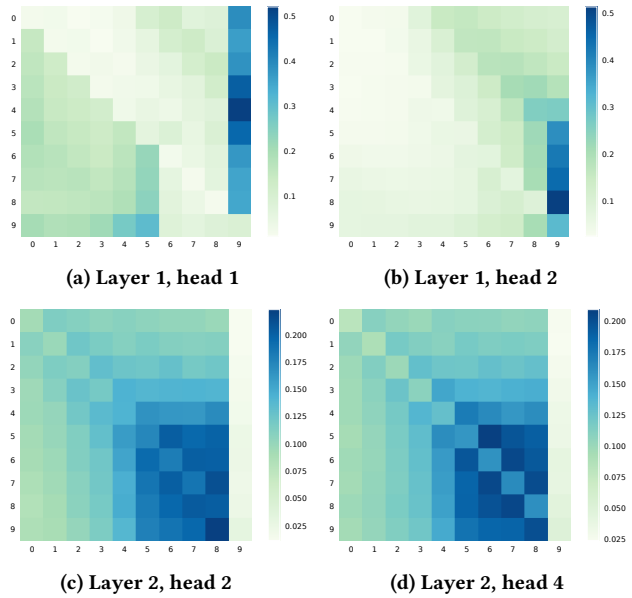
Table 3: Analysis on bidirection and Cloze with $d = 256$.

Model	Beauty			ML-1m		
	HR@10	NDCG@10	MRR	HR@10	NDCG@10	MRR
SASRec	0.2653	0.1633	0.1536	0.6629	0.4368	0.3790
BERT4Rec (1 mask)	0.2940	0.1769	0.1618	0.6869	0.4696	0.4127
BERT4Rec	0.3025	0.1862	0.1701	0.6970	0.4818	0.4254

To answer this question, we try to isolate the effects of these two factors by constraining the Cloze task to mask only one item at a time. In this way, the main difference between our BERT4Rec (with 1 mask) and SASRec is that BERT4Rec predicts the target item jointly conditioning on both left and right context. We report the results on Beauty and ML-1m with $d = 256$ in Table 3 due to the space limitation. The results show that BERT4Rec with 1 mask significantly outperforms SASRec on all metrics. It demonstrates the importance of bidirectional representations for sequential recommendation. Besides, the last two rows indicate that the Cloze objective also improves the performances. Detailed analysis of the mask proportion ρ in Cloze task can be found in § 4.6

Question 2: *Why and how does bidirectional model outperform unidirectional models?*

To answer this question, we try to reveal meaningful patterns by visualizing the average attention weights of the last 10 items during the test on Beauty in Figure 2. Due to the space limitation, we only report four representative attention heat-maps in different layers and heads.



(a) Layer 1, head 1 **(b) Layer 1, head 2**

(c) Layer 2, head 2 **(d) Layer 2, head 4**

Figure 2: Heat-maps of average attention weights on Beauty, the last position "9" denotes "`[mask]`" (best viewed in color).

We make several observations from the results. ***a)*** Attention varies across different heads. For example, in layer 1, head 1 tends

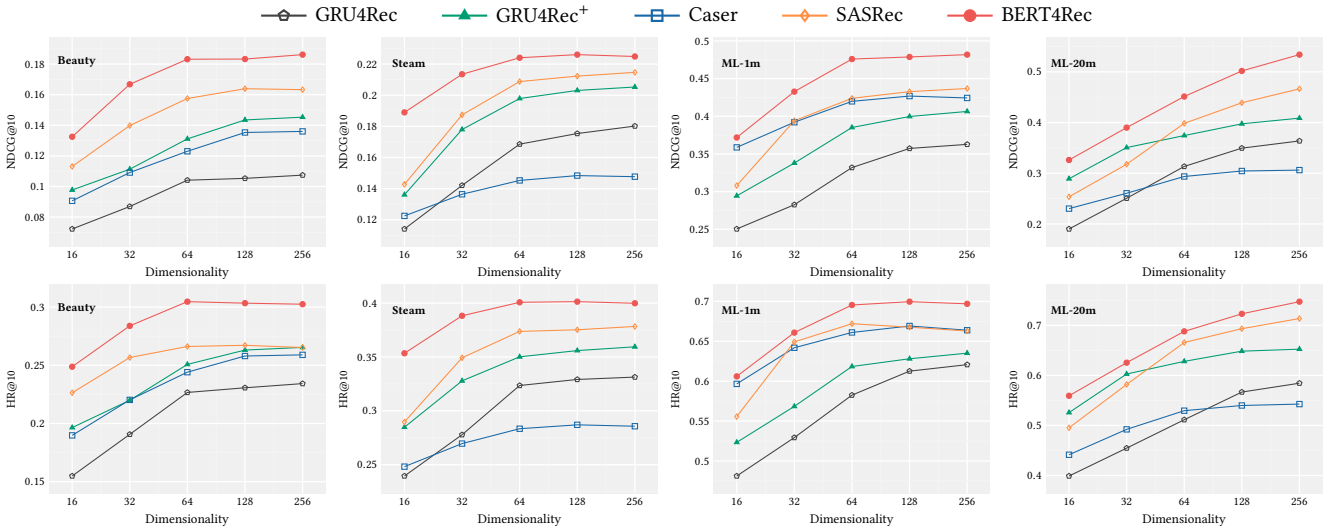


Figure 3: Effect of the hidden dimensionality d on HR@10 and NDCG@10 for neural sequential models.

to attend on items at the left side while head 2 prefers to attend on items on the right side. ***b)*** Attention varies across different layers. Apparently, attentions in layer 2 tend to focus on more recent items. This is because layer 2 is directly connected to the output layer and the recent items play a more important role in predicting the future. Another interesting pattern is that heads in Figure 2a and 2b also tend to attend on `[mask]`[13]. It may be a way for self-attention to propagate sequence-level state to the item level. ***c)*** Finally and most importantly, unlike unidirectional model can only attend on items at the left side, items in BERT4Rec tend to attend on the items at both sides. This indicates that bidirectional is essential and beneficial for user behavior sequence modeling.

In the following studies, we examine the impact of the hyper-parameters, including the hidden dimensionality d, the mask proportion ρ, and the maximum sequence length N. We analyze one hyper-parameter at a time by fixing the remaining hyper-parameters at their optimal settings. Due to space limitation, we only report NDCG@10 and HR@10 for the follow-up experiments.

4.5 Impact of Hidden Dimensionality d

We now study how the hidden dimensionality d affects the recommendation performance. Figure 3 shows NDCG@10 and HR@10 for neural sequential methods with the hidden dimensionality d varying from 16 to 256 while keeping other optimal hyper-parameters unchanged. 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. A larger hidden dimensionality does not necessarily lead to better model performance, especially on sparse datasets like Beauty and Steam. This is probably caused by overfitting. In terms of details, Caser performs unstably on four datasets, which might limit its usefulness. Self-attention based methods (*i.e.*, SASRec and BERT4Rec) achieve superior performances on all datasets. Finally, our model consistently outperforms all other baselines on all datasets even with a relatively small hidden dimensionality. Considering that our model achieves satisfactory performance with $d \geq 64$, we only report the results with $d=64$ in the following analysis.

[13]This phenomenon also exists in text sequence modeling using BERT.

4.6 Impact of Mask Proportion ρ

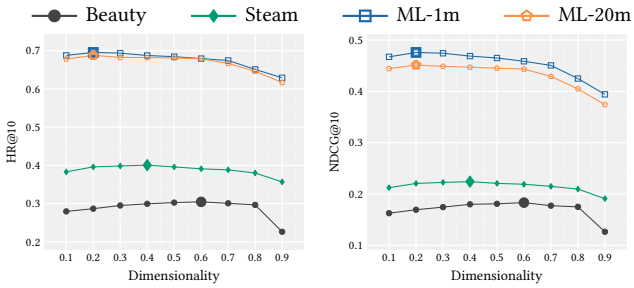


Figure 4: Performance with different mask proportion ρ on $d = 64$. Bold symbols denote the best scores in each line.

As described in § 3.6, mask proportion ρ is a key factor in model training, which directly affects the loss function (Equation 8). Obviously, mask proportion ρ should not be too small or it is not enough to learn a strong model. Meanwhile, it should not be too large, otherwise, it would be hard to train since there are too many items to guess based on a few contexts in such case. To examine this, we study how mask proportion ρ affects the recommendation performances on different datasets.

Figure 4 shows the results with varying mask proportion ρ from 0.1 to 0.9. Considering the results with $\rho > 0.6$ on all datasets, a general pattern emerges, the performances decreasing as ρ increases. From the results of the first two columns, it is easy to see that $\rho = 0.2$ performs better than $\rho = 0.1$ on all datasets. These results verify what we claimed above.

In addition, we observe that the optimal ρ is highly dependent on the sequence length of the dataset. For the datasets with short sequence length (*e.g.*, Beauty and Steam), the best performances are

Table 4: Performance with different maximum length N.

		10	20	30	40	50
Beauty	#samples/s	5504	3256	2284	1776	1441
	HR@10	0.3006	0.3061	0.3057	0.3054	0.3047
	NDCG@10	0.1826	0.1875	0.1837	0.1833	0.1832
		10	50	100	200	400
ML-1m	#samples/s	14255	8890	5711	2918	1213
	HR@10	0.6788	0.6854	0.6947	0.6955	0.6898
	NDCG@10	0.4631	0.4743	0.4758	0.4759	0.4715

achieved at ρ=0.6 (Beauty) and ρ=0.4 (Steam), while the datasets with long sequence length (*e.g.*, ML-1m and ML-20m) prefer a small ρ=0.2. This is reasonable since, compared with short sequence datasets, a large ρ in long sequence datasets means much more items that need to be predicted. Take ML-1m and Beauty as example, ρ=0.6 means we need to predict 98=$\lfloor 163.5 \times 0.6 \rfloor$ items on average per sequence for ML-1m, while it is only 5=$\lfloor 8.8 \times 0.6 \rfloor$ items for Beauty. The former is too hard for model training.

4.7 Impact of Maximum Sequence Length N

We also investigate the effect of the maximum sequence length N on model's recommendation performances and efficiency.

Table 4 shows recommendation performances and training speed with different maximum length N on Beauty and ML-1m. We observe that the proper maximum length N is also highly dependent on the average sequence length of the dataset. Beauty prefers a smaller $N = 20$, while ML-1m achieves the best performances on $N = 200$. This indicates that a user's behavior is affected by more recent items on short sequence datasets and less recent items for long sequence datasets. The model does not consistently benefit from a larger N since a larger N tends to introduce both extra information and more noise. However, our model performs very stably as the length N becomes larger. This indicates that our model can attend to the informative items from the noisy historical records.

A scalability concern about BERT4Rec is that its computational complexity per layer is $O(n^2 d)$, quadratic with the length n. Fortunately, the results in Table 4 shows that the self-attention layer can be effectively parallelized using GPUs.

4.8 Ablation Study

Finally, we perform ablation experiments over a number of key components of BERT4Rec in order to better understand their impacts, including positional embedding (PE), position-wise feed-forward network (PFFN), layer normalization (LN), residual connection (RC), dropout, the layer number L of self-attention, and the number of heads h in multi-head attention. Table 5 shows the results of our default version ($L = 2, h = 2$) and its eleven variants on all four datasets with dimensionality $d = 64$ while keeping other hyper-parameters (*e.g.*, ρ) at their optimal settings.

We introduce the variants and analyze their effects respectively:

(1) **PE**. The results show that removing positional embeddings causes BERT4Rec's performances decreasing dramatically on long sequence datasets (*i.e.*, ML-1m and ML-20m). Without the positional embeddings, the hidden representation $\boldsymbol{H}_i^L$ for each item v_i depends only on item embeddings. In this situation, we predict different target items using the same hidden representation of "`[mask]`". This makes the model ill-posed. This issue is more serious on long sequence datasets since they have more masked items to predict.

(2) **PFFN**. The results show that long sequence datasets (*e.g.*, ML-20m) benefit more from PFFN. This is reasonable since a purpose of PFFN is to integrate information from many heads which are preferred by long sequence datasets as discussed in the analysis about head number h in ablation study (5).

(3) **LN**, **RC**, and **Dropout**. These components are introduced mainly to alleviate overfitting. Obviously, they are more effective on small datasets like Beauty. To verify their effectiveness on large datasets, we conduct an experiment on ML-20m with layer L=4. The results show that NDCG@10 decreases about 10% w/o RC.

(4) **Number of layers** L. The results show that stacking Transformer layer can boost performances especially on large datasets (*e.g*, ML-20m). This verifies that it is helpful to learn more complex item transition patterns via deep self-attention architecture. The decline in Beauty with $L = 4$ is largely due to overfitting.

(5) **Head number** h. We observe that long sequence datasets (*e.g.*, ML-20m) benefit from a larger h while short sequence datasets (*e.g.*, Beauty) prefer a smaller h. This phenomenon is consistent with the empirical result in [48] that large h is essential for capturing long distance dependencies with multi-head self-attention.

Table 5: Ablation analysis (NDCG@10) on four datasets. Bold score indicates performance better than the default version, while ↓ indicates performance drop more than 10%.

Architecture	Dataset			
	Beauty	Steam	ML-1m	ML-20m
$L = 2, h = 2$	0.1832	0.2241	0.4759	0.4513
w/o PE	0.1741	0.2060	0.2155↓	0.2867↓
w/o PFFN	0.1803	0.2137	0.4544	0.4296
w/o LN	0.1642↓	0.2058	0.4334	0.4186
w/o RC	0.1619↓	0.2193	0.4643	0.4483
w/o Dropout	0.1658	0.2185	0.4553	0.4471
1 layer ($L = 1$)	0.1782	0.2122	0.4412	0.4238
3 layers ($L = 3$)	**0.1859**	**0.2262**	**0.4864**	**0.4661**
4 layers ($L = 4$)	**0.1834**	**0.2279**	**0.4898**	**0.4732**
1 head ($h = 1$)	**0.1853**	0.2187	0.4568	0.4402
4 heads ($h = 4$)	0.1830	**0.2245**	**0.4770**	**0.4520**
8 heads ($h = 8$)	0.1823	**0.2248**	0.4743	**0.4550**

5 CONCLUSION AND FUTURE WORK

Deep bidirectional self-attention architecture has achieved tremendous success in language understanding. In this paper, we introduce a deep bidirectional sequential model called BERT4Rec for sequential recommendation. For model training, we introduce the Cloze task which predicts the masked items using both left and right

context. Extensive experimental results on four real-world datasets show that our model outperforms state-of-the-art baselines.

Several directions remain to be explored. A valuable direction is to incorporate rich item features (*e.g.*, category and price for products, cast for movies) into BERT4Rec instead of just modeling item ids. Another interesting direction for the future work would be introducing user component into the model for explicit user modeling when the users have multiple sessions.

REFERENCES

[1] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR* abs/1607.06450 (2016).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of ICLR*.

[3] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential Recommendation with User Memory Networks. In *Proceedings of WSDM*. ACM, New York, NY, USA, 108–116.

[4] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of EMNLP*. Association for Computational Linguistics, 1724–1734.

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of RecSys*. ACM, New York, NY, USA, 191–198.

[6] 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).

[7] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. 2017. Sequential User-based Recurrent Neural Network Recommendations. In *Proceedings of RecSys*. ACM, New York, NY, USA, 152–160.

[8] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (Dec. 2015), 19 pages.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of CVPR*. 770–778.

[10] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based Recommendation. In *Proceedings of RecSys*. ACM, New York, NY, USA, 161–169.

[11] Ruining He and Julian McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation. In *Proceedings of ICDM*. 191–200.

[12] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of WWW*. 173–182.

[13] Dan Hendrycks and Kevin Gimpel. 2016. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR* abs/1606.08415 (2016).

[14] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. In *Proceedings of CIKM*. ACM, New York, NY, USA, 843–852.

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

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

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780.

[18] Liang Hu, Longbing Cao, Shoujin Wang, Guandong Xu, Jian Cao, and Zhiping Gu. 2017. Diversifying Personalized Recommendation with User-session Context. In *Proceedings of IJCAI*. 1858–1864.

[19] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y. Chang. 2018. Improving Sequential Recommendation with Knowledge-Enhanced Memory Networks. In *Proceedings of SIGIR*. ACM, New York, NY, USA, 505–514.

[20] Santosh Kabbur, Xia Ning, and George Karypis. 2013. FISM: Factored Item Similarity Models for top-N Recommender Systems. In *Proceedings of KDD*. ACM, New York, NY, USA, 659–667.

[21] Wang-Cheng Kang, Chen Fang, Zhaowen Wang, and Julian McAuley. 2017. Visually-Aware Fashion Recommendation and Design with Generative Image Models. In *Proceedings of ICDM*. IEEE Computer Society, 207–216.

[22] Wang-Cheng Kang and Julian McAuley. [n. d.]. Self-Attentive Sequential Recommendation. In *Proceedings of ICDM*. 197–206.

[23] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. 2016. Convolutional Matrix Factorization for Document Context-Aware Recommendation. In *Proceedings of RecSys*. ACM, New York, NY, USA, 233–240.

[24] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of ICLR*.

[25] Yehuda Koren. 2008. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *Proceedings of KDD*. ACM, 426–434.

[26] Yehuda Koren and Robert Bell. 2011. Advances in Collaborative Filtering. In *Recommender Systems Handbook*. Springer US, Boston, MA, 145–186.

[27] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37.

[28] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural Attentive Session-based Recommendation. In *Proceedings of CIKM*. ACM, New York, NY, USA, 1419–1428.

[29] Jian Li, Zhaopeng Tu, Baosong Yang, Michael R. Lyu, and Tong Zhang. 2018. Multi-Head Attention with Disagreement Regularization. In *Proceedings of tEMNLP*. 2897–2903.

[30] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A Structured Self-attentive Sentence Embedding. In *Proceedings of ICLR*.

[31] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.Com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (Jan. 2003), 76–80.

[32] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating Wikipedia by Summarizing Long Sequences. In *Proceedings of ICLR*.

[33] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: Short-Term Attention/Memory Priority Model for Session-based Recommendation. In *Proceedings of KDD*. ACM, New York, NY, USA, 1831–1839.

[34] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-Based Recommendations on Styles and Substitutes. In *Proceedings of SIGIR*. ACM, New York, NY, USA, 43–52.

[35] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).

[36] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of NIPS*. Curran Associates Inc., USA, 3111–3119.

[37] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks. In *Proceedings of RecSys*. ACM, New York, NY, USA, 130–137.

[38] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. In *OpenAI Technical report*.

[39] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of UAI*. AUAI Press, Arlington, Virginia, United States, 452–461.

[40] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing Personalized Markov Chains for Next-basket Recommendation. In *Proceedings of WWW*. ACM, New York, NY, USA, 811–820.

[41] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization. In *Proceedings of NIPS*. Curran Associates Inc., USA, 1257–1264.

[42] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann Machines for Collaborative Filtering. In *Proceedings of ICML*. 791–798.

[43] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based Collaborative Filtering Recommendation Algorithms. In *Proceedings of WWW*. ACM, New York, NY, USA, 285–295.

[44] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. AutoRec: Autoencoders Meet Collaborative Filtering. In *Proceedings of WWW*. ACM, New York, NY, USA, 111–112.

[45] Guy Shani, David Heckerman, and Ronen I. Brafman. 2005. An MDP-Based Recommender System. *J. Mach. Learn. Res.* 6 (Dec. 2005), 1265–1295.

[46] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. In *Proceedings of NAACL*. Association for Computational Linguistics, 464–468.

[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958.

[48] Gongbo Tang, Mathias Müller, Annette Rios, and Rico Sennrich. 2018. Why Self-Attention? A Targeted Evaluation of Neural Machine Translation Architectures. In *Proceedings of EMNLP*. 4263–4272.

[49] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *Proceedings of WSDM*. 565–573.

[50] Wilson L. Taylor. 1953. âĂIJCloze ProcedureâĂİ: A New Tool for Measuring Readability. *Journalism Bulletin* 30, 4 (1953), 415–433.

[51] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Proceedings of NIPS*. 2643–2651.

[52] 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.

[53] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative Deep Learning for Recommender Systems. In *Proceedings of KDD*. ACM, New York, NY, USA, 1235–1244.

[54] Shoujin Wang, Liang Hu, Longbing Cao, Xiaoshui Huang, Defu Lian, and Wei Liu. 2018. Attention-Based Transactional Context Embedding for Next-Item Recommendation. , 2532–2539 pages.

[55] Suhang Wang, Yilin Wang, Jiliang Tang, Kai Shu, Suhas Ranganath, and Huan Liu. 2017. What Your Images Reveal: Exploiting Visual Contents for Point-of-Interest Recommendation. In *Proceedings of WWW*. 391–400.

[56] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. 2017. Recurrent Recommender Networks. In *Proceedings of WSDM*. ACM, New York, NY, USA, 495–503.

[57] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In *Proceedings of WSDM*. ACM, New York, NY, USA, 153–162.

[58] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A Dynamic Recurrent Model for Next Basket Recommendation. In *Proceedings of SIGIR*. ACM, New York, NY, USA, 729–732.