Journal of Computer Science and Cybernetics, V.37, N.4 (2021), 511–527
DOI 10.15625/1813-9663/37/4/15844

DUAL TRANSFORMER ENCODERS FOR SESSION-BASED RECOMMENDATION

PHAM HOANG ANH[1], NGO XUAN BACH[1,2], TU MINH PHUONG[1,2,*]

[1]*Department of Computer Science,*
Posts and Telecommunications Institute of Technology, Hanoi, Vietnam
[2]*Machine Learning and Applications Lab,*
Posts and Telecommunications Institute of Technology, Hanoi, Vietnam

Abstract. When long-term user profiles are not available, session-based recommendation methods are used to predict the user's next actions from anonymous sessions-based data. Recent advances in session-based recommendation highlight the necessity of modeling not only user sequential behaviors but also the user's main interest in a session, while avoiding the effect of unintended clicks causing interest drift of the user. In this work, we propose a Dual Transformer Encoder Recommendation model (DTER) as a solution to address this requirement. The idea is to combine the following recipes: (1) A Transformer-based model with dual encoders capable of modeling both sequential patterns and the main interest of the user in a session; (2) A new recommendation model that is designed for learning richer session contexts by conditioning on all permutations of the session prefix. This approach provides a unified framework for leveraging the ability of the Transformer's self-attention mechanism in modeling session sequences while taking into account the user's main interest in the session. We empirically evaluate the proposed method on two benchmark datasets. The results show that DTER outperforms state-of-the-art session-based recommendation methods on common evaluation metrics.

Keywords. Recommender systems; Session-based recommendation; Self-attention; Dual Transformer.

1. INTRODUCTION

Recommender systems are widely used in online applications from different domains such as e-commerce, video-on-demand, or music online service. Most recommendation methods rely on user history logs to predict relevant items of interest for the active user. In many real-life recommendation scenarios, however, user history logs are not available, either because the users are not logged-in or tracked, or because they are newcomers. Session-based recommendation (SR) methods have been proposed to address this problem. In the session-based

Dedicated to Professor Phan Dinh Dieu on the occasion of his 85th birth anniversary.

*Corresponding author.
E-mail addresses: hoanganh.pham.12327@gmail.com (P.H.Anh); bachnx@ptit.edu.vn (N.X.Bach); phuongtm@ptit.edu.vn (T.M.Phuong).

setting, user interactions with the systems, e.g. clicks on items, are organized in anonymous sessions and the system relies solely on the session data to predict the item the user is likely to click on next.

Most SR algorithms employ some form of sequence modeling techniques to capture sequential behaviors of the user encoded in session clicks. Among methods of this kind, models based on recurrent neural networks (RNNs) are the most successful and widely used [12, 24]. Other deep neural network models such as convolutional neural networks [26], graph neural networks [31], and attention networks [18] have also shown their effectiveness in modeling session sequences.

Recent results, however, have shown that taking into account only sequential behaviors of the user is not enough to get accurate results [17, 18]. In addition to sequential behaviors, in which the last clicks influence the immediately following clicks, user actions are also governed by the main interest of the user within a session. For example, a user starts a browsing session to buy a digital camera. During the session, he/she is likely to view several similar cameras for a comparison, but may also click on a photography drone due to curiosity or by accident. A recommender that relies only on sequential behaviors would recommend another photography drone with a high probability based on the last click. However, if the system can guess that the main user's interest is cameras because most his/her clicks are related to them, it would recommend another similar camera, which is an accurate recommendation. Therefore, it is necessary for a good recommendation algorithm to consider both sequential context and the user's main interest when generating predictions.

In this work, we address this problem by proposing a neural network model called Dual Transformer Encoders for Recommendation (DTER). Our method is inspired by the *Transformer* model [27], a new neural architecture that achieved state-of-the-art accuracy and efficiency on language sequence modeling tasks [21, 33]. Transformer is solely based on an attention mechanism called "self-attention" capable of capturing syntactic and semantic relations between words in a sentence. Here, we seek to apply the self-attention idea to session-based recommendation with the hope that it can focus on important clicks on a session sequence. To better consider both sequential behavior and user's main interest, we propose using two Transformer encoders. The first encoder (local encoder) is designed mainly to capture sequential patterns, whereas the second encoder (aggregation encoder) aggregates information from session events and thus can capture the user's main interest. The outputs of two encoders are then combined to form a unified session representation that we use for decoding and make recommendations. To further reduce the effect of noisy clicks and capture user's main interest in long sessions, we propose using the permutation objective for training the model. Permutation objective has been proposed to incorporate bidirectional contexts in the unidirectional autoregressive language model [33]. Here, it is modified to protect the model from noisy clicks and focus the model on important items. To the best of our knowledge, this is the first work that uses two Transformer encoders with the permutation objective to model session data, which makes the proposed method different from existing recommendation models.

Empirically, the proposed method outperforms state-of-the-art SR methods on several benchmark datasets from the RecSys 2015 Challenge and CIKM Cup 2016. We conduct a comprehensive ablation study to examine the contributions of key design components and

quantitatively show the influence of important model hyperparameters[1].

2. RELATED WORK

This section reviews related work on session-based recommendation, including traditional and deep learning based methods. We also introduce the Transformer, a key component of our recommendation model.

2.1. Traditional recommendation methods

To work with implicit feedback data in session-based recommendation, traditional recommendation methods such as matrix completion ones [11, 15] which were designed to work with explicit feedback need to be modified or adapted. A popular approach is item-based recommendation, which recommends items that are similar to the previous item in the session. In this approach, items are considered as similar if they often co-occur (clicked together) in training sessions, while the original version of item-based recommendation treats frequently consumed items as similar ones [7]. Though these methods are easy to implement, they ignore the order of items in the session and therefore are difficult to make accurate recommendations.

Many session-based recommendation methods have been proposed to capture sequential patterns inherent in session data. Such methods usually employ sequence modeling techniques [34] or base on Markov Chains and their variations [3, 22, 23]. The main obstacle to applying Markov models to session-based recommendation is the fast growth of the state space required to cover all possible sequences of user actions over the large number of items. The issue becomes even more serious in the context of data sparsity. To ease this problem, existing methods using Markov Chains often simplify the models by reducing the Markov order [10] or using various heuristics [23], which may limit the performance of recommendation systems.

2.2. Deep learning based recommendation methods

Deep learning methods have been applied successfully to the general recommendation task as well as session-based recommendation [2, 20, 29, 35]. Among other architectures, Recurrent Neural Networks (RNNs) [9] which are specifically designed for sequence data are the most natural choice for session-based recommendation. Hidasi et al. [12] propose to use Gated Recurrent Units (GRUs) [4], a variant of RNNs, to predict the next item in a session. Experimental results on two datasets, an e-commerce and a video, shows that their model outperforms existing methods which do not use deep learning. Following this success, other RNN-based models with several modifications and/or extensions such as data augmentation [24], adding attentive mechanism over session actions [17], and combining user identities with session data [19] have been introduced and achieved impressive performance.

In addition to RNNs, Convolutional Neural Networks (CNNs) [16], another popular class of deep network architectures, have been demonstrated to be effective in session-based recommendation [25, 26]. CNNs are designed to capture local complex patterns of fixed-length

[1]The code and data will be released at publication time.

input sequences, and therefore are difficult to apply directly to the session-based recommendation task, which accepts input sessions with variable length. To overcome this problem, existing CNN-based methods for session-based recommendation usually simplify the model by considering only a few last items [25], truncating the original session to a predefined length before feeding to CNNs [26], or using multiple convolutional layers [35]. However, recent works have shown the advantages of attention-based methods such as the Transformer over RNN-based and CNN-based methods for sequential and other structured data [27]. This motivates the use of Transformer-based architectures in this work.

2.3. The Transformer

Attention, a mechanism that allows a neural network to focus on some particular parts of the input when making a prediction, has become a key component in most successful deep architectures. When combined with CNNs or RNNs, attention mechanisms have achieved great success in various tasks in computer vision and natural language processing. Recently, the Transformer, an encoder-decoder architecture based merely on attention mechanisms, achieved state-of-the-art results on sequence-to-sequence tasks [27]. The key idea of the Transformer is to use self-attention layers to retrieve other relevant positions (words) with the one we are currently processing in the input sequence (a sentence). With multi-head attention, the Transformer can focus on different positions in the input sequence and result in a powerful network which outperforms CNN/RNN based approaches. Following the great success of the Transformer, many Transformer-based architectures have been proposed in a short time, which obtained new state-of-the-art results in various NLP tasks at the time of publication, including GPT [21], BERT [8], Transformer-XL [6] , XLNet [33], and ELECTRA [5], among others [1, 8, 28].

Recently, attention mechanisms and the Transformer have been integrated into recommender systems [14, 17, 30, 32]. The most closely related work to ours is that of [14], which uses the Transformer for sequential recommendation. Despite the great capacity of modeling sequences through self-attention, Transformer-based methods with single encoder cannot model both local context (pattern over short time) and global context (preferences over long time) which often present in recommendation problems. Our model, however, employs two different Transformers for capturing sequential patterns and the main interest of the user in a session. Furthermore, our model can learn richer session contexts by conditioning/considering on all permutations of the factorization order of the session prefix.

3. METHOD

In the session-based recommendation setting, the problem is to predict what item a user is likely to click next given the current session in the form of a sequence of clicks. Formally, let $[x_1, x_2, \ldots, x_T]$ $(T \geq 2)$ denote a session consisting of T click events (e.g. item viewing) ordered by timestamps, where $x_i \in P$ $(1 \leq i \leq T)$ is the index of one clicked item and P is the set of all available items. For any given prefix of the session sequence $[x_1, x_2, \ldots, x_t]$ $(1 \leq t < T)$, the recommendation task is to estimate, for each i-th item from P, a score y_i. We view y_i as a ranking score so that an item with a higher score will get a higher chance to be clicked next (i.e. at time t). In a typically setting, top K items with the highest scores are then provided as candidates for recommendation.

3.1. Method overview

At a high level, the proposed DTER model processes each click sequence $\mathbf{x} = [x_1, x_2, \ldots, x_t]$ separately, and is trained to predict the next clicked item x_{t+1}. To achieve this, the model first encodes x into a high-dimensional session representation $\mathbf{s}$, which it then decodes to get a vector of scores $\mathbf{y} = [y_1, y_2, \ldots, y_{|P|}]$ over all items.

Figure 1 shows the architecture of the proposed model. Our DTER employs the Transformer [27] (shown as stacked blocks Tran) as the base encoder to map click sequence x into a set of hidden vectors. In order to model both user's sequential behaviors and the general interest of the session, DTER uses two separate encoders: a *local encoder* that captures sequential patterns, and an *aggregation encoder* that capture the user's general interest within the session. The outputs of the encoders are then concatenated to form the final session representation to decode. To alleviate the negative effective of unintended clicks, DTER is trained to maximize the likelihood over all permutations of x. In the following sections, we describe in detail the encoders, the decoding process, and the permutation training objective.

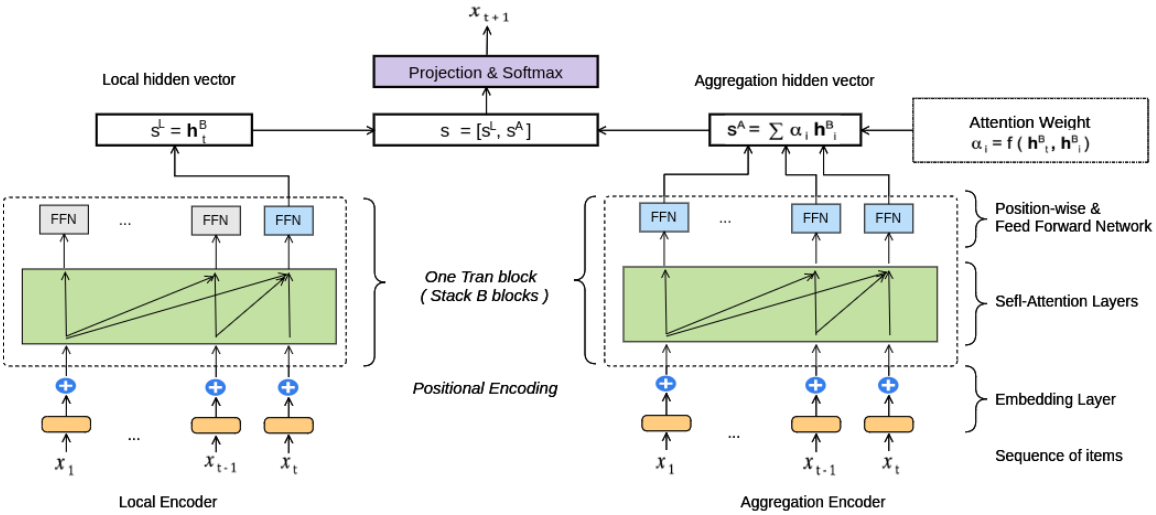

Figure 1: Graphical illustration of DTER model

3.2. Local encoder

Given the entire sequence of previous clicks, the local encoder outputs a representation that summarizes the sequential pattern in this session. The graphical model of the local encoder, shown in the left part of Figure 1, contains several stacked Transformer blocks Tran. We now describe each layer of the local encoder.

Embedding layer. This layer converts the one-hot representation of each item x_i into a dense vector embedding of size d, where d is the latent size, by retrieving row M_{x_i} of a learnable mapping matrix $\mathbf{M} \in R^{|P| \times d}$. Since the self-attention mechanism of the Transformer does not include any recurrence of convolution component, it is not aware of the positions of previous items, Therefore we add a positional embedding P_i to M_{x_i} to get the new embedding $\mathbf{e}_{x_i}$, which we use as input to the first Transformer layer (we use the fixed positional embedding $\mathbf{P}$ matrix as described in [27]

$$\mathbf{e}_{x_i} = M_{x_i} + P_i. \tag{1}$$

We transform each input sequence into a sequence of fixed length l by taking its rightmost l items if the original sequence is longer than l, or padding the left positions by zero vectors if the original sequence has less than l items. The output of the embedding layer is matrix $\mathbf{E} \in R^{|l| \times d}$, which serves as input for the first Transformer block.

Self-attention block. Given the embeddings of the input items, the local encoder applies several Transformer blocks to iteratively compute hidden vector $\mathbf{h}_i^b$ for each position i at each level b. As shown in Figure 1a, each Transformer block consists of two sub-layers: Self-Attention sub-layer, and Position-wise Feed-Forward Network.

Self-Attention sub-layer. Attention mechanism has been successfully used in many domains such as computer vision and natural language processing. The Transformer by [27] has introduced a self-attention method that uses the same objects as queries, keys and values. Specifically, the scaled-dot self-attention is computed as

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}, \tag{2}$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are query, key, and value matrices respectively. Each row of a matrix corresponds to an item and the matrices are learned jointly with the model.

For the first layer of the model, the input is the embedding matrix $\mathbf{E}$. The self-attention operation converts it into queries, keys and values by linear projections and feeds the results into a self-attention layer

$$\mathbf{A} = Attention(\mathbf{E}\mathbf{W}^Q, \mathbf{E}\mathbf{W}^K, \mathbf{E}\mathbf{W}^V), \tag{3}$$

where $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$ are d by d projection matrices to be learnt.

Given an user browsing session, the Self-Attention processes each item and associates it to other positions in that sequence, therefore, have better context to encode this item. This is more important in session-based recommendation problems since users' sessions often contain patterns (items that appear together).

In the case the hidden size d is large, it is useful to use "multi-head" attention, which applies attention in h subspaces of size $d_k = d/h$ by using $3 * h$ projection matrices of size $d \times d_k$ and concatenating the results.

Left-to-right causality. Note that at a time point t, the recommendation model does not have access to the clicks in the future. Therefore, the self-attention module cannot attend to position on the right of the current position. To forbid the information flow from right to left, we mask out all connections from position j to position i if $j > i$ when performing the softmax operation.

Point-wise Feed-Forward Network. By design, the self-attention layer encodes only linear interactions between positions. To add nonlinearity to the model, we apply the same point-wise two-layer feed-forward network on every $\mathbf{a}_i$ (network parameters are shared across all positions)

$$\mathbf{h}_i = FFN(\mathbf{a}_i) = ReLU(\mathbf{a}_i\mathbf{W}^1 + \mathbf{b}^1)\mathbf{W}^2 + \mathbf{b}^2, \tag{4}$$

where $\mathbf{W}$, and $\mathbf{b}$ are parameter matrix and vector of sizes $d \times d$ and d, respectively. ReLU is the Rectified Linear Unit activation function.

Stacking self-attention blocks. To learn more complex interactions between items and between latent dimensions, we stack several Transformer blocks on each other so that the output $\mathbf{H}^{b-1}$ of block $b-1$ is served as input for block b. Concretely, block $b, (b > 1)$ is defined as

$$\mathbf{A}^b = Attention(\mathbf{H}^{b-1}), \tag{5}$$

$$\mathbf{h}_i^b = FFN(\mathbf{A}_i^b), i = 1, \ldots, n. \tag{6}$$

When the network goes deeper with more blocks, the model is more prone to overfitting and is harder to train due to vanishing gradient and other problems. To prevent this, we apply several techniques as in the original Trans-former. Specifically, we add a dropout layer between the two sub-layer of the same self-attention block. Another dropout layer is applied on embedding matrix $\mathbf{E}$. We also add residual connections between the input and output of each sub-layer. Finally, we perform layer normalization for each sub-layer.

We abbreviate the above computation in each block by

$$\mathbf{H}^b = Tran^b(\mathbf{H}^{b-1}). \tag{7}$$

Local encoder output. For local encoder, we simply use the hidden state of the last position from the last Transformer layer as the representation for sequential behavior of the user within the session

$$\mathbf{s}^L = \mathbf{h}_t^B, \tag{8}$$

where B is the number of Trm blocks. Such representation encodes the context as summarized in the hidden state of only the last position, thus it is hard to capture the general interest of a session by using only this representation.

3.3. Aggregation encoder

The aggregation encoder is similar to the local encoder by architecture. It consists of the same number of stacked Tran blocks and takes the same input. However, the two encoders do not share parameters, except the input embedding. As its name suggests, this encoder outputs a representation that is an aggregation of all hidden vectors from all positions. In the simplest case, the aggregation operation outputs a representation that is the sum of the hidden vectors of all positions from the last Transformer blocks

$$\mathbf{s}^A = \sum_{i=1}^{t} \mathbf{h}_i^B. \tag{9}$$

However, because different items may reflect the user's main interest differently, we consider instead an item-level attention mechanism that allows the encoder to adaptively focus on important items of the input sequence

$$\alpha_i = f(\mathbf{h}_t^B, \mathbf{h}_i^B) = \mathbf{q}^T \sigma(\mathbf{W}^3 \mathbf{h}_t^B + \mathbf{W}^4 \mathbf{h}_i^B), \tag{10}$$

$$\mathbf{s}^A = \sum_{i=1}^{t} \alpha_i \mathbf{h}_i^B, \tag{11}$$

where $\mathbf{W}^3$ and $\mathbf{W}^4$ are projection matrices to be learned, $\mathbf{q}$ is a learnable vector, σ is the sigmoid function, and α_i is the attention weight, which is computed as the similarity score between the hidden state of the last and a previous i-th position.

3.4. Session representation and decoding

As explained above, the local encoder outputs a representation of the input sequence as summarized in a single hidden state of the last position, whereas the aggregation encoder explicitly gathers information from all positions. We conjecture that when used together, the two encoders can capture both the user's sequential behaviors and the main user's interest. Therefore, we create the final encoding of the input sequence by concatenating the local and aggregation vectors

$$\mathbf{s} = \mathbf{W}^5[\mathbf{s}^L; \mathbf{s}^A], \tag{12}$$

where matrix $\mathbf{W}^5$ of size $d \times 2d$ is used to project the concatenated vector of size $2d$ into the latent space of d dimensions, which is necessary for the decoding step.

Next, we feed the session embedding $\mathbf{s}$ to the output layer for decoding. First, for each i-th item from $\mathbf{P}$, we multiply its embedding $\mathbf{m}_i$ by the session vector $\mathbf{s}$ to get a score

$$z_i = \mathbf{s}^T \mathbf{m}_i. \tag{13}$$

Then, we apply the softmax function to get the final score

$$y_i = \frac{\exp(z_i)}{\sum_{j=1,\dots,|\mathbf{P}|} \exp(z_j)}, \tag{14}$$

for $i = 1, \dots, |\mathbf{P}|$, where y_i is the probability of the i-th item to be clicked next. We use the shared embedding matrix for the input and output layer to avoid overfitting.

Let $\mathbf{x}_{t+1}$ denote the one-hot vector representation of the next clicked item. The model is trained by minimizing the cross-entropy loss $L(\mathbf{x}_{t+1}, \mathbf{y})$ between ground truth $\mathbf{x}_{t+1}$ and predicted output $\mathbf{y}$.

3.5. Model training and permutation objective

Following [24], we train DTER model on prefixes of training sessions. That is, we first form training sequences of the form $([x_1], x_2)$, $([x_1, x_2], x_3)$, ... from a session of length t, where the item outside a square bracket is the label to predict given the items inside the bracket.

To alleviate the negative effect of unintended clicks within long click sequences, which are noise for the model, we borrow ideas from XLNet [33] and propose the permutation training objective for our model. Specifically, for a sequence of length t, we generate $t!$ permutations of the original sequence, each permutation corresponds to a specific order of items. We train the model on each permuted sequence to predict the next item, i.e. item clicked at time $t + 1$. When training on a permuted sequence, we use the order of clicks in this permutation to factorize the context in the Transformer layers. Recall that each position in a self-attention layer can attend only to the context on the left, which is implemented by masking out all left-to-right connections according to the permuted order. Let $\mathbf{Z}_t$ denote the set of all permutations for a click sequence of length t, and $\mathbf{z}_{<t}$ denote the items on the left of the original t-th item on a permutation $\mathbf{z} \in \mathbf{Z}_t$. We train the model to predict the next item x_{t+1} given $\mathbf{z}_{<t}$. That means in the aggregation encoder we only aggregate over items from $\mathbf{z}_{<t}$. Figure 2 illustrates the attention connections and aggregation operation for a permuted sequence.

Unlike the XLNet, which uses permutation to simulate the autoregressive objective, here we use permutation to enrich the context of a session, thus avoiding noisy clicks and better capturing the user's main interest. And unlike the embedding dropout technique proposed in [24], using permuted orders not only drop items but also alter the order of the remaining items.

Since too short sequences do not contain enough information while considering all permutation increase training time, we use only 10% permutations with the longest $\mathbf{z}_{<t}$ to train the model.

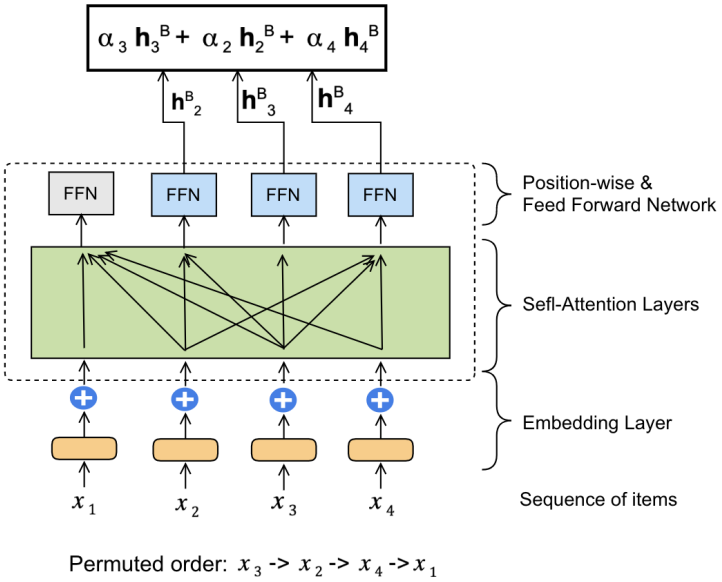

Figure 2: Illustration of permutation for the aggregation encoder. Each position can attend to the left according to the permuted order. The aggregation operation applies only the left positions of x_4 according to the permuted order.

4. EXPERIMENTS AND RESULTS

In this section we describe experiments we used to evaluate the effectiveness of the proposed method and compare it with existing methods.

4.1. Datasets

We evaluate different recommendation methods on two e-commerce datasets: Yoochoose and Diginetica.

- Yoochoose[2]. This dataset was made available for the RecSys 2015 challenge. The dataset contains click streams from a retail website for a period of six months.
- Diginetica[3]. This dataset was released for the CIKM Cup 2016 and contains click streams from an e-commerce website. We use only provided transaction data.

[2]http://2015.recsyschallenge.com/challenge.html
[3]http://cikm2016.cs.iupui.edu/cikm-cup

In both datasets, events were already divided into sessions. We keep only click events and perform preprocessing steps, following common practice in [17]. We filter out sessions of length 1 and items that appear less than 5 times. From Yoochoose we use sessions of the subsequent day to form the test set, and from Diginetica we use data of the subsequent week as the test set. Because Yoochoose dataset is large and previous work has shown the importance of focusing on more recent data [24], we create two smaller training sets Yoochoose 1/64 and Yoochoose 1/4, which contain only 1/64 and 1/4 fractions of the most recent training sequences, respectively, and report results of the models trained on the Yoochoose 1/64 and 1/4. We do not consider items that appear in a test set but not in the training set. The statistics of the datasets are summarized in Table 1.

Table 1: Statistics of datasets used in the experiments (Yoo denotes Yoochoose)

Dataset	**Yoo 1/4**	**Yoo 1/64**	**Diginetica**
Clicks	8326407	557248	982961
Items	29618	16766	43097
Training sessions	5917746	369859	719470
Test sessions	55898	55898	60858
Clicks per session	5.7	6.2	5.1

4.2. Experimental setup

4.2.1. Evaluation metrics

In practice, a recommender system can recommend only a short list of items at a time. The actual next item must present in this list, for an accurate recommendation. Therefore, we used Recall@20 as the first metric to evaluate the accuracy. Recall@20 is defined as the proportion of times the designed item appears in the top-20 recommended items in all test cases. Recall@N considers all items among top-N equally and ignores the actual ranking. Hence, we also use MRR@20 (Mean Reciprocal Rank) to evaluate the effectiveness of algorithms in giving higher rankings to actual items. MRR@20 is calculated as the average of reciprocal ranks of desired items. The reciprocal rank is set to zero if the rank exceeds 20.

4.2.2. Baselines

We evaluate our DTER by comparing it with several recommendation methods including traditional item-based kNN and state-of-the-art deep learning session-based recommendation algorithms.

- Item-KNN. This baseline recommends items similar to previous items in the sessions. The similarity is calculated as the number of co-occurrences of two items in sessions divided by the square root of the product of the number of sessions in which either of items occurs.
- GRU4REC and GRU4REC+. GRU4REC [12] is the first deep learning based session-recommendation method, which uses GRU RNNs to model sessions. The authors of GRU4REC have made a number of modifications (e.g. a new loss function), which result in a new version (GRU4REC+) with improved performance [13]. We use both versions in our experiments.

- NARM [17]. NARM is also based on GRU RNN but has an attention mechanism to adaptively focus on important items within a session, hence can capture the main purpose on a session.
- STAMP [18]. This baseline uses two MLP networks, one operates on the last clicked item while the other operates on whole session, thus it can model both the general interest in the session and the influence of the last click. An attention module is used to shift the focus between two networks.
- NextItNet [35]. This method uses Convolutional Neural Networks (CNNs) to capture sequential behaviors in session sequences.
- SR-GNN [31]. This method uses graph neural networks to represent complex transitions between items in a session and has been report to achieve state-of-the-art performance on several datasets.

We use the implementations provided by the authors of these methods (Item-KNN is provided with GRU4REC).

4.2.3. Implementation details

For perform grid search on a validation set to find the hyperparameter values for each method. We form the validation set by selecting 10% most recent sessions from the training set. Specifically we consider latent size in range [30, 100], GRU size in range [100, 1000]. Other parameters are those suggested in the implementations. For NARM, STAMP, SR-GNN, which have been tested in Yoochoose and Diginetica datasets, we use the best reported hypeparameter. We terminate training if the validation performance does not improve for 5 epochs.

For DTER, we set the maximum session length l to 20 (see Embedding layer in Method), which is enough for most sessions. We implement DTER with TensorFlow. Xavier initialization is used and the network is trained from scratch. The model is optimized using Adam optimizer, the learning rate is 0.001, and the batch size is 512. All parameters are initialized uniformly in range [-1,1] We search for the best number of Tran blocks in {1,2,3,4,6} and the best number of attention head in {1,2,4,8}. The best hyperparameters for DTER on two datasets are given in Table 2.

Table 2: The best hyperparameter values for DTER

Dataset	**Yoochoose**	**Diginetica**
Batch size	512	512
Embedding size	64	64
FFN hidden size	512	256
# of Tran blocks	2	2
# of multi-head	2	4
Embedding dropout	0.25	0.25
Tran layer dropout	0.2	0.2

4.3. Results and analysis

4.3.1. Comparison of different model designs

In the first experiment, we compare the performance of different encoder designs as measured by Recall@20 and MRR@20. Specifically, we conducted an experiment to investigate the following aspects of our DTER model:

- The effectiveness of using dual encoders.
- The usefulness of using adaptive weights in the aggregation encoder instead of fixed weights.
- The advantage of using permutation.

Table 3 summarizes the Recall@20 and MRR@20 scores on three datasets. As can be seen from the first two rows, the local encoder performs better than the aggregation encoder on Yoochoose datasets but worse on Diginetica. Since the local encoder focuses only on the last hidden vector, these results show that the last clicks have more impact on the next action on Yoochoose while the main interest expressed as aggregation over all hidden vectors is more important on Diginetica. Using both encoders consistently improves the Recall and MRR scores on all datasets. The improvements are more significant for Diginetica, on which Recall@20 improves 3.6% (7.5% relative improvement) and MRR@20 improves 2.4% (15% relative improvement). The superiority of the dual configuration provides evidence that two encoders are complement to each other and should be used together to capture more complex patterns in session sequences. The model with adaptive weights achieves better or equal accuracy than the model with fixed weights (the former outperforms latter on Yoochoose but achieves similar results on Diginetica). The results also show that using permutation leads to further improvements across datasets and evaluation metrics. For example, the model with dual encoders and permutation achieves the highest Recall and MRR values of 71.0 and 31.1 on Yoochoose 1/64.

Table 3: Performance comparison of different DTER models. The best scores in each column are boldfaced.

Models	**Yoochoose 1/64**		**Yoochoose 1/4**		**Diginetica**	
	Recall@20	**MRR@20**	**Recall@20**	**MRR@20**	**Recall@20**	**MRR@20**
Local encoder only	70.26	30.75	70.67	31.22	48.24	15.52
Aggregation encoder only	67.82	28.51	68.72	28.98	50.65	17.06
Dual encoder, fixed weights	70.51	30.87	71.01	31.47	51.75	17.78
Dual encoder, adaptive weights	70.73	31.01	71.32	31.83	51.82	17.89
Dual encoder with permutation	**71.04**	**31.12**	**71.95**	**32.03**	**52.02**	**17.90**

4.3.2. Comparison with baselines

In the next experiment we compare the accuracy values of our DTER model with those of the baselines. Tables 4 shows results over three datasets. As can be seen, Item-KNN and GRU4REC perform the worst across datasets with GRU4REC achieve better accuracy than Item-KNN on Yoochoose but lower accuracy on Diginetica. NextItNet performs just slightly better than GRU4REC and Item-KNN but is far inferior to the remaining methods, possibly because CNNs can capture only sequential patterns. GRU4REC+ substantially outperforms

Table 4: Performance comparison of DTER and baselines on three datasets. The best scores in each column are boldfaced.

Models	**Yoochoose 1/64**		**Yoochoose 1/4**		**Diginetica**	
	Recall@20	**MRR@20**	**Recall@20**	**MRR@20**	**Recall@20**	**MRR@20**
Item-kNN	51.60	21.81	52.31	21.70	35.75	11.57
GRU4Rec	60.64	22.89	59.53	22.60	29.45	8.33
GRU4Rec+	68.21	29.90	68.62	30.25	42.51	13.34
NARM	68.32	28.63	69.73	29.23	49.70	16.17
STAM	68.74	29.67	70.44	30.00	45.64	14.32
NextItNet	61.05	23.17	60.24	22.63	30.28	9.19
SR-GNN	70.57	30.94	71.36	31.89	50.73	17.59
DTER	**71.04**	**31.12**	**71.95**	**32.03**	**52.02**	**17.90**

GRU4REC and NextItNet, showing the usefulness of modifications it made to vanilla RNN to make the model more suitable for session-based recommendation.

Methods that model both sequential behaviors and main session's interest, i.e. NARM, SR-GNN, and DTER, achieve top Recall@20 and MRR@20 scores and their superiority to other methods is more significant on Diginetica dataset. For example, NARM achieves 7.2% Recall and 3% MRR improvements (12% and 20% relative improvements) over GRU4REC+ on Diginetica.

Our DTER achieve the highest Recall and MRR values across all datasets. On Diginetica dataset, for example, DTER gains 2.1% Recall and 1.5% MRR improvements (4.2% and 8.8% relative improvements respectively) against the second best method (SR-GNN). The superior performance of DTER might be attributed to the fact that it is built on top of several successful design solutions such as self-attention [27], dual encoders [17], and permutation training objective [33].

4.3.3. Further observations

We also study the influence of different parameters and components on the performance of DTER.

Number of Tran blocks. For this, we keep other hyperparameters at the optimal values and vary the number of layers between 1 and 6. The results are given in Table 5. As the results show, DTER achieves the best performance with only two blocks for all datasets. Adding more blocks leads to lower accuracy, possibly due to overfitting.

Number of attention heads. We fix the hidden size at d=64 and vary the number of heads in range 1,2,4,8. The results are summarized in Table 6. The authors of the Transformer [27] found that a large number of heads (eight or more) is useful for language modeling tasks. In our case, however, two or four heads yield good results across datasets. A possible reason is that the hidden size in this case is only 64, far less than 512 in their work.

5. CONCLUSION

We have proposed a novel method called DTER for session-based recommendation. Based on the Transformer architecture that was successful for language modeling, we elaborate by using two Transformers encoders designed to capture both user's sequential behaviors and

Table 5: Influence of number of Tran blocks (R@20 and M@20 denote Recall@20 and MRR@20, respectively). The best values in each column are boldfaced.

#layers	**Yoochoose 1/64**		**Yoochoose 1/4**		**Diginetica**	
	R@20	**M@20**	**R@20**	**M@20**	**R@20**	**M@20**
1	71.00	31.11	71.91	32.00	51.87	17.78
2	**71.04**	**31.12**	**71.95**	32.03	**52.02**	**17.90**
3	70.95	30.97	71.78	31.92	51.92	17.81
4	70.82	30.93	71.52	**32.63**	51.51	17.72
5	70.64	30.85	71.31	32.58	51.04	17.45
6	70.41	30.78	71.05	32.30	50.81	17.13

Table 6: Influence of number of attention heads (R@20 and M@20 denote Recall@20 and MRR@20, respectively). The best values in each column are boldfaced.

#heads	**Yoochoose 1/64**		**Yoochoose 1/4**		**Diginetica**	
	R@20	**M@20**	**R@20**	**M@20**	**R@20**	**M@20**
1	70.92	31.05	71.72	31.93	51.52	17.56
2	**71.04**	**31.12**	**71.95**	**32.03**	51.70	17.72
4	70.87	30.92	71.52	31.77	**52.02**	**17.90**
8	71.00	31.15	71.89	32.05	51.95	17.90

main interests in a session. We jointly train two encoders on permutations of original session sequences to reduce the negative effect of unintended clicks. Empirical results on real datasets show the superiority of the proposed method over state-of-the-art session-based recommendation methods. The results also highlight the importance of each model component. It is interesting to investigate how other features such as timestamps or item descriptions can be incorporated via the attention mechanism and their impact on recommendation accuracy, which we left for the future work.

REFERENCES

[1] C. Alt, M. Hübner, and L. Hennig, "Fine-tuning pre-trained transformer language models to distantly supervised relation extraction," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 1388–1398. [Online]. Available: https://www.aclweb.org/anthology/P19-1134

[2] Z. Batmaz, A. Yurekli, A. Bilge, and C. Kaleli, "A review on deep learning for recommender systems: challenges and remedies," *Artificial Intelligence Review*, vol. 52, pp. 1–37, 2019.

[3] S. Chen, J. L. Moore, D. Turnbull, and T. Joachims, "Playlist prediction via metric embedding," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '12. New York, NY, USA: ACM, 2012, pp. 714–722. [Online]. Available: http://doi.acm.org/10.1145/2339530.2339643

[4] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine

translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://www.aclweb.org/anthology/D14-1179

[5] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1xMH1BtvB

[6] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov., "Transformer-XL: attentive language models beyond a fixed-length context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 2978–2988.

[7] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 143–177, Jan. 2004. [Online]. Available: http://doi.acm.org/10.1145/963770.963776

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

[9] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[10] F. Figueiredo, B. Ribeiro, J. M. Almeida, and C. Faloutsos, "Tribeflow: Mining & predicting user trajectories," in *Proceedings of the 25th International Conference on World Wide Web*, ser. WWW '16. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2016, pp. 695–706. [Online]. Available: https://doi.org/10.1145/2872427.2883059

[11] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Commun. ACM*, vol. 35, no. 12, pp. 61–70, Dec. 1992. [Online]. Available: http://doi.acm.org/10.1145/138859.138867

[12] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in *Proceedings of the International Conference on Learning Representations*, 2016.

[13] B. Hidasi and A. Karatzoglou, "Recurrent neural networks with top-k gains for session-based recommendations," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, ser. CIKM '18. New York, NY, USA: ACM, 2018, pp. 843–852. [Online]. Available: http://doi.acm.org/10.1145/3269206.3271761

[14] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *Proceedings of IEEE International Conference on Data Mining (ICDM'18)*, 2018, pp. 197–206.

[15] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.263

[16] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 396–404.

[17] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma, "Neural attentive session-based recommendation," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, ser. CIKM '17. New York, NY, USA: ACM, 2017, pp. 1419–1428. [Online]. Available: http://doi.acm.org/10.1145/3132847.3132926

[18] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang, "STAMP: short-term attention/memory priority model for session-based recommendation," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: ACM, 2018, pp. 1831–1839. [Online]. Available: http://doi.acm.org/10.1145/3219819.3219950

[19] T. M. Phuong, T. C. Thanh, and N. X. Bach, "Neural session-aware recommendation," *IEEE Access*, vol. 7, pp. 86 884–86 896, 2019.

[20] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi, "Personalizing session-based recommendations with hierarchical recurrent neural networks," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, ser. RecSys '17. New York, NY, USA: ACM, 2017, pp. 130–137. [Online]. Available: http://doi.acm.org/10.1145/3109859.3109896

[21] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[22] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 811–820. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772773

[23] G. Shani, R. I. Brafman, and D. Heckerman, ""An MDP-based Recommender System"," *arXiv e-prints*, p. arXiv:1301.0600, "Dec" "2012".

[24] Y. K. Tan, X. Xu, and Y. Liu, "Improved recurrent neural networks for session-based recommendations," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, ser. DLRS 2016. New York, NY, USA: ACM, 2016, pp. 17–22. [Online]. Available: http://doi.acm.org/10.1145/2988450.2988452

[25] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, ser. WSDM '18. New York, NY, USA: ACM, 2018, pp. 565–573. [Online]. Available: http://doi.acm.org/10.1145/3159652.3159656

[26] T. X. Tuan and T. M. Phuong, "3D convolutional networks for session-based recommendation with content features," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, ser. RecSys '17. New York, NY, USA: ACM, 2017, pp. 138–146. [Online]. Available: http://doi.acm.org/10.1145/3109859.3109900

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[28] E. Voita, R. Sennrich, and I. Titov, "The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 4396–4406.

[29] S. Wang, L. Cao, and Y. Wang, "A survey on session-based recommender systems," *arXiv:1902.04864*, 2019. [Online]. Available: https://arxiv.org/abs/1902.04864

[30] S. Wang, L. Hu, L. Cao, X. Huang, D. Lian, and W. Liu, "Attention-based transactional context embedding for next-item recommendation," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.

[31] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural networks," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2019.

[32] J. Xiao, H. Ye, X. He, H. Zhang, F. Wu, and T.-S. Chua, "Attentional factorization machines: Learning the weight of feature interactions via attention networks," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, ser. IJCAI'17. AAAI Press, 2017, p. 3119–3125.

[33] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: generalized autoregressive pretraining for language understanding," *arXiv:1906.08237*, 2019. [Online]. Available: https://arxiv.org/abs/1906.08237

[34] G. Yap, X. Li, and P. Yu, "Effective next-items recommendation via personalized sequential pattern mining," in *Proceedings of the 17th International Conference on Database Systems for Advanced Applications*, 2012, pp. 48–64.

[35] F. Yuan, A. Karatzoglou, I. Arapakis, J. M. Jose, and X. He, "A simple convolutional generative network for next item recommendation," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, ser. WSDM '19. New York, NY, USA: ACM, 2019, pp. 582–590.

Received on January 18, 2021
Accepted on July 13, 2021