SSE-PT: Sequential Recommendation Via Personalized Transformer

Liwei Wu
University of California, Davis
liwu@ucdavis.edu

Shuqing Li
University of California, Davis
qshli@ucdavis.edu

Cho-Jui Hsieh
University of California, Los Angles
chohsieh@cs.ucla.edu

James Sharpnack
University of California, Davis
jsharpna@ucdavis.edu

ABSTRACT

Temporal information is crucial for recommendation problems because user preferences are naturally dynamic in the real world. Recent advances in deep learning, especially the discovery of various attention mechanisms and newer architectures in addition to widely used RNN and CNN in natural language processing, have allowed for better use of the temporal ordering of items that each user has engaged with. In particular, the SASRec model, inspired by the popular Transformer model in natural languages processing, has achieved state-of-the-art results. However, SASRec, just like the original Transformer model, is inherently an un-personalized model and does not include personalized user embeddings. To overcome this limitation, we propose a Personalized Transformer (SSE-PT) model, outperforming SASRec by almost 5% in terms of NDCG@10 on 5 real-world datasets. Furthermore, after examining some random users' engagement history, we find our model not only more interpretable but also able to focus on recent engagement patterns for each user. Moreover, our SSE-PT model with a slight modification, which we call SSE-PT++, can handle extremely long sequences and outperform SASRec in ranking results with comparable training speed, striking a balance between performance and speed requirements. Our novel application of the Stochastic Shared Embeddings (SSE) regularization is essential to the success of personalization. Code and data are open-sourced at https://github.com/wuliwei9278/SSE-PT.

CCS CONCEPTS

• **Information systems** → **Collaborative filtering**; **Recommender systems**; • **Computing methodologies** → **Neural networks**; **Learning latent representations**.

KEYWORDS

recommender system, sequential recommendation, stochastic shared embeddings, personalized transformer, neural networks, temporal collaborative ranking

1 INTRODUCTION

Traditionally, recommender system works mainly focus on standard collaborative filtering and ranking approaches [34–36, 38, 39]. The sequential recommendation problem has been a relatively new but important open research question, yet using temporal information to improve recommendation performance has proven to be challenging. SASRec, proposed by [16] for sequential recommendation problems, has achieved state-of-the-art results and enjoyed more than 10x speed-up when compared to earlier CNN/RNN-based methods. However, the model used in SASRec is the standard Transformer which is inherently an un-personalized model. In practice, it is important to include a personalized Transformer in SASRec especially for recommender systems, but [16] found that adding additional personalized embeddings did not improve the performance of their Transformer model, and postulate that the failure of adding personalization is due to the fact that they already use the user history and the user embeddings only contribute to overfitting. In this work, we propose a novel method, Personalized Transformer (SSE-PT), that successfully introduces personalization into self-attentive neural network architectures.

Introducing user embeddings into the standard transformer model is intrinsically difficult with existing regularization techniques, as unavoidably a large number of user parameters are introduced, which is often at the same scale of the number of training data. But we show that personalization can greatly improve ranking performance with a recent regularization technique called Stochastic Shared Embeddings (SSE) [37]. The personalized Transformer (SSE-PT) model with SSE regularization works well for all 5 real-world datasets we consider without overfitting, outperforming previous state-of-the-art algorithm SASRec by almost 5% in terms of NDCG@10. Furthermore, after examining some random users' engagement history, we find our model is not only more interpretable but also able to focus on recent engagement patterns for each user. Moreover, our SSE-PT model with a slight modification, which we call SSE-PT++, can handle extremely long sequences and outperform SASRec in ranking results with comparable training speed, striking a balance between performance and speed requirements.

Note that our SSE-PT method does not directly compete against Bert4Rec [30] so we exclude Bert4Rec in our experiments, because our backbone is the original transformer model [33] instead of the more advanced Bert Model [6]. But this does not invalidate the main contribution of the paper, which is to show the effectiveness of personalization when applied to transformer-based session recommendation models, by taking SASRec as an example. We have confidence to believe that our personalization technique can be similarly applied to Bert4Rec and models with more advanced Bert Backbones (such as Roberta [22], ALBert[20], and Electra [5]) or GPT-3 [3] in future works.

2 RELATED WORK

2.1 Session-based and Sequential Recommendation

Both session-based and sequential (i.e., next-basket) recommendation algorithms take advantage of additional temporal information to make better personalized recommendations. The main difference between session-based recommendations [12] and sequential recommendations [16] is that the former assumes that the user ids are not recorded and therefore the length of engagement sequences are relatively short. Therefore, session-based recommendations normally do not consider user factors. On the other hand, sequential recommendation treats each sequence as a user's engagement history [16]. Both settings, do not explicitly require time-stamps: only the relative temporal orderings are assumed known (in contrast to, for example, timeSVD++ [18] using time-stamps). Initially, sequence data in temporal order are usually modelled with Markov models, in which a future observation is conditioned on the last few observed items [26]. In [26], a personalized Markov model with user latent factors is proposed for more personalized results.

In recent years, deep learning techniques, borrowed from natural language processing (NLP) literature, are getting widely used in tackling sequential data. Like word sentences in NLP, item sequences in recommendations can be similarly modelled by recurrent neural networks (RNN) [11, 12] and convolutional neural network (CNN) [31] models. Recently, attention models are increasingly used in both NLP [6, 33] and recommender systems [16, 21]. SASRec [16] is a recent method with state-of-the-art performance among the many deep learning models. Motivated by the Transformer model in neural machine translation [33], SASRec utilizes a similar architecture to the encoder part of the Transformer model. Our proposed model, SSE-PT, is a personalized extension of the transformer model.

2.2 Regularization Techniques

In deep learning, models with many more parameters than data points can easily overfit to the training data. This may prevent us from adding user embeddings as additional parameters into complicated models like the Transformer model [16], which can easily have 20 layers with millions of parameters for a medium-sized dataset like Movielens10M [8]. ℓ_2 regularization [14] is the most widely used approach and has been used in many matrix factorization models in recommender systems; ℓ_1 regularization [32] is used when a sparse model is preferred. For deep neural networks, it has been shown that ℓ_p regularizations are often too weak, while

dropout [13, 29] is more effective in practice. There are many other regularization techniques, including parameter sharing [7], max-norm regularization [28], gradient clipping [24], etc. Very recently, a new regularization technique called Stochastic Shared Embeddings (SSE) [37] is proposed as a new means of regularizing embedding layers, which has been proved useful not only in recommender systems but also in natural language processing and computer vision [1, 37]. In this paper, we find that the base version SSE-SE is essential to the success of our Personalized Transformer (SSE-PT) model.

3 METHODOLOGY

3.1 Sequential Recommendation

Given n users and each user engaging with a subset of m items in a temporal order, the goal of sequential recommendation is to learn a good personalized ranking of top K items out of total m items for any given user at any given time point. We assume data in the format of n item sequences:

$$s_i = (j_{i1}, j_{i2}, \ldots, j_{iT}) \text{ for } 1 \le i \le n. \tag{1}$$

Sequences s_i of length T contain indices of the last T items that user i has interacted with in the temporal order (from old to new). For different users, the sequence lengths can vary, but we can pad the shorter sequences so all of them have length T. We cannot simply randomly split data points into train/validation/test sets because they come in temporal orders. Instead, we need to make sure our training data is before validation data which is before test data temporally. We use last items in sequences as test sets, second-to-last items as validation sets and the rest as training sets. We use ranking metrics such as NDCG@K and Recall@K for evaluations, which are defined in the Appendix.

3.2 Personalized Transformer Architecture

Our model, which we call SSE-PT, is motivated by the Transformer model in [33] and [16]. It also utilizes a new regularization technique called stochastic shared embeddings [37]. In the following sections, we are going to examine each important component of our Personalized Transformer (SSE-PT) model, especially the embedding layer, and the novel application of stochastic shared embeddings (SSE) regularization technique.

Embedding Layer. We define a learnable user embedding look-up table $U \in R^{n \times d_u}$ and item embedding look-up table $V \in R^{m \times d_i}$, where d_u, d_i are the number of hidden units for user and item respectively. We also specify learnable positional encoding table $P \in R^{T \times d}$, where $d = d_u + d_i$. So each input sequence $s_i \in R^T$ will be represented by the following embedding:

$$E = \begin{bmatrix} [v_{j_{i1}}; u_i] + p_1 \\ [v_{j_{i2}}; u_i] + p_2 \\ \vdots \\ [v_{j_{iT}}; u_i] + p_T \end{bmatrix} \in R^{T \times d}, \tag{2}$$

where $[v_{j_{it}}; u_i]$ represents concatenating item embedding $v_{j_{it}} \in R^{d_i}$ and user embedding $u_i \in R^{d_u}$ into embedding $E_t \in R^d$ for time t. Note that the main difference between our model and [16]
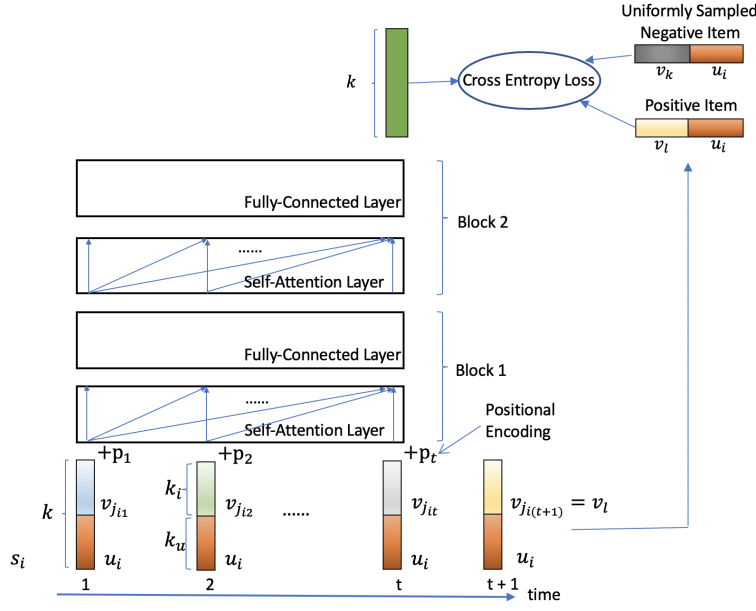
Figure 1: Illustration of our proposed SSE-PT model

is that we introduce the user embeddings u_i, making our model personalized.

Transformer Encoder. On top of the embedding layer, we have B blocks of self-attention layers and fully connected layers, where each layer extracts features for each time step based on the previous layer's outputs. Since this part is identical to the Transformer encoder used in the original papers [16, 33], we will skip the details.

Prediction Layer. At time t, the predicted probability of user i engaged item l is:
$$p_{itl} = \sigma(r_{itl}), \tag{3}$$
where σ is the sigmoid function and r_{itl} is the predicted score of item l by user l at time point t, defined as:
$$r_{itl} = F_{t-1}^{B} \cdot [v_l; u_i], \tag{4}$$
where F_{t-1}^{B} is the output hidden units associated with the transformer encoder at the last timestamp. Although we can use another set of user and item embedding look-up tables for the u_i and v_l, we find it better to use the same set of embedding look-up tables U, V as in the embedding layer. But regularization for those embeddings can be different. To distinguish the u_i and v_l in (4) from u_i, v_j in (2), we call embeddings in (4) output embeddings and those in (2) input embeddings.

The binary cross entropy loss between predicted probability for the positive item $l = j_{i(t+1)}$ and one uniformly sampled negative item $k \in \Omega$ is given as $-[\log(p_{itl}) + \log(1 - p_{itk})]$. Summing over s_i and t, we obtain the objective function that we want to minimize is:
$$\sum_i \sum_{t=1}^{T-1} \sum_{k \in \Omega} -\big[\log(p_{itl}) + \log(1 - p_{itk})\big]. \tag{5}$$
At the inference time, top-K recommendations for user i at time t can be made by sorting scores r_{itl} for all items ℓ and recommending the first K items in the sorted list.

Novel Application of Stochastic Shared Embeddings. The most important regularization technique to SSE-PT model is the Stochastic Shared Embeddings (SSE) [37]. The main idea of SSE is to stochastically replace embeddings with another embedding with some pre-defined probability during SGD, which has the effect of regularizing the embedding layers. Without SSE, all the existing well-known regularization techniques like layer normalization, dropout and weight decay fail and cannot prevent the model from over-fitting badly after introducing user embeddings. [37] develops two versions of SSE, SSE-Graph and SSE-SE. In the simplest uniform case, SSE-SE replaces one embedding with another embedding uniformly with probability p, which is called SSE probability in [37]. Since we don't have knowledge graphs for user or items, we simply apply the SSE-SE to our SSE-PT model. We find SSE-SE makes possible training this personalized model with $O(nd_u)$ additional parameters.

There are 3 different places in our model that SSE-SE can be applied. We can apply SSE-SE to input/output user embeddings, input item embeddings, and output item embeddings with probabilities p_u, p_i and p_y respectively. Note that input user embedding and output user embedding are always replaced at the same time with SSE probability p_u. Empirically, we find that SSE-SE to user embeddings and output item embeddings always helps, but SSE-SE to input item embeddings is only useful when the average sequence length is large, e.g., more than 100 in Movielens1M and Movielens10M datasets.

Other Regularization Techniques. Besides the *SSE* [37], we also utilized other widely used regularization techniques, including *layer normalization* [2], *batch normalization* [15], *residual connections* [9], *weight decay* [19], and *dropout* [29]. Since they are used in the same way in the previous paper [16], we omit the details to the Appendix.

3.3 Handling Long Sequences: SSE-PT++

To handle extremely long sequences, a slight modification can be made on the base SSE-PT model in terms of how input sequences s_i's are fed into the SSE-PT neural network. We call the enhanced model SSE-PT++ to distinguish it from the previously discussed SSE-PT model, which cannot handle sequences longer than T.

The motivation of SSE-PT++ over SSE-PT comes from: sometimes we want to make use of extremely long sequences, $s_i = (j_{i1}, j_{i2}, \ldots, j_{it})$ for $1 \leq i \leq n$, where $t > T$, but our SSE-PT model can only handle sequences of maximum length of T. The simplest way is to sample starting index $1 \leq v \leq t$ uniformly and use $s_i = (j_{iv}, j_{i(v+1)}, \ldots, j_{iz})$, where $z = \min(t, v + T - 1)$. Although sampling the starting index uniformly from $[1, t]$ can accommodate long sequences of length $t > T$, this does not work well in practice. Uniform sampling does not take into account the importance of recent items in a long sequence. To solve this dilemma, we introduce an additional hyper-parameter p_s which we call *sampling probability*. It implies that with probability p_s, we sample the starting index v uniformly from $[1, t - T]$ and use sequence $s_i = (j_{iv}, j_{i(v+1)}, \ldots, j_{i(v+T-1)})$ as input. With probability $1 - p_s$, we simply use the recent T items $(j_{i(t-T+1)}, \ldots, j_{it})$ as input. If the sequence s_i is already shorter than T, then we always use the recent input sequence for user i.

Our proposed SSE-PT++ model can work almost as well as SSE-PT with a much smaller T. One can see in Table 2 with $T = 100$, SSE-PT++ can perform almost as well as SSE-PT. The time complexity of the SSE-PT model is of order $O(T^2d + Td^2)$. Therefore, reducing T by one half would lead to a theoretically 4x speed-up in terms of the training and inference speeds. As to the model's space complexity, both SSE-PT and SSE-PT++ are of order $O(nd_u + md_i + Td + d^2)$.

4 EXPERIMENTS

In this section, we compare our proposed algorithms, Personalized Transformer (SSE-PT) and SSE-PT++, with other state-of-the-art algorithms on real-world datasets. We implement our codes in Tensorflow and conduct all our experiments on a server with 40-core Intel Xeon E5-2630 v4 @ 2.20GHz CPU, 256G RAM and Nvidia GTX 1080 GPUs.

Datasets. We use 5 datasets. The first 4 have exactly the same train/dev/test splits as in [16]. The datasets are: *Beauty* and *Games* categories from Amazon product review datasets[1]; *Steam* dataset introduced in [16], which contains reviews crawled from a large video game distribution platform; *Movielens1M* dataset [8], a widely used benchmark datasets containing one million user movie ratings; *Movielens10M* dataset with ten million user ratings cleaned by us. Detailed dataset statistics are given in Table 4. One can easily see that the first 3 datasets have short sequences (average length < 12) while the last 2 datasets have very long sequences (> 10x longer).

Evaluation Metrics. The evaluation metrics we use are standard ranking metrics, namely NDCG and Recall for top recommendations (See Appendix). We follow the same evaluation setting as the previous paper [16]: predicting ratings at time point $t + 1$ given the previous t ratings. For a large dataset with numerous users and items, the evaluation procedure would be slow because (6) would

require computing the ranking of all items based on their predicted scores for every single user. As a means of speed-up evaluations, we sample a fixed number C (e.g., 100) of negative candidates while always keeping the positive item that we know the user will engage next. This way, both R_{ij} and Π_i will be narrowed down to a small set of item candidates, and prediction scores will only be computed for those items through a single forward pass of the neural network.

Ideally, we want both NDCG and Recall to be as close to 1 as possible, because NDCG@$K = 1$ means the positive item is always put on the top-1 position of the top-K ranking list, and Recall@$K = 1$ means the positive item is always contained by the top-K recommendations the model makes.

Baselines. We include 5 non-deep-learning and 6 deep-learning algorithms in our comparisons.

Non-deep-learning Baselines. The simplest baseline is *PopRec*, basically ranking items according to their popularity. More advanced methods such as matrix factorization based baselines include Bayesian personalized ranking for implicit feedback [25], namely *BPR*; Factorized Markov Chains and Personalized Factorized Markov Chains models [26] also known as *FMC* and *PFMC*; and translation based method [10] called *TransRec*.

Deep-learning Baselines. Recent years have seen many advances in deep learning for sequential recommendations. *GRU4Rec* is the first RNN-based method proposed for this problem [12]; *GRU4Rec*$^+$ [11] later is proposed to address some shortcomings of the initial version. *Caser* is the corresponding CNN-based method [31]. *STAMP* [21] utilizes the attention mechanism without using RNN or CNN as building blocks. Very recently, *SASRec* utilizes state-of-art Transformer encoder [33] with self-attention mechanisms. Hierarchical gating networks, also known as *HGN* [23] are also proposed to solve this problem. For comparisons, we have to change HGN evaluation criteria into SASRec's, which may put HGN at a disadvantage. HGN uses a similar setup to GRU4REC [12] and GRU4Rec$^+$ [11], so it is not too surprising to see that both HGN and GRU4REC under-perform under SASRec's evaluation criteria in Table 1. But still HGN is more superior than GRU4REC and GRU4Rec$^+$ in this unfavorable scenario. We exclude Bert4Rec [30] from our baselines due to the reason we described in Section 2.1.

Experiment Setup. We use the same datasets as in [16] and follow the same procedure in the paper: use last items for each user as test data, second-to-last as validation data and the rest as training data. We implemented our method in Tensorflow and solve it with Adam Optimizer [17] with a learning rate of 0.001, momentum exponential decay rates $\beta_1 = 0.9$, $\beta_2 = 0.98$ and a batch size of 128. In Table 1, since we use the same data, the performance of previous methods except STAMP have been reported in [16]. We tune the dropout rate, and SSE probabilities p_u, p_i, p_y for input user/item embeddings and output embeddings on validation sets and report the best NDCG and Recall for top-K recommendations on test sets. For a fair comparison, we restrict all algorithms to use up to 50 hidden units for item embeddings. For the SSE-PT and SASRec models, we use the same number of transformer encoder blocks (i.e. $B = 2$) and set the maximum length $T = 200$ for Movielens 1M and 10M dataset and $T = 50$ for other datasets. We use top-K with $K = 10$ and the number of negatives $C = 100$ in the evaluation

---

[1]http://jmcauley.ucsd.edu/data/amazon/

Table 1: Comparing various state-of-the-art temporal collaborative ranking algorithms on various datasets. The (A) to (E) are non-deep-learning methods, the (F) to (K) are deep-learning methods and the (L) to (O) are our variants. We did not report SSE-PT++ results for beauty, games and steam, as the input sequence lengths are very short (see Table 4), so there is no need for SSE-PT++.

Dataset	BEAUTY		GAMES		STEAM		ML-1M	
Metric	Recall@10	NDCG@10	Recall@10	NDCG@10	Recall@10	NDCG@10	Recall@10	NDCG@10
(A) POPRec	0.4003	0.2277	0.4724	0.2779	0.7172	0.4535	0.4329	0.2377
(B) BPR	0.3775	0.2183	0.4853	0.2875	0.7061	0.4436	0.5781	0.3287
(C) FMC	0.3771	0.2477	0.6358	0.4456	0.7731	0.5193	0.6983	0.4676
(D) FPMC	0.4310	0.2891	0.6802	0.4680	0.7710	0.5011	0.7599	0.5176
(E) TRANSREC	0.4607	0.3020	0.6838	0.4557	0.7624	0.4852	0.6413	0.3969
(F) GRU4REC	0.2125	0.1203	0.2938	0.1837	0.4190	0.2691	0.5581	0.3381
(G) STAMP	0.4607	0.3020	0.6838	0.4557	0.7624	0.4852	0.6413	0.3969
(H) GRU4REC+	0.3949	0.2556	0.6599	0.4759	0.8018	0.5595	0.7501	0.5513
(I) CASER	0.4264	0.2547	0.5282	0.3214	0.7874	0.5381	0.7886	0.5538
(J) SASREC	0.4837	0.3220	0.7434	0.5401	0.8732	0.6293	0.8233	0.5936
(K) HGN	0.4469	0.2994	0.7164	0.5209	0.7426	0.4871	0.7584	0.5241
(L) SSE-SASREC	0.4878	0.3342	0.7517	0.5535	0.8697	0.6333	0.8230	0.5995
(M) PT	0.3954	0.2449	0.6427	0.4434	0.7535	0.4853	0.7658	0.5241
(N) SSE-PT	**0.5028**	**0.3370**	**0.7757**	**0.5660**	**0.8772**	**0.6378**	0.8341	0.6281
(O) SSE-PT++	–	–	–	–	–	–	**0.8389**	**0.6292**

Table 2: Comparing SASRec, SSE-PT and SSE-PT++ on Movielens1M Dataset while varying the maximum length allowed and dimension of embeddings.

Methods	NDCG@10	Recall@10	Max Len	user dim	item dim
SASREC	0.5769	0.8045	100	N/A	100
SASREC	0.5936	0.8233	200	N/A	50
SASREC	0.5919	0.8202	200	N/A	100
SSE-PT	0.6142	0.8212	100	50	100
SSE-PT	0.6191	0.8358	200	50	50
SSE-PT	0.6281	0.8341	200	50	100
SSE-PT++	0.6186	0.8318	100	50	100
SSE-PT++	0.6208	0.8358	200	50	50
SSE-PT++	**0.6292**	**0.8389**	200	50	100

procedure. In practice, using a different K and C does not affect our conclusions.

Comparisons. One can easily see from Table 1 that our proposed SSE-PT has the best performance over all previous methods on all four datasets. On most datasets, our SSE-PT improves NDCG by more than 4% when compared with SASRec [16] and more than 20% when compared to non-deep-learning methods. SSE-SE, together with dropout and weight decay, is the best choice for regularization, which is evident from Table 3. SSE-SE is a more effective way to regularize our neural networks than any existent techniques including parameter sharing, dropout, weight decay. In practice, these SSE probabilities, just like dropout rate, can be treated as tuning parameters and easily tuned. Movielens10M results are left to Table 6 in the Appendix.

4.1 Attention Mechanism Visualization

Apart from evaluating our SSE-PT against SASRec using well-defined ranking metrics on real-world datasets, we also visualize the differences between both methods in terms of their attention mechanisms. In Figure 2, a random user's engagement history in Movielens1M dataset is given in temporal order (column-wise). We hide the last item whose index is 26 in test set and hope that a temporal collaborative ranking model can figure out item-26 is the one this user will watch next using only previous engagement history. One can see for a typical user; they tend to look at a different style of movies at different times. Earlier on, they watched a variety of movies, including Sci-Fi, animation, thriller, romance, horror, action, comedy and adventure. But later on, in the last two columns of Figure 2, drama and thriller are the two types they like to watch most, especially the drama type. In fact, they watched 9

Table 3: Comparing Different Regularizations for SSE-PT on Movielen1M Dataset. NO REG stands for no regularization. PS stands for parameter sharing across all users while PS(AGE) means PS is used within each age group. SASRec is added to last row after all SSE-PT results as a baseline.

Regularization	NDCG@5	% GAIN	Recall@5	% GAIN
NO REG (BASELINE)	0.4855	-	0.6500	-
PS	0.5065	4.3	0.6656	2.4
PS (JOB)	0.4938	1.7	0.6570	1.1
PS (GENDER)	0.5110	5.3	0.6672	2.6
PS (AGE)	0.5133	5.7	0.6743	3.7
l_2	0.5149	6.0	0.6786	4.4
DROPOUT	0.5165	6.4	0.6823	5.0
l_2 + DROPOUT	0.5293	9.0	0.6921	6.5
SSE-SE	0.5393	11.1	0.6977	7.3
l_2 + SSE-SE + DROPOUT	**0.5870**	**20.9**	**0.7442**	**14.5**
SASRec (l_2 + DROPOUT)	0.5601		0.7164	

Figure 2: Illustration of how SASRec (Left) and SSE-PT (Right) differs on utilizing the Engagement History of A Random User in Movielens1M Dataset.

drama movies out of recent 10 movies. For humans, it is natural to reason that the hidden movie should probably also be drama type. So what about the machine's reasoning?

Table 4: Description of Datasets Used in Evaluations.

dataset	#users	#items	avg sequence len	max sequence len
Beauty	52,024	57,289	7.6	291
games	31,013	23,715	7.3	858
steam	334,730	13,047	11.0	1,229
ml-1m	6,040	3,416	163.5	2,275
ml-10m	69,878	65,133	141.1	7,357

For our SSE-PT, the hidden item indexed 26 is put in the first place among its top-5 recommendations. Intelligently, the SSE-PT recommends 3 drama movies, 2 thriller movies and mixing them up in positions. Interestingly, the top recommendation is 'Othello', which like the recently watched 'Richard III', is an adaptation of a Shakespeare play, and this dependence is reflected in the attention weight. On the contrast, SASRec cannot provide top-5 recommendations that are personalized enough. It recommends a variety of action, Sci-Fi, comedy, horror, and drama movies but none of them match item-26. Although this user has watched all these types of movies in the past, they do not watch these anymore as one can easily tell from his recent history. Unfortunately, SASRec cannot capture this and does not provide personalized recommendations for this user by focusing more on drama and thriller movies. It is easy to see that in contrast, our SSE-PT model shares with human reasoning that more emphasis should be placed on recent movies.
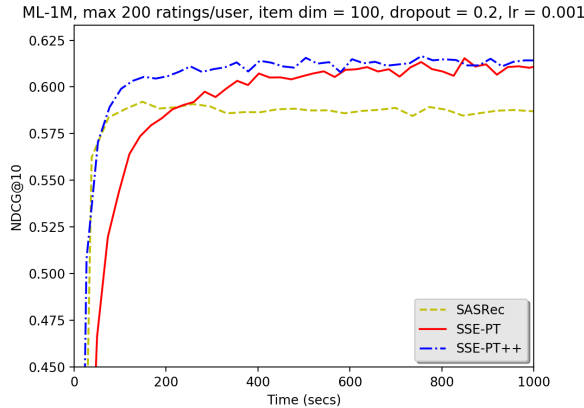

Figure 3: Illustration of the speed of SSE-PT

4.2 Training Speed
In [16], it has been shown that SASRec is about 11 times faster than Caser and 17 times faster than GRU4Rec$^+$ and achieves much better NDCG@10 results so we did not include Caser and GRU4Rec$^+$ in our comparisons. In Figure 3, we only compare the training speeds and ranking performances among SASRec, SSE-PT and SSE-PT++ for Movielens1M dataset. Given that we added additional user

embeddings into our SSE-PT model, it is expected that it will take slightly longer to train our model than un-personalized SASRec. We find empirically that training speed of the SSE-PT and SSE-PT++ model are comparable to that of SASRec, with SSE-PT++ being the fastest and the best performing model. It is clear that our SSE-PT and SSE-PT++ achieve much better ranking performances than our baseline SASRec using the same training time.

4.3 Ablation Study
SSE probability. Given the importance of SSE regularization for our SSE-PT model, we carefully examined the SSE probability for input user embedding in Table 7 in Appendix. We find that the appropriate hyper-parameter SSE probability is not very sensitive: anywhere between 0.4 and 1.0 gives good results, better than parameter sharing and not using SSE-SE. This is also evident based on comparison results in Table 3.

Sampling Probability. Recall that the sampling probability is unique to our SSE-PT++ model. We show in Table 8 in Appendix using an appropriate sampling probability like $0.2 \rightarrow 0.3$ would allow it to outperform SSE-PT when the same maximum length is used.

Number of Attention Blocks. We find for our SSE-PT model, a larger number of attention blocks is preferred. One can easily see in Table 9 in Appendix, the optimal ranking performances are achieved at $B = 4$ or 5 for Movielens1M dataset and at $B = 6$ for Movielens10M dataset.

Personalization and Number of Negatives Sampled. Based on the results in Table 10 in Appendix, we are positive that the personalized model always outperforms the un-personalized one when we use the same regularization techniques. This holds true regardless of how many negatives sampled or what ranking metrics are used during evaluation.

5 CONCLUSION
In this paper, we propose a novel neural network architecture called Personalized Transformer for the temporal collaborative ranking problem. It enjoys the benefits of being a personalized model, therefore achieving better ranking results for individual users than the current state-of-the-art. By examining the attention mechanisms during inference, the model is also more interpretable and tends to pay more attention to recent items in long sequences than un-personalized deep learning models.

6 APPENDIX
6.0.1 Ranking Metrics.
- NDCG@K: defined as:

$$\text{NDCG@}K = \frac{1}{n} \sum_{i=1}^{n} \frac{\text{DCG@}K(i, \Pi_i)}{\text{DCG@}K(i, \Pi_i^*)}, \quad (6)$$

Table 5: Comparing our SSE-PT, SSE-PT++ with SASRec on Movielen1M dataset. We use number of negatives $C = 100$, dropout probability of 0.2 and learning rate of $1e^{-3}$ for all experiments while varying others. p_u, p_i, p_u are SSE probabilities for user embedding, input item embedding and output item embedding respectively.

	Movielens1m		Dimensions		Number of Blocks	Sampling Probability	SSE-SE Parameters		
Model	NDCG@10	Recall@10	d_u	d_i	b	p_s	p_u	p_i	p_y
SASRec	0.5961	0.8195	-	50	2	-	-	-	-
SASRec	0.5941	0.8182	-	100	2	-	-	-	-
SASRec	**0.5996**	**0.8272**	-	100	6	-	-	-	-
SSE-PT	0.6101	0.8343	50	50	2	-	0.92	0.1	0
SSE-PT	0.6164	0.8336	50	50	2	-	0.92	0	0.1
SSE-PT	0.5832	0.8091	50	50	2	-	0	0.1	0.1
SSE-PT	**0.6174**	**0.8351**	50	50	2	-	0.92	0.1	0.1
SSE-PT	0.5949	0.8205	75	25	2	-	0.92	0.1	0.1
SSE-PT	**0.6214**	**0.8359**	25	75	2	-	0.92	0.1	0.1
SSE-PT	0.6281	0.8341	50	100	2	-	0.92	0.1	0.1
SSE-PT++	**0.6292**	**0.8389**	50	100	2	0.3	0.92	0.1	0.1

Table 6: Comparing our SSE-PT with SASRec on Movielens10M dataset. Unlike Table 5, we use the number of negatives $C = 500$ instead of 100 as $C = 100$ is too easy for this dataset and it gets too difficult to tell the differences between different methods: Hit Ratio@10 approaches 1.

	Movielens1m		Dimensions		Number of Blocks	SSE-SE Parameters		
Model	NDCG@10	Recall@10	d_u	d_i	b	p_u	p_i	p_y
SASRec	0.7268	0.9429	-	50	2	-	-	-
SASRec	0.7413	0.9474	-	100	2	-	-	-
SSE-PT	0.7199	0.9331	50	100	2	PS	0.01	0.01
SSE-PT	0.7169	0.9296	50	100	2	0.0	0.01	0.01
SSE-PT	0.7398	0.9418	50	100	2	0.2	0.01	0.01
SSE-PT	0.7500	0.9500	50	100	2	0.4	0.01	0.01
SSE-PT	0.7484	0.9480	50	100	2	0.6	0.01	0.01
SSE-PT	**0.7529**	0.9485	50	100	2	0.8	0.01	0.01
SSE-PT	0.7503	**0.9505**	50	100	2	1.0	0.01	0.01

where i represents i-th user and

$$\text{DCG@}K(i, \Pi_i) = \sum_{l=1}^{K} \frac{2^{R_{i\Pi_{il}}} - 1}{\log_2(l + 1)}. \tag{7}$$

In the DCG definition, Π_{il} represents the index of the l-th ranked item for user i in test data based on the learned score matrix X. R is the rating matrix and R_{ij} is the rating given to item j by user i. Π_i^* is the ordering provided by the ground truth rating.

• Recall@K: defined as a fraction of positive items retrieved by the top K recommendations the model makes:

$$\text{Recall@}K = \frac{\sum_{i=1}^{n} \mathbb{q}\{\exists 1 \leq l \leq K : R_{i\Pi_{il}} = 1\}}{n}, \tag{8}$$

here we already assume there is only a single positive item that user will engage next and the indicator function $\mathbb{q}\{\exists 1 \leq l \leq k : R_{i\Pi_{il}} = 1\}$ is defined to indicate whether the positive item falls into the top K position in our obtained ranked list using scores predicted in (4).

Table 7: Comparing Different SSE probability for user embeddings for SSE-PT on Movielens1M Dataset. Embedding hidden units of 50 for users and 100 for items, attention blocks of 2, SSE probability of 0.01 for item embeddings, dropout probability of 0.2 and max length of 200 are used.

User-Side SSE-SE Probability	NDCG@10	Recall@10
Parameter Sharing	0.6188	0.8294
1.0	0.6258	0.8346
0.9	**0.6275**	0.8321
0.8	0.6244	0.8359
0.6	0.6256	0.8341
0.4	0.6237	**0.8369**
0.2	0.6163	0.8281
0.0	0.5908	0.8048

6.0.2 Regularization Techniques.

Table 8: Comparing Different Sampling Probability, p_s, of SSE-PT++ on Movielens1M Dataset. Hyper-parameters the same as Table 7, except that the max length T allowed is set 100 instead of 200 to show effects of sampling sequences.

Sampling Probability	NDCG@10	Recall@10
SASRec ($T = 100$)	0.5769	0.8045
SSE-PT ($T = 100$)	0.6142	0.8212
1.0	0.5697	0.7977
0.8	0.5735	0.7801
0.6	0.6062	0.8242
0.4	0.6113	0.8273
0.3	0.6186	**0.8318**
0.2	**0.6193**	0.8233
0.0	0.6142	0.8212

Layer Normalization. Layer normalization [2] normalizes neurons within a layer. Previous studies [2] show it is more effective than batch normalization for training recurrent neural networks (RNNs). One alternative is the batch normalization [15] but we find it does not work as well as the layer normalization in practice even for a reasonable large batch size of 128. Therefore, our SSE-PT model adopts layer normalization.

Residual Connections. Residual connections are firstly proposed in ResNet for image classification problems [9]. Recent research finds that residual connections can help training very deep neural networks even if they are not convolutional neural networks [33]. Using residual connections allows us to train very deep neural networks here. For example, the best performing model for Movielens10M dataset in Table 9 is the SSE-PT with 6 attention blocks, in which $1 + 6 * 3 + 1 = 20$ layers are trained end-to-end.

Weight Decay. Weight decay [19], also known as l_2 regularization [14], is applied to all embeddings, including both user and item embeddings.

Dropout. Dropout [29] is applied to the embedding layer E, self-attention layer and pointwise feed-forward layer by stochastically dropping some percentage of hidden units to prevent co-adaption of neurons. Dropout has been shown to be an effective way of regularizing deep learning models.

In summary, layer normalization and dropout are used in all layers except prediction layer. Residual connections are used in both self-attention layer and pointwise feed-forward layer. SSE-SE is used in embedding layer and prediction layer.

6.0.3 Non-Deep-learning Baselines.

- PopRec: ranking items according to their popularity.
- BPR: Bayesian personalized ranking for implicit feedback setting [25]. It is a low-rank matrix factorization model with a pairwise loss function. But it does not utilize the temporal information. Therefore, it serves as a strong baseline for non-temporal methods.
- FMC: Factorized Markov Chains: a first-order Markov Chain method, in which predictions are made only based on previously engaged item.

Table 9: Comparing Different Number of Blocks for SSE-PT while Keeping The Rest Fixed on Movielens1M and Movielens10M Datasets.

Datasets	# of blocks	NDCG@10	Recall@10
	SASREC (6 blocks)	0.5984	0.8207
Movielens1M	1	0.6162	0.8301
	2	0.6280	0.8365
	3	0.6293	0.8376
	4	0.6270	**0.8401**
	5	**0.6308**	0.8361
	6	0.6270	0.8397
	SASRec (6 blocks)	0.7531	0.9490
Movielens10M	1	0.7454	0.9478
	2	0.7512	0.9522
	3	0.7543	0.9491
	4	0.7608	0.9485
	5	0.7619	0.9524
	6	**0.7683**	**0.9537**

Table 10: Varying number of negatives C in evaluation on Movielens1M dataset. Other hyper-parameters are fixed for a fair comparison.

METRIC	NDCG@10	Recall@10	C
Un-Personalized	0.3787	0.6119	500
Personalized	**0.3846**	**0.6171**	500
Un-Personalized	0.2791	0.4781	1000
Personalized	**0.2860**	**0.4929**	1000
Un-Personalized	0.1939	0.3515	2000
Personalized	**0.1993**	**0.3667**	2000

- PFMC: a personalized Markov chain model [26] that combines matrix factorization and first-order Markov Chain to take advantage of both users' latent long-term preferences as well as short-term item transitions.
- TransRec: a first-order sequential recommendation method [10] in which items are embedded into a transition space and users are modelled as translation vectors operating on item sequences.

SQL-Rank [36] and item-based recommendations [27] are omitted because the former is similar to BPR [25] except using the listwise loss function instead of the pairwise loss function and the latter has been shown inferior to TransRec [10].

6.0.4 Deep-learning Baselines.

- GRU4Rec: the first RNN-based method proposed for the session-based recommendation problem [12]. It utilizes the GRU structures [4] initially proposed for speech modelling.
- GRU4Rec$^+$: follow-up work of GRU4Rec by the same authors: the model has a very similar architecture to GRU4Rec but has a more complicated loss function [11].

- Caser: a CNN-based method [31] which embeds a sequence of recent items in both time and latent spaces forming an 'image' before learning local features through horizontal and vertical convolutional filters. In [31], user embeddings are included in the prediction layer only. On the contrast, in our Personalized Transformer, user embeddings are also introduced in the lowest embedding layer so they can play an important role in self-attention mechanisms as well as in prediction stages.
- STAMP: a session-based recommendation algorithm [21] using attention mechanism. [21] only uses fully connected layers with one attention block that is not self-attentive.
- SASRec: a self-attentive sequential recommendation method [16] motivated by Transformer in NLP [33]. Unlike our method SSE-PT, SASRec does not incorporate user embedding and therefore is not a personalized method. SASRec paper [16] also does not utilize SSE [37] for further regularization: only used dropout and weight decay.
- HGN: hierarchical gating networks method to solve the sequential recommendation problem [23], which incorporates the user embeddings and gating networks for better personalization than the SASRec model.

REFERENCES

[1] Mahdi Abavisani, Liwei Wu, Shengli Hu, Joel Tetreault, and Alejandro Jaimes. 2020. Multimodal Categorization of Crisis Events in Social Media. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14679–14689.
[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).
[3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).
[4] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
[5] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555* (2020).
[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[7] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.
[8] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2016), 19.
[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[10] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 161–169.
[11] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 843–852.
[12] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).
[13] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).
[14] Arthur E Hoerl and Robert W Kennard. 1970. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics* 12, 1 (1970), 55–67.

[15] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).
[16] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. *arXiv preprint arXiv:1808.09781* (2018).
[17] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[18] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 447–456.
[19] Anders Krogh and John A Hertz. 1992. A simple weight decay can improve generalization. In *Advances in neural information processing systems*. 950–957.
[20] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).
[21] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. 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*. ACM, 1831–1839.
[22] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[23] Chen Ma, Peng Kang, and Xue Liu. 2019. Hierarchical Gating Networks for Sequential Recommendation. *arXiv preprint arXiv:1906.09217* (2019).
[24] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. 1310–1318.
[25] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.
[26] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 811–820.
[27] Badrul Munir Sarwar, George Karypis, Joseph A Konstan, John Riedl, et al. 2001. Item-based collaborative filtering recommendation algorithms. *Www* 1 (2001), 285–295.
[28] Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. 2005. Maximum-margin matrix factorization. In *Advances in neural information processing systems*. 1329–1336.
[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[30] 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 *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1441–1450.
[31] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 565–573.
[32] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.
[33] 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 *Advances in Neural Information Processing Systems*. 5998–6008.
[34] Liwei Wu. 2020. Advances in Collaborative Filtering and Ranking. *arXiv preprint arXiv:2002.12312* (2020).
[35] Liwei Wu, Cho-Jui Hsieh, and James Sharpnack. 2017. Large-scale Collaborative Ranking in Near-Linear Time. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 515–524.
[36] Liwei Wu, Cho-Jui Hsieh, and James Sharpnack. 2018. SQL-Rank: A Listwise Approach to Collaborative Ranking. In *International Conference on Machine Learning*. 5315–5324.
[37] Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James Sharpnack. 2019. Stochastic Shared Embeddings: Data-driven Regularization of Embedding Layers. *arXiv preprint arXiv:1905.10630* (2019).
[38] Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James Sharpnack. 2019. Temporal Collaborative Ranking Via Personalized Transformer. *arXiv preprint arXiv:1908.05435* (2019).
[39] Liwei Wu, Hsiang-Fu Yu, Nikhil Rao, James Sharpnack, and Cho-Jui Hsieh. 2020. Graph dna: Deep neighborhood aware graph encoding for collaborative filtering. In *International Conference on Artificial Intelligence and Statistics*. 776–787.