RSS: Effective and Efficient Training for Sequential Recommendation using Recency Sampling*

ALEKSANDR PETROV, University of Glasgow, United Kingdom
CRAIG MACDONALD, University of Glasgow, United Kingdom

Many modern sequential recommender systems use deep neural networks, which can effectively estimate the relevance of items, but require a lot of time to train. Slow training increases the costs of training, hinders product development timescales and prevents the model from being regularly updated to adapt to changing user preferences. The training of such sequential models involves appropriately sampling past user interactions to create a realistic training objective. The existing training objectives have limitations. For instance, next item prediction never uses the beginning of the sequence as a learning target, thereby potentially discarding valuable data. On the other hand, the item masking used by the state-of-the-art BERT4Rec recommender model is only weakly related to the goal of the sequential recommendation; therefore, it requires much more time to obtain an effective model. Hence, we propose a novel Recency-based Sampling of Sequences (RSS) training objective (which is parameterized by a choice of recency importance function) that addresses both limitations. We apply our method to various recent and state-of-the-art model architectures – such as GRU4Rec, Caser, and SASRec. We show that the models enhanced with our method can achieve performances exceeding or very close to the effective BERT4Rec, but with much less training time. For example, on the MovieLens-20M dataset, RSS applied to the SASRec model can result in a 60% improvement in NDCG over a vanilla SASRec, and a 16% improvement over a fully-trained BERT4Rec model, despite taking 93% less training time than BERT4Rec. We also experiment with two families of recency importance functions and show that they perform similarly. We further empirically demonstrate that RSS-enhanced SASRec successfully learns to distinguish differences between recent and older interactions – a property that the original SASRec model does not exhibit. Overall, we show that RSS is a viable (and frequently better) alternative to the existing training objectives, which is both effective and efficient for training sequential recommender model when the computational resources for training are limited.

1 INTRODUCTION

Sequential recommender models, which are a class of recommender systems that consider the order of the user-item interactions, are increasingly popular [45]. In contrast to non-sequential recommender systems, such as Matrix Factorization based methods, these models account for the changes of individual user preferences, as well as global changes of item popularity. Ultimately, this ability to utilize sequential information helps models to make better recommendations when the order of interactions is known.

Early sequential recommender systems used Markov Chains [49, 64], however most modern ones use deep neural networks and have adapted ideas from other domains such as language modeling [16, 17, 20, 50] or image processing [51]. These deep neural models have been shown to outperform traditional non-neural methods by a significant margin [20, 50, 51, 59].

However, the most advanced sequential models, such as BERT4Rec, suffer from a slow training problem. Indeed, our experiments show that in order to reproduce the result reported in the original publication, BERT4Rec requires more than 10 hours training using modern hardware (see also our recent replicability paper on the issue [37]). This is also illustrated in Figure 1, which portrays the NDCG@10 of MF-BPR [48], SASRec [20] and BERT4Rec [50] models for different training durations on the MovieLens-20M dataset [15].

* This manuscript extends an earlier RecSys' 22 publication [36].

Authors' addresses: Aleksandr Petrov, University of Glasgow, United Kingdom, a.petrov.1@research.gla.ac.uk; Craig Macdonald, University of Glasgow, United Kingdom, craig.macdonald@glasgow.ac.uk.

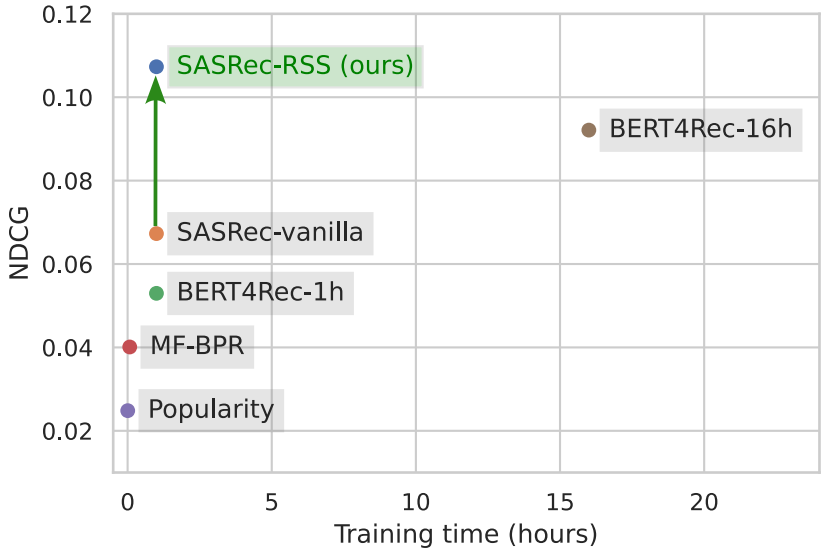


Fig. 1. The SASRec [20] model trained with our proposed training method outperforms BERT4Rec [50] on the MovieLens-20M dataset [15] and requires much less training time. SASRec-vanilla corresponds to the original version of SASRec; BERT4Rec-1h and BERT4Rec-1h are versions of original BERT4Rec implementations that have been trained for 1 hour and 16 hours, respectively.

Slow training is a problem in both research and production environments. For research, slow training limits the number of experiments that can run using available computational resources. In production, it increases the costs of using recommender systems due to the high running costs of GPU or TPU accelerators. Furthermore, slow training hinders how quickly the model can be retrained to adapt to changing user interests. For example, when a new episode of a popular TV show is released, the recommender system might still be recommending the old episode because it has not finished retraining yet. Hence, in this paper, we focus on the time-limited training of models. The main question we address in this paper is *can the training of existing sequential recommendation models be improved so that they attain state-of-the-art performance in limited training time?*

The primary components of model training can be characterized as follows: (i) the model architecture that is being trained, (ii) the training objective that defines what the model is being trained to reconstruct, and (iii) the loss function used to measure its success. All three components have a marked impact on training efficiency. For example, Hidasi et al. [17] showed that changing only the loss function can dramatically change the model performance. Raffel et al. [47] made similar findings for the model architecture and training objective for related tasks in language modeling. However, in this work, we focus on the training objective, identifying two key limitations in existing approaches, as well as an appropriate loss function for the objective.

Among the training objectives in the literature, there are two popular objectives for training sequential recommenders: *sequence continuation* and *item masking*. In this paper, we show that both of these objectives have limitations.

First, *sequence continuation* [16, 17, 51] (including its popular form, next item prediction) is probably the most intuitive and popular. Sequence continuation splits the sequence into a prefix and a suffix and then a model is trained to recover the suffix given the prefix. This objective closely matches our goal (predict the next interaction of the user) and arguably should perform the best in the case when both training data and computational resources are unlimited. However, this objective

never uses the beginning of the sequence as a training target, hence it discards potentially valuable knowledge and limits the number of training samples it can generate from a single sequence, which is specifically problematic when the datasets contains a relatively small number of sequences.

Second, in the *item masking* approach – which is used by BERT4Rec [50] – the task of the model is to recover masked items at any position in the sequence, which is a much more general and complex task than the next item prediction. We argue that this training objective is only weakly related to the end goal of sequential recommendation. Indeed, we will show that, despite leading to better effectiveness, this more general training task requires considerable training time.

These limitations of the existing approaches motivate us to design a new *Recency-based Sampling of Sequences (RSS)* approach that probabilistically selects positives from the sequence to build training samples. In our method, more recent interactions have more chances of being sampled as positives; however, due to the sampling process' probabilistic nature, even the oldest interactions have a non-zero probability of being selected as positives. The sampling probability distribution in RSS is controlled by the *recency importance function*, which may have different shapes, for example exhibiting exponential or power distributions.

Our experiments are conducted on four large sequential recommender datasets, and demonstrate the application of the proposed RSS approach upon three recent sequential recommendation model architectures (GRU4Rec, Caser and SASRec), when combined with both pointwise and listwise loss functions. We find that RSS improves the effectiveness of all three model architectures. Moreover, on all four experimental datasets, versions of RSS-enhanced SASRec trained for one hour can markedly outperform state-of-the-art baselines. Indeed, RSS applied to the SASRec model can result in a 60% improvement in NDCG over a vanilla SASRec, and a 16% improvement over a fully-trained BERT4Rec model, despite taking 93% less training time than BERT4Rec (see also Figure 1). We also find that both exponential and power importance functions result in similar optimal sampling probability distribution after fine-tuning their shape to best fit the MovieLens-20M dataset.

Moreover, we run experiments to better understand *how* RSS changes the resulting learned recommendation model. RSS is based on the idea that recent interactions are more important than the earlier ones. To check whether or not RSS helps models learn this difference in practice, we analyze how it changes the learned models with respect to interaction recency. Fortunately, some of the state-of-the-art models are based on the Transformer [53] architecture, which encodes positional information explicitly, in the form of *positional embeddings*. To understand how RSS changes the model, we perform a novel analysis by comparing the positional embeddings learned by the original and RSS-enhanced versions of the SASRec model. We show that compared to the original SASRec, the RSS-enhanced version successfully learns to distinguish recent and earlier positions.

An earlier version of this paper that appeared in RecSys' 22 [36], made the following contributions:

(1) We identify limitations in the existing training objectives used by sequential recommendation models;
(2) We propose Recency-based Sampling of Sequences (RSS), which emphasizes the importance of more recent interactions during training;
(3) We perform extensive empirical evaluations on four sequential recommendation datasets, demonstrating significant improvements over existing state-of-the-art approaches.

In this extended version of the paper we make the following additional contributions:

(4) We propose a novel methodology for analyzing position embeddings and use this methodology to empirically show that an RSS-enhanced version of SASRec learns to treat recent and earlier positions differently, whereas original SASRec fails to find any differences;

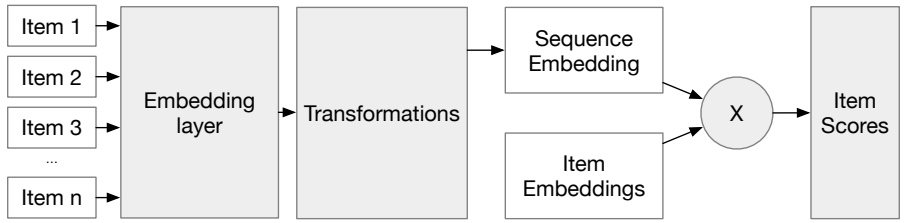


Fig. 2. Principal architecture of many sequential recommenders. This applies to GRU4rec [17], GRU4rec$^+$ [16], Caser [51] and, with minor modifications to SASRec [20].

(5) We experiment with the *power* family of importance functions and show that despite differences with the exponential family, the optimal functions from both families are mostly indistinguishable;
(6) We describe details of our version of LambdaRank [4] loss, including two modifications to the original version

The structure of this paper is as follows: Section 2 provides a background in sequential recommendation; Section 3 covers existing approaches and identifies their limitations; In Section 4 we explain Recency-based Sampling of Sequences for efficient training. Section 5 describes research questions and experimental setup; In Sections 6 & 7 we respectively provide analysis of the experiments and concluding remarks. In Appendix A we provide details of our version of LambdaRank, including the differences from the original version. Appendix B provides more details on the training of the models, such as the number of epochs and the number of training samples used for the training of each model.

2 BACKGROUND

In this section, we cover the important background for our work. In particular, Section 2.1 provides an overview of neural models used for sequential recommendation. In Section 2.2, we provide a more detailed description of Transformer architecture used by state-of-the-art sequential recommendation models.

2.1 Neural Sequential Models

In the following, we provide an overview of neural sequential recommendation models. Indeed, over the last several years, most of the next item prediction approaches have applied deep neural network models. Some of the first solutions based on deep neural networks were GRU4Rec [17] and the improved GRU4Rec$^+$ [16] (using an improved listwise loss function), which are models that use the Recurrent Neural Networks (RNN) architecture. On the other hand, Caser [51] uses ideas from computer vision; it generates a 2D “image" of the sequence using item embeddings and then applies horizontal and vertical convolution operations to that image. Another model that is based on convolution operation is NextItNet [59], which applies several layers of 1D convolutions to generate rich semantic representations of each user sequence. These models all use variations of a sequence continuation task for training, details of which we provide in Section 3.

2.2 Transformer Architecture and Positional Embeddings.

Figure 2 illustrates the principal architecture of many of the sequential recommendation models used in this work. These generate an embedding of the user’s sequence and then multiply this embedding by the matrix of item embeddings to obtain item scores. GRU4Rec, Caser, and – with

minor modifications (see Section 3) – SASRec use this architecture. Recent state-of-the-art sequential recommendation models use variations of the Transformer [53] architecture. SASRec [20] uses Transformer blocks to predict the next item in the sequence based on all previous elements. BERT-4Rec [50] adapts the well-known BERT language model [10] for the sequential recommendation task. Following the original BERT model, BERT4Rec is trained to reconstruct *masked* items that are hidden from the model during training. In particular, as both SASRec and BERT4Rec use the Transformer architecture, the only significant difference between these two models is the training scheme. Using item masking, BERT4Rec outperforms SASRec in terms of quality; however, it requires much more training time. In this work, we identify limitations in the existing training objectives, which we discuss further in Section 3. Indeed, the goal of this work is to close the gap between effectiveness and efficiency and design a new training scheme that allows matching the performance of state-of-the-art models within a limited training time.

Finally, recent advances have used graph neural networks (GNNs) for sequential recommendation [40–42, 44]. These models usually use additional information, such as cross-session connections or item attributes. In this work, we focus on a more general case of sequential recommendations without the assumption of availability of cross-session (user) information or cross-item connections; therefore, graph-based models, as well as those tailored for personalized shopping basket completion (e.g. [34, 54]), are out of the scope of this work. On the other hand, CL4SRec [57] applies data augmentation by modifying the input sequences (e.g. cropping, masking, or reordering). These augmentations are orthogonal to changing the training task and could be used together with an improved training objective. Nevertheless, we focus on the training objective for sequential models operating without the use of GNNs or data augmentation. We provide details of these training objectives in Section 3.

State-of-the-art models for sequential recommendation [20, 50, 63] are based on the *Transformer*[53] architecture, which we cover in this section. We specifically emphasize our attention on the *Positional Embeddings*—a component of the Transformer architecture, which encapsulates the positional information in the Transformer models— as we use it to analyze effects of the RSS training objective on learned models.

The key component of the Transformer architecture is the *Transformer block*, which is based on a Self-Attention mechanism [53], which allows to effectively encode item representations in the context of the other items in each sequence.

According to Vaswani et al. [53], Self-Attention is defined as:

$$Attention(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

where K, Q and V are three independent linear projections of the original item representation matrix E:

$$\begin{aligned} Q_i &= W_{qi}E_i \\ K_i &= W_{ki}E_i \\ V_i &= W_{vi}E_i \end{aligned} \tag{2}$$

Here W_{qi}, W_{ki} and W_{vi} represent three different learnable projection matrices and index i corresponds to item representations after i Transformer blocks. A Transformer block also includes a small pointwise feed-forward network, as well as residual connections and layer normalizations – standard machine learning techniques to improve model training. For more details on Transformer architecture, we refer to the original paper [53].

As can be observed from Equations (1) and (2), Attention-encoded representations do not depend on item position in the sequence. On the other hand, information about positions of items in

a sequence is crucial for sequential recommendation, as this information is the only difference between sequential and non-sequential recommendations. Therefore, information about item positions in the sequence has to be encoded in item representations themselves. To achieve this, following original the language models, recommendation models use positional embeddings:

$$E_0 = ItemEmbeddings(sequence) + PE \tag{3}$$

Here PE is a matrix of positional embeddings, which only depend on item positions, but do not depend on the items themselves. The original Transformer architecture [53] uses an *absolute* (constant) PE matrix, whose elements p_{ij} are defined as:

$$p_{ij} = \begin{cases} sin\left(\frac{i}{10000^{\frac{j}{d}}}\right); j = 2k \\ cos\left(\frac{i}{10000^{\frac{j-1}{d}}}\right); j = 2k+1 \end{cases} \tag{4}$$

where d is the size of the embeddings.

Another approach employed by later Transformer-based models [10, 46] is to use a *learnable* PE matrix; in that case we learn the positional embeddings as just one of the model parameters. According to Tunstall et al. [52, Chapter 3], absolute positional embeddings are preferable when the dataset size is small, whereas learnable embeddings are a good choice with large datasets. Many sequential recommender systems based on transformers use learnable position embeddings [20, 50].

In particular, SASRec – one of the architectures which we adapt in this work – also uses learnable position embeddings. This means that it is possible to analyse these learnable embeddings to determine the effect of RSS training objective on different positions in the sequence. In Section 4.4 we describe our methodology of using learned positional embeddings to analyze the effects of a training objective on positions in the sequence; the results of the experiments using this methodology are covered later in Section 6.6.

With this, we conclude the background review, which is necessary to understand our work. In the next section, we dive deeper into the training of the sequential recommender systems and identify the limitations of the existing training objectives, which we then solve with RSS in Section 4.

3 TRAINING SEQUENTIAL RECOMMENDATION MODELS

Consider a set of users U and items I. Each user $u \in U$ has a sequence of interactions $s_u = \{i_{u_1}, i_{u_2}, i_{u_3} ... i_{u_n}\}$ where items $i_{u_\tau} \in I$ are ordered by the interaction time. The *next item prediction task* is defined as follows: given a sequence s_u, rank the items from I, according to their likelihood of being the sequence continuation $i_{u_{n+1}}$. This task corresponds to *Leave One Out* evaluation - hold out the last element from each user's interactions sequence and then evaluate how well a model can predict each held-out element.

As mentioned in Section 2, the best models for the next item prediction task are based on deep neural networks. Generally speaking, their training procedure consists of iterations of the following steps: (1) Generate a batch of training samples, each with positive and negative items; (2) Generate predictions, using the model; (3) Compute the value of the loss function; (4) Update model parameters using backpropagation.

We aim to improve the training of existing models, so step 2 is not within the scope of our work. Backpropagation (step 4) – e.g. through stochastic gradient descent – is a very general and well-studied procedure, and we follow the best practices used by the deep learning models, details of which we describe in Section 5. This leaves us with two essential parts of model optimization - generation of the training samples and the loss function. These two parts are not independent: a loss function designed for one training task does not always fit into another. For example, BPR-max

loss (used by GRU4Rec$^+$ [16]) has an assumption of only one positive item per training sample and therefore is not applicable to a sequence continuation with multiple positives task, as used by Caser. Hence, a new training task requires the selection of an appropriate loss function. We further discuss some possible choices of the loss functions for our proposed method later in Section 4.3. In the following, we describe existing approaches to generate training samples and identify their limitations, a summary of which we provide in Section 3.2.

3.1 Generation of Training Samples

A training sample for a sequential model consists of three parts - the input sequence, positive items, and negative samples. Sequential recommender models [16, 17, 20, 51] treat ground truth relevance as a binary function; by definition, every non-positive item is negative. In practice, to make the training more tractable, most models only consider samples of negative items, identified using techniques such as random sampling [20, 48], in-batch negatives [17], or the negatives with highest scores [58]. This work focuses on constructing positive samples. Negatives sampling approaches are orthogonal to positive sampling and can be applied independently. We do not use negative sampling in our work and leave improvement of our method via negative sampling to future research. In the remainder of this section, we describe positive sampling strategies for sequential recommendations. Figure 3 illustrates sequence continuation and item masking, the most commonly used strategies, which we discuss in turn below.

Matrix factorization methods use a straightforward *matrix reconstruction* training objective: for each user u and item i, the goal of the model is to estimate whether the user interacted with the item. This goal leads to a simple procedure for generating training samples - the training algorithm samples (user, item) pairs as inputs and assigns labels for the pairs based on interactions. A classic model that uses matrix reconstruction is Bayesian Personalized Rank (BPR) [48], which we use as one of our baselines. The main disadvantage of matrix reconstruction is that it does not consider the order of the interactions, and therefore sequential recommendation models can not use it.

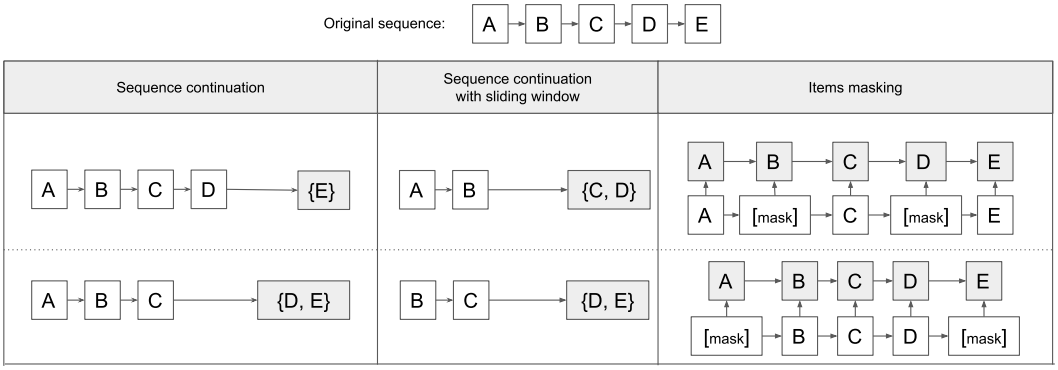


Fig. 3. Training sample generation strategies used in existing models. White boxes represent model inputs, and filled boxes represent model outputs. In Sequence Continuation, the sequence is split into two parts, with the aim of predicting whether or not an item belongs to the second part based on the sequence of elements in the first part. In Sequence Continuation with a sliding window, we first generate shorter sub-sequences from the original sequence and then apply the sequence continuation method. In item masking, some elements are removed and replaced with a special "[mask]" value, with the aim of correctly reconstructing these masked items.

In the *sequence continuation* training objective, training samples are generated by splitting the sequence of interactions into two consequent parts:

$$s = \{i_1, i_2, i_3..i_n\} \mapsto \begin{cases} s_{input} = \{i_1, i_2, i_3, ...i_{n-k}\}; \\ s_{target} = \{i_{n-k+1}, i_{n-k+2}, ..i_n\} \end{cases}$$

where k is a hyperparameter. The model uses s_{input} as the input sequence, and assigns label 1 to the positive items from $i_+ \in s_{target}$ and label 0 to the negative items $i_- \notin s_{target}$. If k is equal to 1, the sequence continuation task turns into the next item prediction task, which matches the the end goal of sequential recommender systems.

Using sequence continuation in its basic form, we can produce precisely one training sample out of a single sequence of interactions. Some models (e.g. Caser [51]) use the sliding window approach to generate more than one sequence - which generates shorter subsequences out of a whole sequence and then creates training samples out of these shorter subsequences. The sliding window approach allows the model to generate up to $n-1$ training samples from a sequence of n interactions. However, shorter sequences only allow to model short-term user preferences, and researches have to find a balance between the number of generated samples and the maximum length of the sequence [51]. GRU4rec, GRU4rec$^+$, and Caser models use variations of the sequence continuation task for training. The *sequence shifting* training task used by SASRec and NextItNet [59] is essentially a version of sequence continuation: it trains the model to predict the target sequence, which is shifted by one element compared to the input. they These models predict the second element of the input sequence by the first, third by the first two, etc.. When these models predict the j^{th} item in the output, they only have access to the first $(j-1)$ elements of the input so that this shifted sequence prediction task essentially is n independent sequence continuation tasks. Figure 4 graphically illustrates shifted sequence task and equivalent sequence continuation training samples. Thus, the main limitation of sequence continuation is that it only generates a small number of training samples out of a single sequence, and the items in the first part of the user's sequence never have a chance to be selected as a target, which means that the recommender system is unlikely to learn how to recommend these items, even though they may be relevant for some users. We refer to this limitation as Limitation L1.

In contrast to earlier neural sequential models, BERT4Rec [50] uses an *item masking* training objective, which it inherited from the original BERT model. In BERT, the idea is to hide some terms from the sentence and then ask the model to reconstruct these hidden elements. Similarly, in

(a) Shifted Sequence training sample

(b) Sequence Continuation training samples equivalent to using Shifted Sequences

Fig. 4. Equivalence of Shifted Sequence and Sequence Continuation training tasks. In Shifted Sequence, the model is trained to predict its output shifted by one element to the left. The models that use this training task only use elements in positions $0..i$ to predict i^{th} output. This is equivalent to n sequence continuation tasks, where n is the length of the sequence.

BERT4Rec, some items in the sequence are masked, and the model is retrained to recover these items. The target sequence, in this case, exactly matches the original sequence (without masking):

$$s = \{i_1, i_2, i_3, i_4, ..i_n\} \mapsto \begin{cases} s_{input} = \{i_1, [mask], i_3, [mask], ...i_n\}; \\ s_{target} = \{i_1, i_2, i_3, i_4, ..i_n\} \end{cases}$$

This approach generates up to 2^n training samples out of a single training sequence of length n. BERT4Rec does not mask more than τ percent of items in a sequence, where τ is a hyperparameter; however, it still generates many more training samples compared to the single training sample generated from a sequence under sequence continuation. As Sun et al. [50] showed, more training ensures to avoid overfitting and achieves better performance compared to other models with similar architecture.

However, we argue that the main disadvantage of the item masking approach is that it is weakly related to the next item prediction task. To make a prediction, BERT4Rec adds the [mask] element to the end of the input sequence and tries to reconstruct it; so that training and evaluation samples have a different distribution. The model must learn how to solve the evaluation task (reconstruct the last item in the sequence) as part of a much more general and more complicated task (reconstruct any item in the sequence). BERT4Rec adds a small proportion of training samples with only the last element masked to address this mismatch, but the consequence is still a substantially more complicated training task and longer time to converge compared to the models that use sequence continuation. We refer to this problem of weak correspondence to the original task as Limitation L2.

3.2 Summary of Limitations

We described the two main training objectives used by sequential recommendations approaches: sequence continuation (including its variations, shifted sequence prediction, and sliding window) and item masking. Indeed, as argued above, both of these training objectives have their limitations, which we summarize as follows

- **L1** Sequence continuation can only generate a small number of training samples from a single training sequence. This allows training to be performed relatively quickly, but performance of these models is lower compared to a state-of-the-art model such as BERT4Rec.
- **L2** Reconstruction of masked items is a very general task, which is loosely connected to the sequential recommendation task. Using this task, models can reach state-of-the-art performance, but model training can take markedly longer than other training objectives.

In the next section, we introduce *Recency-based Sampling of Sequences*, a novel training task that addresses these limitations and discuss possible choices of the loss function for this training task.

4 RSS: RECENCY-BASED SAMPLING OF SEQUENCES

As shown in Section 3, to train a model we need to have a training task and choose a loss function that matches the task. Hence, the training task and the loss function are both essential parts of our solution. In this section, we introduce both training objective (Section 4.1), describe two families of recency importance functions (Section 4.2) and choice of loss function. (Section 4.3). Later, in Section 4.4, we introduce a concept of position similarity matrix and describe how it can be used to analyze effect of RSS on trained models. Section 4.5 provides a summary of the salient characteristics of RSS.

4.1 The RSS Training Objective

Recency-based Sampling of Sequences (RSS) is a training objective that is closely related to the sequential recommendations and allows the model to generate many training samples out of a single user sequence simultaneously. To address the limitations of existing training objectives described in Section 3.2, we first outline the principles used to design our training task:

P1 Each element in a sequence can be selected as the target; multiple items can be selected as a target in each training sample. Using this principle, we match the main advantage of the item masking approach - generating up to 2^n training samples out of each user sequence. This principle addresses Limitation L1.

P2 More recent training interactions in a sequence better indicate the user's interests, and hence these are more realistic targets. User interests change over time, and one of the main advantages of sequential recommender systems is taking these changes into account. Therefore, the methods that rely on this principle will retain a close connection to sequential recommendations. This principle addresses Limitation L2.

In our proposed training objective, to follow these two principles, we use a *recency importance function*, $f(k)$, that is defined for each position $0 \,..\, n-1$ in the sequence of the length n and indicates chances of each position to be selected as a target: the probability of an item at position k of being selected as a positive is proportional to the value of $f(k)$. $f(k)$ must exhibit the following properties:

(i) $f(k)$ is positive:

$$f(k) > 0 \tag{5}$$

(ii) $f(k)$ is monotonically growing:

$$f(k) \leq f(k+1) \tag{6}$$

This first property corresponds to Principle P1 and defines that the likelihood of each item to be selected as a target are positive. The second property corresponds to Principle P2, and ensures that more recent items have higher or equal chances to be selected as a target.

To generate a training sample, we first calculate c - how many target items we want to sample. Following BERT4Rec, we define a parameter τ that controls the maximum percentage of items that can be used as targets and then calculate c via multiplying τ by the length of the sequence. We then randomly sample, with replacement, c targets from the sequence, with the probability of being sampled, $p(i)$, proportional to the value of a recency importance function, $f(i)$:

$$p(i) = \frac{f(i)}{\sum_{j=0}^{n-1} f(j)} \tag{7}$$

We generate the input sequence to the model by removing targets from the original sequence. The full procedure is described in Algorithm 1. We now describe two families of recency importance functions that have the required properties.

4.2 Recency Importance Functions

In this Section we describe properties of two families of recency importance functions, which can be used with RSS (i. e. they satisfy the requirements described by Equations (5) and (6)): *exponential importance* and *power importance*

Algorithm 1 Recency-based Sampling of Sequences

Input: *sequence* - a sequence of interactions; τ - maximum percent of target items; f - recency importance function

Output: *input* is a generated input sequence for the model; *target* is a set of sampled positive items

function RecencySequenceSampling($sequence$, τ, f)
  $sampledIdx \leftarrow set()$
  $n \leftarrow length(sequence)$
  $c \leftarrow max(1, int(n * \tau))$
  $prob \leftarrow Array[n]$
  $prob[i] \leftarrow \frac{f(i)}{\sum_{j=0}^{n-1} f(j)}$ **for** i **in** [0,$n-1$]
  $sampledIdx \leftarrow random.choice(range(0..n-1), c, prob)$
  $input \leftarrow list()$
  $target \leftarrow set()$
  for $i \leftarrow 0, n-1$ **do**
    if $i \in sampledIdx$ **then** $target.add(sequence[i])$ **else** $input.append(sequence[i])$
  end for
  return $input, target$
end function

We assume that function $random.choice(a, c, p)$ is an equivalent of the $numpy.random.choice$ function from the numpy package. It iteratively samples c samples from collection a, where the probability of each item i of being sampled equals $p[i]$ at each stage, with replacement.

4.2.1 Exponential Importance. Our first proposal for a recency importance function that has the required properties is the exponential function:

$$f_{exp}(k) = \alpha^{n-k} \tag{8}$$

where $0 < \alpha \leq 1$ is a parameter that controls importance of the recent items in the sequence and n is the sequence length. If $\alpha = 1$, then each item has equal chances of being sampled as a target, and Recency-based Sampling of Sequences becomes similar to the item masking approach (but without providing the positions of masked items) or to the matrix reconstruction approach, where items are sampled uniformly from the sequence. If α is close to zero, items from the end of the sequence have a much higher chance of being sampled, and therefore RSS becomes equivalent to the sequence continuation task. Figure 5 provides an example of the recency importance (for $\alpha = 0.8$) and the generated samples.

Koren et al. [23] recently used a similar exponential function for modeling temporal dynamics in neighborhood-based recommendation methods. Indeed, the authors successfully model a user-item interaction as a weighted sum of the other interactions of the same user, with the weight proportional to the exponential function:

$$e^{-\beta_u \Delta t} \tag{9}$$

where Δt is the time interval between the modeled and a known interaction, and β_u is a user-specific learnable parameter. In contrast to Koren et al. [23], we use the recency importance function for the target items selection instead of the explicit interaction similarity modeling. Nevertheless, inspired by their work, we use the exponential importance as the main approach throughout the paper.

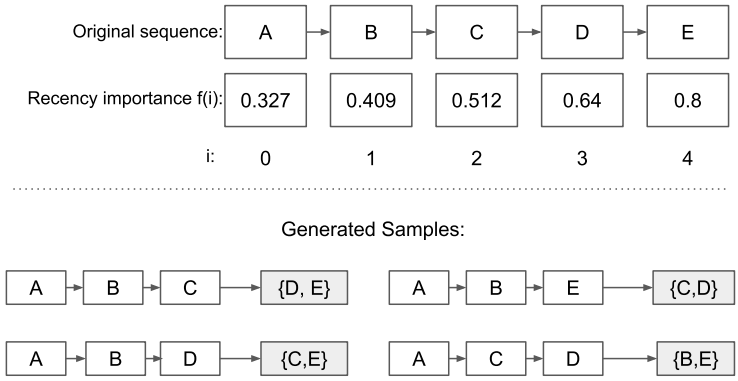
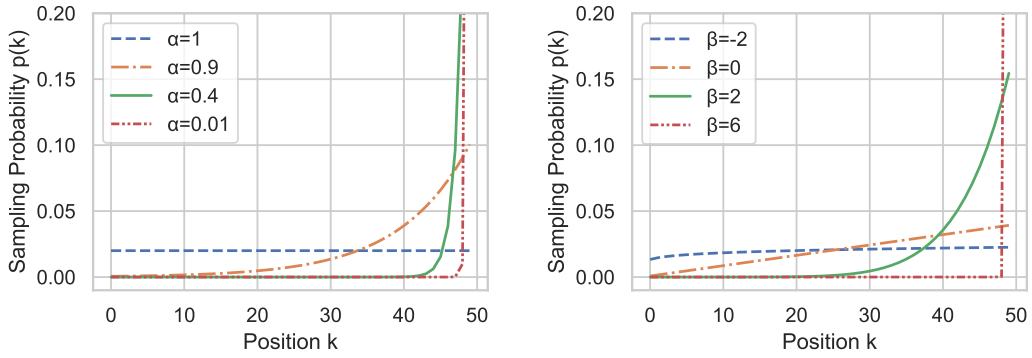


Fig. 5. Recency-based Sampling of Sequences. The beginnings of the sequences remain largely unchanged, whereas elements from the end of the sequence are chosen as positive samples more frequently.



(a) Exponential Importance
$f_{exp}(k) = \alpha^{n-k}$

(b) Power Importance
$f_{pow}(k) = \left(\frac{k+1}{n+1}\right)^{e^{\beta}}$

Fig. 6. Sampling probability distributions produced by exponential and power recency importance functions. Sequence length n is set to 50.

Figure 6a illustrates the sampling probability distributions generated by the exponential importance function for different values of the importance parameter α. As the figure shows, with the exception of the situation when $\alpha \approxeq 1$ (which corresponds to uniform sampling), exponential importance produces probability distributions that are very strongly skewed towards the most recent items. For example, even with $\alpha = 0.9$, the probability of sampling the item at position 10 in a sequence of 50 interactions is less than 0.002. Overall, the exponential nature of the function means that the probability of sampling always decays faster than linearly (wrt. to position k) for nearly all values of α. On the other hand, Ludewig and Jannach [31] successfully deployed a linear decay of position importance weights in the V-SKNN model. Similarity to the method proposed by Koren et al. [23] (mentioned above), V-SKNN uses these weights for explicit item-item similarity modeling rather than for computing target item sampling probabilities; therefore, it differs from RSS. However, the fact that V-SKNN used a linear decay of position similarity motivates us to investigate

broader set recency importance functions, which are capable of generating linear and sub-linear decay of recency importance. Hence, we also experiment with a *power* importance function, which we describe in the next section.

4.2.2 Power importance. Another family of importance functions we use for our experiments is the *power importance* function:

$$f_{pow}(k) = \left(\frac{k+1}{n+1}\right)^{e^{\beta}} \tag{10}$$

where n is the sequence length, k is the position and β is a hyperparameter. In this function, the position of the item in the sequence is raised to a constant power (controlled by the hyperparameter β). For example, when $\beta = 0$, this results in a linear function, when $\beta = -\infty$ the importance becomes constant (meaning that there is an equal chance of sampling any item in the sequence).

Figure 6b illustrates sampling probability distributions generated by the power importance functions with different values of the parameter β. Comparing the figure with Figure 6a, we can see that in contrast with the exponential importance, power importance can generate linear (when $\beta = 0$) and sub-linear (when $\beta < 0$) shapes of probability distributions.

Note that, because n and β are constants, the set of sampling probability distributions produced by this family is equal to the one produced by a simpler function:

$$\hat{f}_{pow}(k) = (k+1)^{\tau} \tag{11}$$

where $\tau = e^{\beta}$. Note that this simpler form omits the normalization coefficient $\frac{1}{n+1}$, because it is reduced anyway while computing probability distribution in Equation (7). However, this normalization is important in practice: without the normalization $\hat{f}_{pow}(k)$ becomes very large and numerically unstable even when k and τ have modest values. For example, for $k = 50$ and $\tau = 400$ (realistic numbers we use in our experiments), $\hat{f}_{pow}(k) = 51^{400} \approx 10^{683}$. This number is larger than the maximum value that can be represented with a standard *float32* data type and therefore causes errors during computations. By keeping the normalization, we bring the value of the function into the $[0..1]$ interval and therefore help to avoid problems with infinities during computations.

The substitution of the parameter $\tau = e^{\beta}$ in Equation (11) helps to squeeze the area of the interesting hyperparameter values to a more symmetric and smaller interval. In particular, as we mention in Section 4.2.1, we are specifically interested in the cases when recency importance decays sub-linearly because faster-than-linear cases are covered by the exponential importance function. Without the substitution, we have following cases for the behaviour of $\hat{f}_{pow}(k)$:

$$\begin{cases} \tau < 0 \rightarrow \text{out of the scope (violates the monotonic growth requirement (Equation 6))} \\ \tau = 0 \rightarrow \text{uniform sampling probability} \\ \tau \in (0,1) \rightarrow \textbf{sub-linear decay of sampling probability} \\ \tau = 1 \rightarrow \text{linear decay of sampling probability} \\ \tau > 1 \rightarrow \text{faster-than-linear decay of sampling probability} \end{cases} \tag{12}$$

From these cases we can see that uniformly sampling of τ from the allowed range $\tau \in [0..+\infty]$ will almost certainly be sampled from the less interesting area of the faster-than-linear growth (with a probability of 1). Instead, by making the substitution $e^{\beta} = \tau$, we obtain a simpler and symmetric set of cases for the behavior of $f_{pow}(k)$:

$$\begin{cases} \beta < 0 \rightarrow \textbf{sub-linear decay of sampling probability} \\ \beta = 0 \rightarrow \text{linear decay of sampling probability} \\ \beta > 0 \rightarrow \text{faster-than-linear decay of sampling probability} \end{cases} \tag{13}$$

Thus, a randomly sampled β has equal chances of producing faster-than-linear and sub-linear probability decay. In practice we chose β from the interval (-2..6): as Figure 6b shows, for sequences of length 50, $\beta = -2$ produces a sampling probability distribution close to uniform, whereas for $\beta = 6$ the probability of sampling the last item becomes close to one and the probability of sampling earlier items is close to zero; RSS, therefore, becomes similar to sequence continuation.

In summary, we have proposed *exponential importance* and *power importance* families of importance functions, which can produce a broad set of shapes, including exponential, linear, and sub-linear shapes. Sections 6.3 and 6.4 cover our experiments with these two families of importance functions. In particular, in Section 6.4 we show that optimal shapes generated by both these families are very similar to each other.

4.3 Loss Functions for RSS

The second important component of the training procedure is the loss function. Loss functions for recommender systems can be generally divided into three categories - *pointwise* (optimize the relevance estimation of each item independently), *pairwise* (optimize a partial ordering between pairs of items) and *listwise* (optimize the recommendations list as a whole) losses [29]. RSS works with all types of loss functions that support multiple positive samples within each training sample.

GRU4rec$^+$ [16] showed the advantages of applying a listwise loss function above pointwise and pairwise methods, however the Top-1-max and BPR-max losses introduced in that paper have an assumption that there is only one positive item within each training sample. Instead, we use LambdaRank [4] (denoted λRank), another listwise optimization loss function. λRank has been widely deployed in training learning-to-rank scenarios [6, 39] for web search. Similarly, λRank has been shown to be advantageous for recommender tasks [28], for example when applied to Factorization Machines [58] or Transformer-based sequential models [38].

λRank [4] uses λ-*gradients* instead of objective function gradients in the gradient descent method. According to Burges [4], the λ-gradient for an item $i \in I$ is defined as follows:

$$\lambda_i = \sum_{j \in I} |\Delta NDCG_{ij}| \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} \tag{14}$$

where s_i and s_j are predicted scores, $\Delta NDCG_{ij}$ is the change that would be observed in an NDCG metric items i & j were swapped, and σ is a hyperparameter, defining the shape of the sigmoid, typically set to 1. Burges [4] used λRank to build one of the most successful learning-to-rank algorithms *LambdaMART*, which is based on gradient boosting trees. Versions of LambdaMART still produce state-of-the-art results for the learning-to-rank task [18, 39].

There are multiple available implementations of LambdaMART. Qin et al. [39] have shown that a version of LambdaMART implemented in a popular LightGBM library [21] performs better than other available implementations, and therefore we matched our implementation of λRank with LighghtGBM's implementation. On analysis of the LightGBM source code, we found that it uses a normalized version of the λ-gradients (compared to Equation (14)). To keep our version consistent with the best available implementation, we include these modifications into our version of λRank. We provide more details on these modifications in Appendix A.

In addition to λRank, we also experiment with the pointwise Binary Cross-Entropy (BCE), following [20, 51], to investigate the effect of the listwise loss and the necessity of both training objective and the loss function in our solution.

4.4 RSS and Positional Embeddings in Transformer-based models.

In this section we describe the methodology of analyzing the effects of RSS on the learned model using positional embeddings.

The main idea of RSS is that recent interactions are more important to the model compared to the older ones and therefore the model should treat them differently. To analyze whether or not this happens in practice, we need a mechanism that allows us to understand the learned differences between different positions in a sequence. Fortunately, the Transformer architecture (as used by SASRec [20]) provides this mechanism in the form of positional embeddings, details of which we describe in Section 2.2.

Our goal is to examine whether or not RSS helps the model distinguish between recent and earlier positions. To achieve this goal, we need to measure *position similarity* between different positions learned by the model. Intuitively, we would like the learned similarities to exhibit the following properties:

- **Pr.1** Similarities between **high** nearby positions, as the nearby interactions are likely to be related to each other.
- **Pr.2** Similarities between distant positions are **low**; as distant interactions are unlikely to be related to each other.
- **Pr.3** For recent positions, similarity decays faster with the distance between positions than for the earlier positions. This means that the relative order of recent interactions is more important than the relative order of earlier interactions.

Property Pr.3 can be written as

$$similarity(i_r, i_r + \Delta_i) < similarity(i_e, i_e + \Delta_i) \tag{15}$$

where i_r corresponds to recent positions and i_e correspond to earlier positions.

SASRec explicitly separates interaction representations into item embeddings and position embeddings (see Equation (3)), allowing us to focus solely on the position representations – indeed, as the item embeddings are independent of the positions, we can therefore exclude them from our analysis of position similarities.

After summation with the item embeddings, SASRec uses the position embeddings as the input to linear projections. Therefore it makes sense to use cosine similarity of positional embeddings as a means to measure the learned similarity between two positions in the sequence.

Formally, we define the learned position similarity matrix S, which elements $s_{i,j}$ correspond to the learned similarity between positions i and j and are defined as:

$$s_{i,j} = \frac{e_i \cdot e_j}{\|e_i\| \|e_j\|} \tag{16}$$

where e_i and e_j are positional embeddings learned by SASRec for positions i and j. Based on the definition, matrix S is symmetric with respect to the main diagonal, i.e.:

$$s_{i,j} = s_{j,i} \tag{17}$$

However, the symmetry with respect to the secondary diagonal is an undesirable property, as it violates Property Pr.3. Indeed, let's assume the symmetry with respect to the secondary diagonal:

$$s_{i,j} = s_{n-j,n-i} \tag{18}$$

Let's denote $\Delta_i = j - i$, so

$$s_{i,i+\Delta_i} = s_{n-i-\Delta_i,n-i} \tag{19}$$

Equation (19) can be also written as

$$s_{i,i+\Delta_i} = s_{i',i'+\Delta i} \tag{20}$$

where $i' = n - i - \Delta_i$. If i is a recent position, ($i \approx n$), then i' corresponds to a early position ($i' \approx 0$). In that case Equation (19) contradicts Equation (15), which requires $s_{i,i+\Delta_i} < s_{i',i'+\Delta i}$. This means that a symmetric similarity matrix with respect to the secondary diagonal violates our desirable Property Pr.3. Ideally, we would like to avoid this kind of symmetry.

Overall, the position similarity matrix S allows us to analyze how the learned model treats interactions in different positions. For example, the original SASRec model is trained to predict original input shifted by one element. In this training task every position in the sequence has equal importance and therefore we expect that the similarity between positions only depends on the distance between positions, i.e.:

$$s_{i,j} \approx g(|i - j|) \tag{21}$$

which will lead to symmetry with respect to the secondary diagonal:

$$s_{i,j} \approx g(|i - j|) = g(|(n - j) - (n - i)|) \approx s_{n-j,n-i} \tag{22}$$

hypothesize that RSS training objective helps the model to avoid this undesirable symmetry. Indeed, by the design of the training objective, recent positions and their relative order are more important earlier positions in the sequence and their order, and therefore we expect that the similarity $s_{i,j}$ between positions i and j depends on both absolute values of position and their relative distance between them:

$$s_{i,j} \approx g(i, |i - j|) \tag{23}$$

In Section 6.6 we analyze position similarity matrices of SASRec-vanilla and SASRec-RSS to show in practice that similarity matrices of SASRec-RSS indeed exhibit all three desirable properties on 3 out of 4 experimental datasets, whereas the positional similarity matrices of the original SASRec model do not exhibit Property Pr.3 on all datasets.

4.5 Summary

Overall, RSS is a novel training objective based on a probabilistic sampling of target items, with more probability assigned to recent items. Our motivation for the probabilistic target sampling objective encodes two principled intuitions for the sequential recommendation. RSS is model- and loss-agnostic: it can be used with various model architectures and loss functions. RSS is parametrized by the recency importance function, which defines the probability decay. Two examples of recency function families include exponential importance and power importance.

RSS only affects model training and leaves other model characteristics, such as the number of parameters or model throughput, unchanged. Hence, in the next section, through detailed experimentation, we investigate the effects of RSS on model training.

5 EXPERIMENT SETUP

In the following, we list our research questions (Section 5.1), our experimental datasets (Section 5.2), the recommender models on which we build, and our comparative baselines (Section 5.3), and finally, evaluation details (Section 5.4).

Table 1. Datasets we use for experiments.

Name	Users	Items	Interactions	Average length	Median length	sparsity
Booking.com	140746	34742	917729	6.52	6	0.999812
Gowalla	86168	1271638	6397903	74.24	28	0.999942
Yelp	287116	148523	4392169	15.29	8	0.999897
MovieLens-20M	138493	26744	20000263	144.41	68	0.994600

5.1 Research Questions

Our experiments aim to address the following research questions:

RQ1 Does Recency-based Sampling of Sequences (RSS) help for training sequential recommendation models compared to sequence continuation?
RQ2 Does a listwise λRank loss function benefit RSS training?
RQ3 What is the impact of the recency importance parameter α in the exponential recency importance function (Equation (8)) of RSS?
RQ4 What is the effect of the recency important function shape in RSS?
RQ5 How do RSS-enhanced models compare with state-of-the-art baselines?
RQ6 What is the effect of RSS on the positional embeddings learned by the SASRec model?

5.2 Datasets

Our experiments are performed on four large-scale datasets for sequential recommendation:

MovieLens-20M [15] is a movie recommendation dataset, and is popular for benchmarking sequential recommenders [8, 11, 27, 32, 33, 42, 50, 56, 62]. Note that MovieLens-20M is a rating dataset where users rate movies with stars, however following common practice [27, 33, 50] we consider all ratings as positive interactions. However, MovieLens-20M timestamps correspond to the time when ratings were provided rather than when the items were consumed, so the task is best described as "next movie to rate" rather than "next movie to watch". Nevertheless, as versions of the MovieLens dataset are used in both well-cited [19, 20, 50] and recent [43, 60, 61] sequential recommendation papers, we conclude that it is well suited for the the problem and it is important to include it as one of our benchmarks.

Yelp[1] is a business reviews dataset. It is another popular dataset for sequential recommendations [2, 3, 35, 43, 55, 63]. As for MovieLens-20M, we consider all user reviews as positives.

Gowalla [7] contains user check-ins to a location-based social network. This dataset contains a large number of items (more than 10^6) and is very sparse: it has only 0.0058% out of all possible user-item interactions.

Booking.com [13] is a travel destination dataset. Each interaction sequence in this dataset represents a single multi-city trip of a user. In contrast to other types of recommendations, such as movies or books, multi-city trips have a strong sequential nature. Indeed, for example, if a user is making a road trip by car, there could be only one or two neighboring cities where the user can stop, and hence all other more distant items are non-relevant. This strong sequential nature could be problematic for RSS, as it contradicts Principle P1, which says that any item in the sequence can be selected as a relevant target for the preceding items.

Following common practice [20, 50, 63], we discard cold-start users with fewer than 5 interactions from each dataset. Table 1 reports the salient statistics of the four datasets. As the table shows,

[1] https://www.yelp.com/dataset

the experimental datasets are quite distinct in terms of sequence length: median sequence length varies from 6 in the Booking.com dataset to 68 in the MovieLens-20M dataset. Indeed, our choice of datasets allows testing of RSS performance in settings with different sequence lengths.

5.3 Models

5.3.1 Experimental Architectures. RSS is a training objective that can be used with a large variety of model architectures. We identify three large groups of model architectures that can be used with RSS: Recurrent Neural Network-based models, Convolutional Neural Network-based models, and Attention-based models. In each group, we select a well-cited representative architecture. Overall, we experiment using RSS with three recent model architectures for sequence recommendation:

(1) **Recurrent Neural Networks:** *GRU4Rec* [17] is a sequential recommender architecture based on recurrent networks;
(2) **Convolutional Neural Network:** *Caser* [51] applies a convolutional neural network structure for sequential recommendation. For our experiments, we use the basic architecture described in [59];
(3) **Attention network:** *SASRec* [20] is a sequential recommendation architecture based on Transformer [53]. The original implementation of SASRec is trained as a sequence-to-sequence model, however only the final element from the target sequence is used at inference time. In order to match our common training framework and train the model with the RSS training objective, we ignore all outputs of the architecture except the final one. This is a notable change in the training process, because the original SASRec computes its loss over all outputs. To make sure that this change does not lead to significant quality degradation we include the original version of SASRec as a baseline (see Section 5.3.2).

We implement these architectures using the TensorFlow version 2 [1].[2] Note that for our experiments we reuse only the architectures of these models and not the training methods or hyperparameters. Indeed, because our goal is to research the impact of the training task, the appropriate training parameters may differ from the original implementation.

We implement RSS and sequence continuation training objectives on each of the three experimental architectures. We do not apply an item masking training objective with these architectures: item masking assumes that a model produces a scores distribution per masked item, which is not compatible with those architectures; however, as discussed below, we include BERT4Rec as an item masking baseline.

For our experiments, we set common training parameters for all model architectures, following the settings in [20]. In particular, we set the size of the item embeddings to be 64, we use the Adam optimizer, applying the default learning rate of 0.001 and following SASRec [20] we set the β_2 parameter, which controls the second moment decay rate in Adam, to 0.98[3].

We also fix the length of the user sequences to 50 items for all four following [20]. If a user has less than 50 interactions, we add special "[pad]" interactions at the beginning of the sequence. If the user has more than 50 interactions, we take the last 50 interactions from the sequence. Figure 7 illustrates this padding/truncation scheme visually. The scheme ensures that the user's most recent interaction is always located in the rightmost position (position with index 49 in our case), and padding is added to the beginning of the sequence. We select target items for both RSS and Sequence Continuation before applying padding/truncation.

[2] The code for this paper is publicly available in a joint repository with our BERT4Rec replicability paper [38] and can be found at https://github.com/asash/bert4rec_repro [3] Note that the datasets used in the SASrec paper [20] are different from the ones we use in our experiments. However, the datasets used in [20] represent a wide range of data types, including e-commerce, games, and movie recommendations, and therefore we use these hyperparameters without changing them.

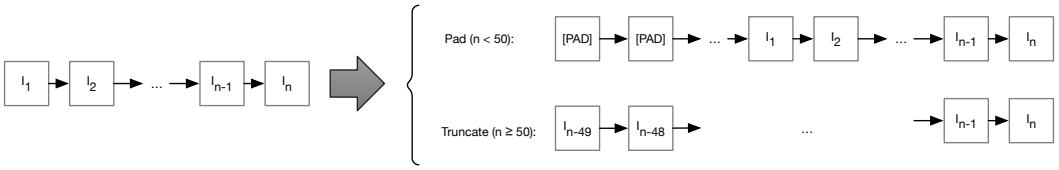


Fig. 7. Padding/Truncation scheme. We use left padding in experiments, ensuring that the rightmost input to the model is always the most recent interaction. Note that we sample target items (using RSS or Sequence Continuation) *before* applying padding or truncation.

Following BERT4Rec [50], we set the maximum percentage of a sequence to sample, τ, to 0.2. Except where otherwise noted, we deploy the exponential recency function, and set the recency parameter α to 0.8. Finally, in order to estimate the performance of the models under limited training time, we fix the training time of all models to 1 hour (we provide details on the amount of training data used for training of each model within the 1-hour limit in Appendix B). Experiments are conducted using 16-cores of an AMD Ryzen 3975WX CPU, 128GB of memory, and an NVIDIA A6000 GPU.

5.3.2 Baselines. In order to validate that using Recency-based Sampling of Sequences is possible to achieve performance comparable to state-of-the-art recommender models, we compare with a selection of popular and state-of-the-art recommenders. We use the following models non-neural models as baselines: (i) *Popularity* - the most popular items in the dataset; (ii) *MF-BPR* - Matrix factorization with BPR Loss [48]. We use the implementation of this recommendation model from the popular LightFM library [25].

We also use two Transformer-based models as state-of-the-art baselines: (i) *SASRec-vanilla* - the original version of SASRec recommender [20], a Transformer-based model that uses a shifted sequence task, described in Section 3.1. To make the comparison fair with the RSS-enhanced variant, we limit the training time of this model to 1 hour; (ii) *BERT4Rec* is another Transformer-based model [50] based on the BERT [10] architecture. BERT4Rec has been shown to outperform other traditional and neural architectures and has been used as a strong baseline in a number of recent works (e.g. [22, 26, 30, 39]). In our recent replicability study [37], we have also shown that despite the existence of many later models, BERT4Rec still exhibits state-of-the-art performance.

We use two versions of this model: *BERT4Rec-1h* denotes where the training time of BERT4Rec is limited to 1 hour, to allow a fair comparison in a limited-time setting; *BERT4Rec-16h*, where training time is limited to 16 hours in order to compare the performance of our approach with the state-of-the-art model. The original BERT4Rec publication [50] does not report the required amount of training, but we find empirically that reproducing the reported results takes around 16 hours on our hardware [37]. We set the other parameters of BERT4Rec following the original paper [50]. In particular, we use two transformer layers and masked 20% of each training input sequence.

In contrast with other baselines, BERT4Rec calculates a score distribution across all items in the catalog for each element in the sequence, whereas other baselines calculate a single distribution of scores per sequence. This means that BERT4Rec requires $O(N)$ more memory per training sample for storing output scores and ground truth labels compared to other baseline models. This makes training the original implementation of BERT4Rec infeasible when a dataset has too many items. Indeed, the original BERT4Rec publication [50] only reports results on relatively small datasets with no more than 55000 items and our own attempts to train BERT4Rec on a large Gowalla dataset with more than 1 Million items failed because of memory and storage issues (see also Section 6.5).

Table 2. Comparing sequence continuation with Recency-based Sampling of Sequences training objectives under limited training for various model architectures. **Bold** denotes a more effective training objective for an (Architecture, Loss, Dataset) triplet. We use * to denote statistically significant differences compared to the other training objective (left vs. right), and † to denote significant differences on the change of loss function (upper vs. lower). All tests apply a paired t-test with Bonferroni multiple testing correction ($pvalue < 0.05$). Training time of all models is limited to 1 hour.

(a) Recall@10

		MovieLens-20M		Yelp		Gowalla		Booking.com	
Architecture	Loss	Cont	RSS	Cont	RSS	Cont	RSS	Cont	RSS
GRU4Rec	BCE	0.0221†	**0.0354***	0.0075†	**0.0100*†**	**0.0026***	0.0005	0.4621	**0.4962***
	λRank	0.0082	**0.1544*†**	0.0009	**0.0045***	0.0068†	**0.0119*†**	**0.4780†**	**0.5084*†**
Caser	BCE	0.1424†	**0.1866***	0.0046†	**0.0099*†**	0.0076	**0.0081**	**0.5600*†**	0.5454†
	λRank	0.0330	**0.1496*†**	0.0009	**0.0017***	0.0087†	**0.0157*†**	0.4968	**0.5273***
SASRec	BCE	0.1537†	**0.1888***	0.0146†	**0.0269*†**	0.0089	0.0089	**0.5845*†**	0.5178
	λRank	0.1050	**0.1968*†**	0.0045	**0.0052***	0.0715	**0.1020*†**	**0.5662***	0.52464†

(b) NDCG@10

		MovieLens-20M		Yelp		Gowalla		Booking.com	
Architecture	Loss	Cont	RSS	Cont	RSS	Cont	RSS	Cont	RSS
GRU4Rec	BCE	0.0115†	**0.0183***	0.0035†	**0.0049*†**	**0.0017***	0.0002	0.2829	**0.2899***
	λRank	0.0040	**0.0839*†**	0.0004	**0.0014***	0.0033†	**0.0067*†**	**0.3132*†**	**0.3093†**
Caser	BCE	0.0784†	**0.0995***	0.0021†	**0.0049*†**	0.0039	**0.0040**	**0.3665*†**	0.3311†
	λRank	0.0177	**0.0814*†**	0.0003	**0.0007***	0.0055†	**0.0100*†**	0.3181	**0.3226***
SASRec	BCE	0.0850†	**0.1002***	0.0076†	**0.0136*†**	0.0044	**0.0044**	**0.3633*†**	0.2966
	λRank	0.0579	**0.1073*†**	0.0021	**0.0025***	0.0478†	**0.0749*†**	**0.3623***	0.3122†

Table 3. Comparing RSS-enhanced SASRec with baseline models under limited training. **Bold** denotes the best model for a dataset by the metric in the main group, underlined the second best. Symbols * and † denote a statistically significant difference compared with SASRec-RSS-BCE and SASRec-RSS-λRank respectively, according to a paired t-test with Bonferroni multiple testing correction ($pvalue < 0.05$).
[1] We do not report results for BERT4Rec models for the Gowalla dataset because due to the large number of items in this dataset, we were not able to train the model. [2] We report results for BERT4rec-16h separately due to its larger training time.

		MovieLens-20M		Yelp		Gowalla		Booking.com	
Model	Train time	Recall @10	NDCG @10	Recall @10	NDCG @10	Recall @10	NDCG @10	Recall @10	NDCG @10
Popularity	1h	0.049†*	0.025†*	0.006†	0.003†*	0.008*	0.004*	0.097†*	0.043†*
MF-BPR	1h	0.079†*	0.040†*	0.019†*	0.009†*	0.029†*	0.018†*	0.449†*	0.279†*
SASRec-vanilla	1h	0.136†*	0.067†*	0.022†*	0.011†*	0.010*	0.005†*	0.463†*	0.270†*
BERT4rec-1h	1h	0.107†*	0.053†*	0.014†*	0.007†*	N/A[1]	N/A[1]	0.479†*	0.288†*
SASRec-RSS-BCE	1h	0.189*	0.100*	**0.027***	**0.014***	0.009*	0.004*	0.518*	0.297*
SASRec-RSS-λRank	1h	**0.197†**	**0.107†**	0.005†	0.003†	**0.102†**	**0.075†**	**0.525†**	**0.312†**
BERT4Rec-16h[2]	16h	0.173†*	0.092†*	0.028*	0.014*	N/A[1]	N/A[1]	0.565†*	0.354†*

Hence, we do not report BERT4Rec results for Gowalla and leave scaling BERT4Rec to datasets with a large number of items for future research.

5.4 Data Splitting and Evaluation Measures

Following many existing publications [20, 50, 51] we evaluate our method using a Leave-One-Out strategy. Specifically, for each user from we hold out the final interaction as the test set, which we use to report metrics. We also construct a validation set using the same Leave-One-Out strategy, using the second last interaction for a group of 1024 users as validation. We set the number of training epochs to maximize NDCG@10 on the validation sets. For training, we use all interactions except those included in the test and validation sets.

We report two ranking evaluation measures: Recall[4] and Normalized Discounted Cumulative Gain (NDCG). For both metrics, we apply a rank cutoff of 10. To measure the significance of performance differences, we apply the paired t-test, and apply Bonferroni multiple testing correction, following recommended practices in IR [12]. Following the recent guidance for the evaluation of recommender systems [14], our evaluation unit is a *user* (i.e. we measure statistical significance with respect to per-user results), and we use a significance level (or *pvalue*) of 0.05.

Until recently, for efficiency reasons, most of the sequential recommendations papers reported *sampled* metrics - i.e. they sampled a small proportion of negative items and used only these items and the positive item when calculating evaluation measures. However, recent work by Krichene & Rendle [24] as well as Cañamares & Castells [5] both showed that using sampled metrics frequently leads to incorrect performance estimates and the relative order of evaluated models can change. Hence in our experiments, we use full *unsampled* metrics: we rank all possible items in the catalog and calculate metrics on this full ranking[5].

6 RESULTS

We now analyse our experimental results for each of the four research questions stated in Section 5.1.

6.1 RQ1. The benefit of Recency Sampling

To address our first research question, we compare our experimental architectures (GRU4Rec, Caser, SASRec) trained with either sequence continuation or RSS objectives. Table 2 reports the effectiveness results, in terms of Recall@10 and NDCG@10, of the three architectures, trained with both sequence continuation (denoted Cont) or RSS, and applying two different loss functions (Binary Cross-Entropy – BCE – and λRank) on four datasets (MovieLens-20M, Yelp, Gowalla, Booking.com). Statistically significant differences – according to a paired t-test with Bonferroni multiple testing correction ($pvalue < 0.05$) – among the training objectives for a given architecture, model, and loss function are shown. On first inspection of Table 2, we note that the general magnitudes of the reported effectiveness results are smaller than those reported in [50] - indeed, as stated in Section 5.4, in contrast to [50], we follow recent advice [5, 24] to avoid sampled metrics, instead preferring the more accurate unsampled metrics. The magnitudes of effectiveness reported for MovieLens-20M are in line with those reported by [9] (e.g. a Recall@10 of 0.137 for SASRec-vanilla is reported in [9] when also using a Leave-One-Out evaluation scheme and unsampled metrics).

We now turn to the comparison of training objectives. In particular, we note from the table that, on the MovieLens-20M, Yelp, and Gowalla datasets, RSS results in improved NDCG@10 in 17 out of 18 cases – 15 of which are by a statistically significant margin – and also improved Recall@10 in 16 out of 18 cases (15 statistically significant). For instance, on MovieLens-20M, SASRec is the strongest performing architecture (in line with previous findings [20, 50]); however, applying RSS significantly improves its Recall@10, both when using BCE (0.153→0.188) and when using

[4] In the context of sequential recommender systems, Recall corresponds to chances of correctly retrieving a single relevant item, and therefore many publications [20, 50, 59], call it Hit Ratio (HR). We prefer the more conventional Recall name for this metric. [5] Indeed, we also found that conclusions could change using sampled metrics.

λRank (0.105→0.196). Similarly and interestingly, SASRec with the RSS objective and λRank loss outperformed other models by a very large margin on the Gowalla dataset (e.g. Recall@10 0.102 vs. 0.071 when using sequence continuation). We postulate that the large number of items in the dataset makes the training task very hard, and only the combination of RSS with λRank allows training the model with reasonable quality in the given time limit. The only instance when RSS is worse than Continuation on these three datasets is for the GRU4Rec architecture on Gowalla with BCE loss. This is also likely to be explained by the hardness of the task on this dataset due to a large number of items. This is also reinforced by the hardness of training GRU4Rec architecture in general (see also Section 6.3 and Figure 8).

We also note the three datasets where RSS performs well (MovieLens-20M, Yelp, and Gowalla) are very different in terms of sequence length (see Table 1), varying from median length eight on Gowalla to median length 68 on MovieLens-20M. This means that RSS can be effective with both short and long sequences. On the other hand, for the Booking.com dataset, we observe that in 3 out of 6 cases, RSS is less effective. This is not an unexpected result: as we argued in Section 5.2, this dataset violates the underlying assumption encoded in Principle P1. Indeed, due to the geographical distance between items in this multi-city trip dataset, items cannot be considered out-of-order, and hence RSS does not improve the stronger models on this dataset.

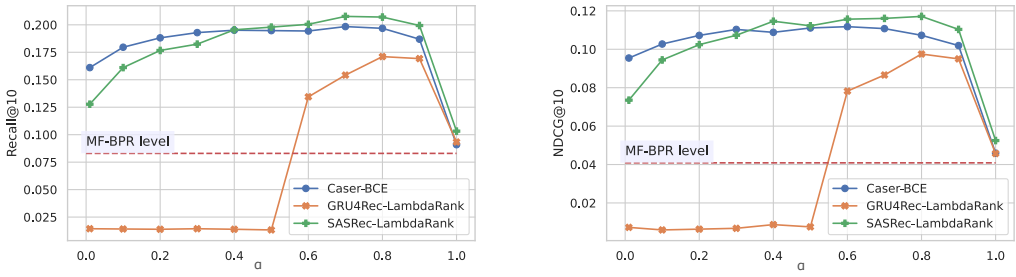
Overall, in response to RQ1, we conclude that Recency-based Sampling of Sequences improves the training of models if the items earlier in the user sequence can be treated as positives (properties exhibited by the MovieLens-20M and Gowalla datasets).

6.2 RQ2. Comparison of Different Loss Functions

Next, we address the choice of the loss function, as per RQ1. We again turn to Table 2, but make comparisons of the upper vs. lower performances in each group. For instance, for RSS, we observe that applying the listwise λRank loss function on the GRU4Rec architecture on MovieLens-20M dataset results in a significant increase (0.035→0.154), as denoted by the † symbol. Indeed, across all of Table 2, we observe that when used with RSS training task, λRank improves NDCG@10 in 8 cases out of 12 (all 8 significantly) as well as Recall@10 in 8 cases out of 12 (8 significantly). In contrast, λRank only improves over BCE in 7 out of 24 cases for the sequence continuation training objective (all by a significant margin). Overall, and in answer to RQ2, we find that λRank usually improves (except Yelp) the effectiveness of our proposed RSS training objective, while it does not offer the same level of improvement for sequence continuation. We explain this finding as follows: in sequence continuation, there is only one relevant item per sequence, and hence the benefit of a listwise loss function is limited. In contrast, RSS selects multiple relevant items for each sequence, and in this case, a listwise loss function can benefit in training the model to rank these items nearer the top of the ranking. However, as λRank did not improve RSS results on Yelp, we can not say that the improvements are consistent, and the question of the loss function selection requires further research.

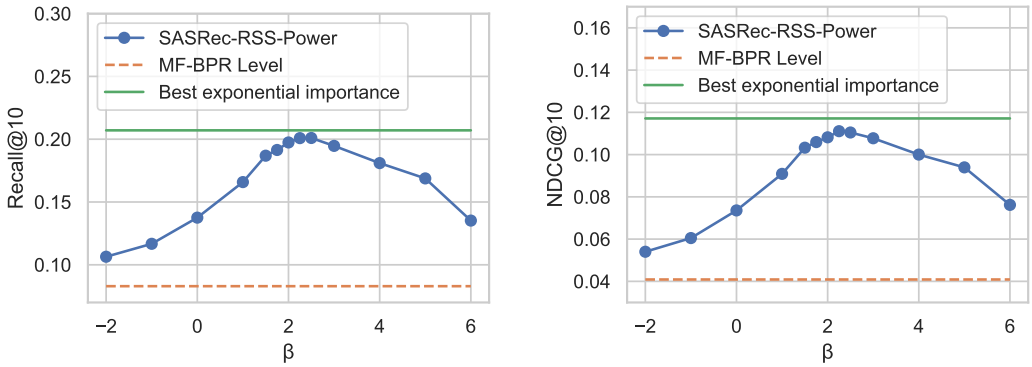
6.3 RQ3. Impact of Position Recency Importance in the Exponential Importance Function

This research question is concerned with the importance of sampling recent items in training sequences. To address this question, we train every experimental architecture with the best-performing loss from Table 2 on the MovieLens-20M dataset. We vary the recency importance parameter α in the exponential recency importance function (Equation (8)), to investigate its effect on effectiveness. In particular, as $\alpha \to 0$, the training task turns to sequence continuation, while with a large α the training task loses its sequential nature and becomes similar to matrix factorization.



(a) Effect on Recall (b) Effect on NDCG

Fig. 8. SASRec, GRU4rec and Caser performance on the MovieLens-20M dataset, when trained with Recency-based Sampling of Sequences with the exponential importance function $f_{exp}(k) = \alpha^{(n-k)}$, where n is the sequence length. Position recency importance parameter α is plotted on the x-axis. When $\alpha = 0$, the training objective turns into sequence continuation, and when $\alpha = 1$, the task becomes similar to item masking or matrix reconstruction. The training time of all models is fixed at 1 hour.



(a) Effect on Recall (b) Effect on NDCG

Fig. 9. SASRec-RSS performance when trained on MovieLens-20M dataset with power recency importance function: $f_{pow}(k) = \left(\frac{k+1}{n+1}\right)^{e^{\beta}}$ and variable parameter β.

Figure 8 summarizes the impact of α on the model effectiveness. We also present the performance of the MF-BPR [48] baseline. From the figures, we observe that when we set α close to zero, the results match those we report in Table 2 for the sequence continuation task, illustrating that under small α, RSS only samples the last element in each sequence. Similarly, for $\alpha = 1$ we observe that the effectiveness of all models drops almost to that of the matrix factorization baseline, as target items are simply sampled from sequences without any ordering preference. Note, that in this case, we sample target items uniformly, which is similar to BERT4Rec's item masking. However, BERT4Rec also has access to the positions of masked items (through the position embeddings), whereas in the case of $\alpha = 1$ the positional information is completely lost and the model can not learn to predict the *next* item and predicts *some* item instead. Overall, the general trends visible in Figure 8 suggest that RSS allows to train effective models across a wide range of the α parameter values:

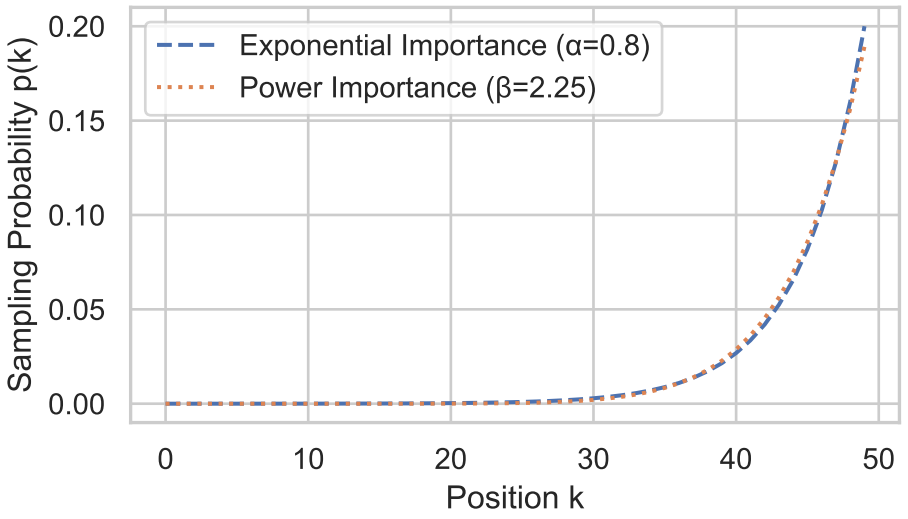


Fig. 10. Optimal sampling probability distributions for SASRec-RSS model generated by exponential ($f_{exp}(k) = 0.8^{n-k}$) and power ($f_{pow}(k) = \left(\frac{k+1}{n+1}\right)^{e^{2.25}}$) recency importance functions. The plotted curves are mostly superimposed.

for Caser and SASRec, large improvements over sequence continuation training are achieved for $0.2 \leq \alpha \leq 0.9$; for GRU4Rec, strong performance is obtained $0.6 \leq \alpha \leq 0.9$. Indeed, for small α, the number of positive items is limited, and hence the lambda gradients in λRank are also small. This provides little evidence to the GRUs in GRU4Rec, which therefore struggles with the vanishing gradient problem (a problem faced by many such recurrent architectures).

Overall, in response to RQ3, we find that the higher values of the recency importance parameter $\alpha \leq 0.9$ results in effective performance for all three model architectures.

6.4 RQ4. Recency Importance Function Shape

We now analyse the effects of replacing the exponential importance function (defined in Equation (8)) with the power importance (defined in Equation (10)).

To understand whether or not this different family of distributions may improve overall recommendations quality compared to exponential recency importance, we train a SASRec-RSS model on MovieLens-20M dataset with Binary Cross-Entropy loss and power importance while varying the power hyperparameter β in the range from -2 to 6. Figure 9 shows the effect of this variation on the model performance in terms of NDCG and Recall metrics.

From the figure we note that the model achieves the best performance when $\beta = 2.25$. Interestingly, the best achieved Recall@1 of 0.2008 and NDCG@10 of 0.1110 are similar to the best results achieved by the model with exponential importance: it obtains Recall@10 of 0.2070 and NDCG@10 of 0.1171.

These similarities are not surprising when we look to Figure 10, which plots the sampling probability distributions corresponding to the optimal power recency function (with $\beta = 2.25$) and optimal exponential recency function (with $\alpha = 0.8$). As we can see from the figure, the shapes of the distributions are almost identical, which leads to similar model performance.

Overall, this analysis suggests that both power and exponential functions are viable alternatives, and if they are tuned properly they are likely to produce similar sampling probability distributions. However, we find that the exponential importance function is easier to tune, as it exhibits high performance in a wide range of hyperparameter values. Overall, we recommend using the exponential function as a default importance function in RSS.

6.5 RQ5. Comparison with Baselines

To address our fifth research question concerning the comparison with baseline models, we compare the best-performing RSS-enhanced model, SASRec-RSS, using both λRank and Binary Cross-Entropy losses, with the 5 baseline models described in Section 5.3. Table 3 summarizes the results of this comparison, reporting effectiveness metrics as well as training time duration. In particular, recall that all models are trained for less than 1 hour, except for BERT4Rec-16h (a full training of BERT4Rec). Moreover, we did not train BERT4Rec on the Gowalla dataset, because the preprocessing code for BERT4Rec does not scale to its large number of items (indeed, Gowalla has more items than users, see Table 1). Indeed, the preprocessing code to generate masked training sequences requires 14GB of storage for MovieLens-20M, but 548GB for Gowalla.

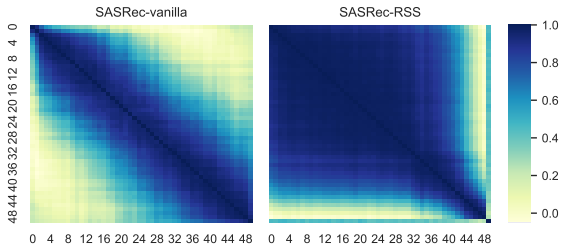
On analyzing Table 3, we observe that SASRec-RSS (λRank or BCE) achieves the most effective performance on all four datasets among the time-limited recommendation models. For instance, on the MovieLens-20M dataset, compared to the original formulation of SASRec (denoted SASRec-vanilla), the RSS adaptation significantly improves NDCG@10 (by the margin of 60%) for the same training duration. Moreover, compared to the 16 hour training of BERT4Rec, SASRec-RSS exhibits 16% higher NDCG@10 (a significant improvement), despite needing only 6% of the training time (16h$\rightarrow$ 1h). For Booking.com, where RSS was less effective, SASRec-RSS with λRank objective obtains the NDCG@10 12% less than that obtained by the expensive BERT4Rec-16h model, and the Recall that is 7% less. Interestingly, on the Yelp dataset, the λRank version of SASRec is not effective (same performance as popularity baseline), but the BCE version of the model significantly outperforms all other models in the main group and achieves performance on par with BERT4Rec-16h. Furthermore, we see that in all cases where we able to train BERT4Rec, under limited training time, BERT4Rec underperforms compared to the SASRec-RSS (λRank or BCE version).

Overall, in answer to RQ5, we find that SASRec-RSS can achieve significantly higher effectiveness than the state-of-the-art SASRec and BERT4Rec approaches when trained for a comparable time. Furthermore, we can achieve performances exceeding or very close to a fully-trained BERT4Rec, but with much less training time. This highlights the importance of an appropriate training objective in general and the benefits of our proposed RSS training objective in particular.

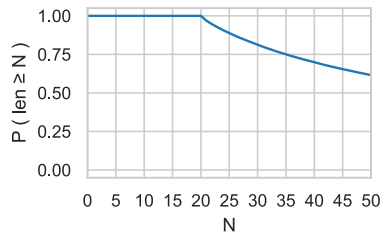
6.6 RQ6. Effect on Positional Embeddings

To better understand the impact of RSS on the resulting learned models, we analyse the similarities between the positional embeddings learned by SASRec when trained with both the shifted sequence training objective (SASRec-vanilla) and RSS. We compute the cosine similarity matrix S as defined by Equation (16) between the embeddings of each pair of positions and then use these matrices to validate desirable properties of positional embeddings, as described in Section 4.4.

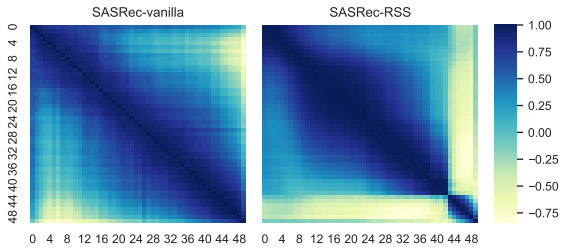
The heatmaps shown in Figure 11 (left) graphically visualize these similarity matrices for all four experimental datasets. Darker color on the figure corresponds to the higher similarity between positions, whereas lighter color corresponds to lower similarity. Figure 11 (right) also shows the distributions of sequence lengths for these datasets, as these distributions help to explain some of the properties of the matrices: For example, in Figure 11e we see that for the Gowalla dataset, there are artifact lines appearing at position 26. As we can see from the corresponding sequence length distribution (Figure 11f), this corresponds to a sudden drop in the sequence length distribution in



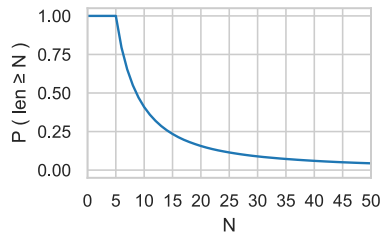
(a) MovieLens-20M position similarities



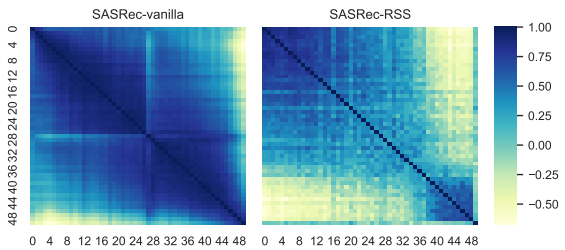
(b) MovieLens-20M sequence lengths



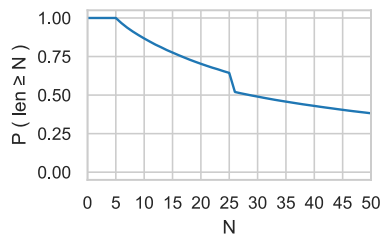
(c) Yelp



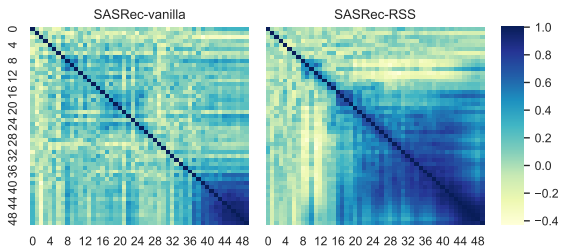
(d) Yelp sequence lengths



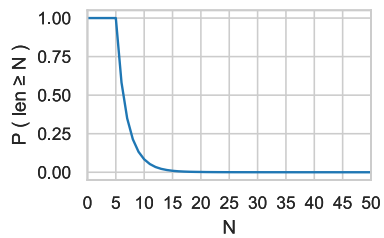
(e) Gowalla



(f) Gowalla sequence lengths



(g) Booking.com



(h) Booking.com sequence lengths

Fig. 11. Similarity matrices of positional embeddings

the dataset. Indeed, for the Gowalla dataset, only 52% of sequences are 26 items or longer, a drop from 64% for the sequences of length ≥ 25 - this important change in distribution is reflected in the learned embeddings. Similarly, for the Booking.com dataset, both SASRec-vanilla and SASRec-RSS

mostly contain noise for early positions, with little correspondence between nearby positions. This can be explained in that the Booking.com dataset has only a very small number of sequences longer than 15 items, as can be seen from the corresponding sequence distribution plot.

The overall general trends that can be observed in Figure 11 are as expected, in that all matrices are symmetric with respect to the main diagonal (because of the symmetry of cosine similarity). We also can see from the figures that, in all cases, the similarity is high close to the main diagonal and lower further away from the diagonal. This means, that both SASRec-Vanilla and SASRec-RSS successfully learned our desirable Properties Pr.1 and Pr.2 (close positions have similar positional embeddings; embeddings of distant positions are less similar) – as defined in Section 4.4.

We now look to the symmetry with respect to the secondary diagonal, which as we argue in Section 4.4 is an undesirable property, as it violates Property Pr.3 (earlier positions are more similar to each other than the recent ones). As we expect, for each dataset except Booking.com, in SASRec-vanilla the similarity between a pair of positional embeddings is mostly defined by the distance between their respective positions (see also Equation (23)). As we showed in Section 4.4, this makes the matrix symmetric with respect to the secondary diagonal; therefore, SASRec-vanilla violates Property Pr.3.

In contrast, the similarity matrices for SASRec trained with the RSS training objective are not symmetric with respect to the secondary diagonal. Instead, for each dataset except Booking.com, we observe two groups of positions: a large group of earlier positions and a small group of recent positions. For example, on the MovieLens-20M dataset, we can say that positions 0..40 comprise the earlier group whereas positions 41..49 comprise the recent group. Positions within each group tend to be similar to each other and different compared to the positions from the other group. In practice this means that in contrast to SASRec-vanilla, the similarity matrices of SASRec-RSS do not violate Property Pr.3. This confirms that RSS helps models to make better distinction between recent and earlier positions, as we hypothesized in Section 4.4.

Furthermore, from Figure 11g we can also see that position similarity matrices for the Booking.com dataset do not look like other similarity matrices. For example, similarity matrices for SASRec-vanilla trained on this dataset are not symmetric with respect to secondary diagonal. This can be explained by the shorter sequence lengths in the Booking.com dataset. As we can see from the figure, there are virtually no sequences with more than 20 interactions in this dataset, which differs from the other datasets in our experiments (even for the Yelp dataset, where the average sequence length is relatively short, approximately 4% of all sequences have at least 50 interactions). According to our applied padding scheme (see Section 5.3.1 and Figure 7), this means that positions on the left side of the sequence will be padded and ignored by the model. As a result, the positional embeddings for earlier positions remain mostly unchanged after random initialization in the model, which explains the differences in similarity matrices with the other datasets.

In summary, we conclude that while both SASRec-vanilla and SASRec-RSS models successfully learn Properties Pr.1 and Pr.2, however, only an RSS-enhanced model successfully learned Property Pr.3 on three out of four datasets.

7 CONCLUSIONS

In this work, we identified two limitations in existing training objectives for sequential recommender models. To address these two limitations, we proposed a refined training objective, called Recency-based Sampling of Sequences (RSS). Through experimentation on four datasets, we found that this relatively simple change in the training objective can bring significant improvements in the overall effectiveness of state-of-the-art sequential recommendation models, such as SASRec and Caser. Furthermore, we showed that the λRank loss function brought further effectiveness benefits to training under RSS not otherwise observed under a more traditional sequence continuation

task. Indeed, on the large MovieLens-20M dataset, we observed that RSS applied to the SASRec model can result in an 60% improvement in NDCG over the vanilla SASRec model (Table 3), and a 16% improvement over a fully-trained BERT4Rec model, despite taking 93% less training time than BERT4Rec (see also Figure 1). Moreover, on the Yelp and Gowalla datasets, which both have geographic and strong sequential characteristics, RSS applied to SASRec brought significant benefits in both NDCG and Recall metrics (Table 3). We further experimented with two families of recency importance functions (power importance and exponential importance) and found that when their parameters are tuned properly, these functions are likely to produce similar sampling probability distributions and consequently achieve similar performance (this is shown in Figures 9 and 10). We also analyzed the effect of RSS on the learned positional embeddings of the SASRec model and have shown that in contrast to the original version of SASRec, the RSS-enhanced version successfully learns to distinguish recent and earlier positions (see also Figure 11). Finally, while we did not apply RSS to BERT4Rec due to its usage of the item masking training objective, which is harder to adapt to RSS however, we believe that BERT4Rec could be adapted in future work to benefit from RSS.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A system for large-scale machine learning. In *Proc. USENIX*. 265–283.

[2] Mehrnaz Amjadi, Seyed Danial Mohseni Taheri, and Theja Tulabandhula. 2021. KATRec: Knowledge aware attentive sequential recommendations. In *Proc. ICDS*. 305–320.

[3] Shuqing Bian, Wayne Xin Zhao, Kun Zhou, Jing Cai, Yancheng He, Cunxiang Yin, and Ji-Rong Wen. 2021. Contrastive Curriculum Learning for Sequential User Behavior Modeling via Data Augmentation. In *Proc. CIKM*. 3737–3746.

[4] Christopher JC Burges. 2010. From RankNet to LambdaRank to LambdaMART: An overview. *Learning* 11, 23-581 (2010), 81.

[5] Rocío Cañamares and Pablo Castells. 2020. On target item sampling in offline recommender system evaluation. In *Proc. RecSys*. 259–268.

[6] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. *Proceedings of Machine Learning Research* (2011), 1–24.

[7] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *Proc. KDD*. 1082–1090.

[8] Sung Min Cho, Eunhyeok Park, and Sungjoo Yoo. 2020. MEANTIME: Mixture of attention mechanisms with multi-temporal embeddings for sequential recommendation. In *Proc. RecSys*. 515–520.

[9] Alexander Dallmann, Daniel Zoller, and Andreas Hotho. 2021. A Case Study on Sampling Strategies for Evaluating Neural Sequential Item Recommendation Models. In *Proc. RecSys*. 505–514.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL-HLT*. 4171–4186.

[11] Elisabeth Fischer, Daniel Zoller, Alexander Dallmann, and Andreas Hotho. 2020. Integrating keywords into BERT4Rec for sequential recommendation. In *German Conference on Artificial Intelligence (Künstliche Intelligenz)*. 275–282.

[12] Norbert Fuhr. 2021. Proof by experimentation? Towards better IR research. In *ACM SIGIR Forum*, Vol. 54. 1–4.

[13] Dmitri Goldenberg and Pavel Levin. 2021. Booking.com Multi-Destination Trips Dataset. In *Proc. SIGIR*. 2457–2462.

[14] Asela Gunawardana, Guy Shani, and Sivan Yogev. 2022. Evaluating Recommender Systems. In *Recommender Systems Handbook*, Francesco Ricci, Lior Rokach, and Bracha Shapira (Eds.). Springer US, New York, NY, 547–601.

[15] F Maxwell Harper and Joseph A Konstan. 2015. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)* 5, 4 (2015), 1–19.

[16] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proc. CIKM*. 843–852.

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

[18] Ziniu Hu, Yang Wang, Qu Peng, and Hang Li. 2019. Unbiased LambdaMART: An Unbiased Pairwise Learning-to-Rank Algorithm. In *Proc. WWW*. 2830–2836.

[19] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y Chang. 2018. Improving sequential recommendation with knowledge-enhanced memory networks. In *Proc. SIGIR*. 505–514.

[20] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *Proc. ICDM*. 197–206.

[21] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Proc. NeurIPS*. 3146–3154.

[22] Tobias Koopmann, Konstantin Kobs, Konstantin Herud, and Andreas Hotho. 2021. CoBERT: Scientific Collaboration Prediction via Sequential Recommendation. In *Proc. ICDMW*. 45–54.

[23] Yehuda Koren, Steffen Rendle, and Robert Bell. 2022. Advances in Collaborative Filtering. In *Recommender Systems Handbook*, Francesco Ricci, Lior Rokach, and Bracha Shapira (Eds.). New York, NY, 91–142.

[24] Walid Krichene and Steffen Rendle. 2020. On sampled metrics for item recommendation. In *Proc. KDD*. 1748–1757.

[25] Maciej Kula. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. In *Proc. Workshop on New Trends on Content-Based Recommender @ RecSys (CEUR Workshop Proc., Vol. 1448)*. 14–21.

[26] Hojoon Lee, Dongyoon Hwang, Sunghwan Hong, Changyeon Kim, Seungryong Kim, and Jaegul Choo. 2021. MOI-Mixer: Improving MLP-Mixer with Multi Order Interactions in Sequential Recommendation. *arXiv preprint arXiv:2108.07505* (2021).

[27] Haoyang Li, Xin Wang, Ziwei Zhang, Jianxin Ma, Peng Cui, and Wenwu Zhu. 2021. Intention-aware sequential recommendation with structured intent transition. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2021).

[28] Roger Zhe Li, Julián Urbano, and Alan Hanjalic. 2021. New Insights into Metric Optimization for Ranking-based Recommendation. In *Proc. SIGIR*. 932–941.

[29] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (2009), 225–331.

[30] Zhiwei Liu, Ziwei Fan, Yu Wang, and Philip S. Yu. 2021. Augmenting Sequential Recommendation with Pseudo-Prior Items via Reversely Pre-training Transformer. In *Proc. SIGIR*. 1608–1612.

[31] Malte Ludewig and Dietmar Jannach. 2018. Evaluation of Session-Based Recommendation Algorithms. *User Modeling and User-Adapted Interaction* 28, 4-5 (Dec. 2018), 331–390.

[32] Chen Ma, Peng Kang, and Xue Liu. 2019. Hierarchical gating networks for sequential recommendation. In *Proc. KDD*. 825–833.

[33] Jianxin Ma, Chang Zhou, Hongxia Yang, Peng Cui, Xin Wang, and Wenwu Zhu. 2020. Disentangled self-supervision in sequential recommenders. In *Proc. KDD*. 483–491.

[34] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. 2021. Variational Bayesian representation learning for grocery recommendation. *Information Retrieval Journal* 24 (10 2021), 1–23.

[35] Umaporn Padungkiatwattana, Thitiya Sae-Diae, Saranya Maneeroj, and Atsuhiro Takasu. 2022. ARERec: Attentive Local Interaction Model for Sequential Recommendation. *IEEE Access*.

[36] Aleksandr Petrov and Craig Macdonald. 2022. Effective and Efficient Training for Sequential Recommendation Using Recency Sampling. In *16th ACM Conference on Recommender Systems (RecSys 2022)*. arXiv:2207.02643 [cs]

[37] Aleksandr Petrov and Craig Macdonald. 2022. A Systematic Review and Replicability Study of BERT4Rec for Sequential Recommendation. In *Proc. RecSys*.

[38] Aleksandr Petrov and Yuriy Makarov. 2021. Attention-based neural re-ranking approach for next city in trip recommendations. In *Proc. WSDM WebTour*. 41–45.

[39] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2021. Are Neural Rankers still Outperformed by Gradient Boosted Decision Trees?. In *Proc. ICLR*.

[40] Ruihong Qiu, Zi Huang, Tong Chen, and Hongzhi Yin. 2021. Exploiting Positional Information for Session-based Recommendation. *ACM Transactions on Information Systems (TOIS)* 40, 2 (2021), 1–24.

[41] Ruihong Qiu, Zi Huang, Jingjing Li, and Hongzhi Yin. 2020. Exploiting cross-session information for session-based recommendation with graph neural networks. *ACM Transactions on Information Systems (TOIS)* 38, 3 (2020), 1–23.

[42] Ruihong Qiu, Zi Huang, and Hongzhi Yin. 2021. Memory Augmented Multi-Instance Contrastive Predictive Coding for Sequential Recommendation. *CoRR* abs/2109.00368 (2021).

[43] Ruihong Qiu, Zi Huang, Hongzhi Yin, and Zijian Wang. 2022. Contrastive learning for representation degeneration problem in sequential recommendation. In *Proc. WSDM*. 813–823.

[44] Ruihong Qiu, Hongzhi Yin, Zi Huang, and Tong Chen. 2020. GAG: Global attributed graph neural network for streaming session-based recommendation. In *Proc. SIGIR*. 669–678.

[45] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–36.

[46] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models Are Unsupervised Multitask Learners.

[47] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67.

[48] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proc. CUAI*. 452–461.
[49] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proc. WWW*. 811–820.
[50] 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 *Proc. CIKM*. 1441–1450.
[51] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proc. WSDM*. 565–573.
[52] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. 2022. *Natural Language Processing with Transformers, Revised Edition*. "O'Reilly Media, Inc.".
[53] 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 *Proc. NeurIPS*. 5998–6008.
[54] Mengting Wan, Di Wang, Jie Liu, Paul Bennett, and Julian McAuley. 2018. Representing and recommending shopping baskets with complementarity, compatibility and loyalty. In *Proc. CIKM*. 1133–1142.
[55] Chenyang Wang, Weizhi Ma, and Chong Chen. 2022. Sequential Recommendation with Multiple Contrast Signals. *ACM Transactions on Information Systems (TOIS)* (2022).
[56] Qitian Wu, Chenxiao Yang, Shuodian Yu, Xiaofeng Gao, and Guihai Chen. 2021. Seq2Bubbles: Region-Based Embedding Learning for User Behaviors in Sequential Recommenders. In *Proc. CIKM*. 2160–2169.
[57] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Bolin Ding, and Bin Cui. 2020. Contrastive Learning for Sequential Recommendation. *arXiv preprint arXiv:2010.14395* (2020).
[58] Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. 2016. LambdaFM: learning optimal ranking with factorization machines using lambda surrogates. In *Proc. CIKM*. 227–236.
[59] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A simple convolutional generative network for next item recommendation. In *Proc. WSDM*. 582–590.
[60] Zhuo-Xin Zhan, Ming-Kai He, Wei-Ke Pan, and Zhong Ming. 2022. Transrec++: Translation-based sequential recommendation with heterogeneous feedback. *Frontiers of Computer Science* 16, 2 (2022), 1–3.
[61] Mengqi Zhang, Shu Wu, Xueli Yu, Qiang Liu, and Liang Wang. 2022. Dynamic graph neural networks for sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2022).
[62] Pengyu Zhao, Tianxiao Shui, Yuanxing Zhang, Kecheng Xiao, and Kaigui Bian. 2021. Adversarial oracular seq2seq learning for sequential recommendation. In *Proc. ICJAI*.
[63] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. 2020. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *Proc. CIKM*. 1893–1902.
[64] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. 2001. Using temporal data for making recommendations. In *Proc. UAI*. 580–588.

A DETAILS OF λRANK IMPLEMENTATION

There are several available implementations of λRank. Qin et al. [39] showed that the λRank version from the LightGBM [21] library significantly outperforms other available implementations in a learning to rank task. We could not use LightGBM directly, as it implements a CPU version of the gradient boosting trees method, and therefore is not compatible with neural tasks that are typically trained on GPUs. Therefore, in our work, we use LightGBM as the reference implementation of λRank, and match its implementation, including modifications, not described in the original λRank paper [4].

On analysis of the LightGBM [21] source code, we find that compared to the original definition of the loss function (Equation (14)), it includes two additional transformations that are not described in the original λRank paper [4]. These modifications are shown in Equations (24) and (25), below:

(1) LightGBM uses normalized $\Delta NDCG$ based on the difference on item scores, as follows:

$$\widehat{\lambda_i} = \sum_{j \in I} \left| \frac{\Delta NDCG_{ij}}{s_i - s_j} \right| \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} \tag{24}$$

(2) LightGBM normalizes the λ-gradients of the predicted item scores across all items I:

$$\overline{\lambda_i} = \widehat{\lambda_i} * \frac{\log_2(1 + \sum_{i \in I} \widehat{\lambda_i})}{\sum_{i \in I} \widehat{\lambda_i}} \tag{25}$$

The justifications[6] for these changes provided by the LightGBM contributors describes them as specific to tree-based models. The idea is that tree-based models are sensitive to the ranges of labels, as they average values in the leaves, so that in skewed distributions, the output of the tree-based models drifts towards larger values. These normalizations help to solve the skewed distribution problem. We keep these normalizations in our implementation of λRank, because we want to match the best available reference implementation. We leave the question of whether or not these normalizations are helpful for optimization of deep neural networks for future research, however our initial experiments suggest that it may improve neural networks training as well.

As λRank relies on λ-gradients, instead of usual loss function gradients, we are not able to rely on automatic gradients computation provided by TensorFlow. Instead, we implement it as a function that returns a dummy value with custom gradients. We test that our implementation of λRank returns exactly same λ-gradients as the C++ implementation from LightGBM[7]. However, compared to the LightGBM version, our version uses vectorized computations, and therefore can effectively make use of GPUs. Our implementation is available in our repository.[8] This implementation is not specific to recommender systems and can be used with other applications, which require ranking metrics optimization (e.g. for search results ranking.).

As can be seen from Equation (25), to compute each $\overline{\lambda_i}$, λRank requires access to the scores of all items in the catalog. Indeed, calculating $\overline{\lambda_i}$ for a catalog of m items, where each application of Equation (25) requires m operations, giving an overall time complexity of λRank is, at least, $O(m^2)$. Moreover, it also requires access to pre-computed NDCG scores (in order to compute $\Delta NDCG$). Calculating NDCG scores requires sorting the items, a procedure which costs $m \log(m)$ operations. Hence, the overall time complexity of λRank for a single training sample is $O(m^2 + m \log(m))$.

However, after a few training iterations, it is reasonable to expect that the model will be able to rank all relevant items close to the top of the ranking. Assuming that k is the rank of the lowest-scored positive element, the contribution of the items ranked below k into the λ-gradients is 0. Hence, following the LightGBM implementation, we use a truncated version of λRank, where we only use k highest-scored items for computing λ-gradients. This is an equivalent of applying ranking cutoff at position k when computing NDCG. In doing so, the computational complexity of λRank for a single training sample becomes $O(k^2 + m \log(k))$.

Our initial experiments showed that setting truncation level k at 4000 allowed us to train models with λRank in a reasonable time without effectiveness degradation for the datasets with a catalog size of fewer than 2,000,000 items. Therefore, we use truncation level 4000 in all our experiments. We leave more detailed research on the effect of the truncation level on model effectiveness/efficiency trade-off to future research.

[6] https://github.com/microsoft/LightGBM/pull/2331#issuecomment-523259298

[7] https://github.com/microsoft/LightGBM/blob/master/src/objective/rank_objective.hpp

[8] https://github.com/asash/bert4rec_repro/blob/main/losses/lambda_gamma_rank.py

B ADDITIONAL INFORMATION ON MODEL TRAINING

Table 4 presents the number of training sequences and the number of full epochs for which we trained each model. These numbers are defined by the 1-hour training time limit. We check the time limit after every epoch and employ an early stopping mechanism - we always save the best model according to validation data and stop training if the model does not improve for 100 epochs. As can be seen from the table, model throughput is mostly defined by the model architecture. For example, compared to SASRec, GRU4Rec has a much lower training throughput on all datasets except Gowalla. On Gowalla, all models have relatively low training throughput, which is explained by a large number of items in this dataset. RSS usually has negative (18/24 cases) or no effect (6/24 cases) on the total number of trained epochs. The negative effect is explained by a more complex training sample construction procedure. λRank has no effect in 10/24 cases, negative effect in 10/24 cases and small positive effect in 4/10 cases.

Table 4. The number of training sequences and full data passes (epochs) used for training. Training time is limited by 1 hour. We check the time limit condition after a full epoch (this explains repeating numbers in Table 4a)

(a) Number of training sequences

		MovieLens-20M		Yelp		Gowalla		Booking.com	
Architecture	Loss	Cont	RSS	Cont	RSS	Cont	RSS	Cont	RSS
GRU4Rec	BCE	1,149,992	1,043,016	6,603,668	6,029,436	1,292,520	1,464,856	1,125,968	1,125,968
	λRank	1,096,504	1,043,016	6,029,436	6,029,436	1,464,856	1,464,856	1,125,968	1,125,968
Caser	BCE	6,070,888	3,984,856	12,345,988	11,197,524	1,292,520	1,464,856	6,755,808	5,911,332
	λRank	4,947,640	3,449,976	12,345,988	11,197,524	1,464,856	1,464,856	5,911,332	5,348,348
SASRec	BCE	6,873,208	5,161,592	12,345,988	11,197,524	1,292,520	1,464,856	14,356,092	11,118,934
	λRank	6,926,696	4,305,784	12,345,988	11,197,524	1,464,856	1,464,856	10,978,188	9,007,744

(b) Number of epochs

		MovieLens-20M		Yelp		Gowalla		Booking.com	
Architecture	Loss	Cont	RSS	Cont	RSS	Cont	RSS	Cont	RSS
GRU4Rec	BCE	43	39	23	21	15	17	8	8
	λRank	41	39	21	21	17	17	8	8
Caser	BCE	227	149	43	39	15	17	48	42
	λRank	185	129	43	39	17	17	42	38
SASRec	BCE	257	193	43	39	15	17	102	79
	λRank	259	161	43	39	17	17	78	64