Effective and Efficient Training for Sequential Recommendation using Recency Sampling

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

ABSTRACT

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 expenses, hinders product development timescales and prevents the model from being regularly updated to adapt to changing user preferences. Training 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 BERT4Rec 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 training objective 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 state-of-the-art BERT4Rec, but with much less training time.

ACM Reference Format:
Aleksandr Petrov and Craig Macdonald. 2022. Effective and Efficient Training for Sequential Recommendation using Recency Sampling. In *Sixteenth ACM Conference on Recommender Systems (RecSys '22), September 18–23, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3523227.3546785

1 INTRODUCTION

Sequential recommender models is a class of recommender systems, which consider the order of the user-item interactions, are increasingly popular [39]. Early sequential recommender systems used Markov Chains [41, 55], however most modern ones use deep neural networks, and have adapted ideas from other domains such as language modelling [15, 16, 18, 42] or image processing [43]. These deep neural models have been shown to outperform traditional non-neural methods by a significant margin [18, 42, 43, 50].

However, the most advanced sequential models, such as BERT4-Rec, suffer from a slow training problem. Indeed, our experiments show that in order to reproduce result reported in the original publication, BERT4Rec requires more than 10 hours training using modern hardware (see also replicability paper [31]). This is illustrated in Figure 1, which portrays the NDCG@10 of MF-BPR [40], SASRec [18] and BERT4Rec [42] models for different training durations on the MovieLens-20M dataset [14].

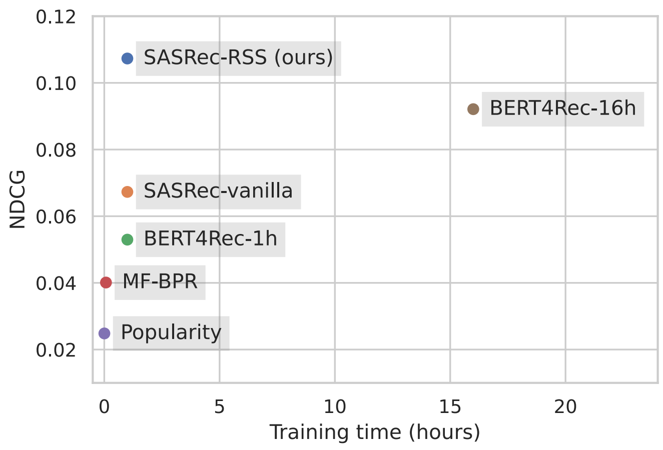


Figure 1: The SASRec [18] model trained with our proposed training method outperforms BERT4Rec on the MovieLens-20M dataset [14] and requires much less training time. SASRec-vanilla corresponds to the original version of SASRec; BERT4Rec-1h and BERT4Rec-1h are versions of BERT4Rec trained for 1 hour 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 was not retrained 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. Although all three components have a marked impact on training efficiency, 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, sequence continuation [15, 16, 43] (including its popular form, next item prediction) is probably the most intuitive and popular. 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.

Second, in the item masking approach – which is used by BERT4Rec [42] – 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 is only weakly related to the end goal of sequential recommendation. Indeed, we will show that, despite leading to better effectiveness, the 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.

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 an 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).

In short, the main contributions of this paper are as follows: (i) We identify limitations in the existing training objectives used by sequential recommendation models; (ii) We propose Recency-based Sampling of Sequences, which emphasises the importance of more recent interactions during training; (iii) We perform extensive empirical evaluations on four sequential recommendation datasets, demonstrating significant improvements over existing state-of-the-art approaches. 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.

2 BACKGROUND

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 [16] and the improved GRU4Rec$^+$ [15] (using an improved listwise loss function), which are models that use the Recurrent Neural Networks (RNN) architecture. On the other hand, Caser [43] 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 [50], 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 the Section 3.

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 [44] architecture. SASRec [18] uses transformer blocks to predict the next item in the sequence based on all previous elements. BERT4Rec [42] 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 limited training time.

Finally, recent advances have used graph neural networks (GNNs) for sequential recommendation [34–36, 38]. 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 personalised shopping basket completion (e.g. [29, 45]) are out of the scope of this work. On the other hand, CL4SRec [48] 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 use of GNNs nor data augmentation. We provide details of these training objectives in the next section.

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

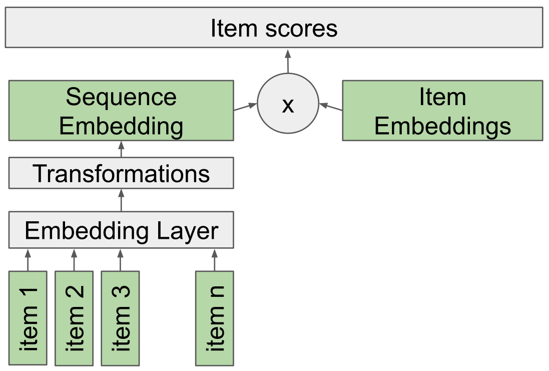


Figure 2: Principal architecture of many sequential recommenders. This applies to GRU4rec [16], GRU4rec⁺ [15], Caser [43] and, with minor modifications to SASRec [18].

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$^+$ [15]) 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 selection of an appropriate loss function. We further discuss some possible choices of the loss functions for our proposed method later in Section 4.2. In the following, we review approaches to generate training samples and identify their limitations, a summary of which we provide in the 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 [15, 16, 18, 43] 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 [18, 40], in-batch negatives [16], or the negatives with highest scores [49]. 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 training samples generation procedure - we sample (user, item) pairs as inputs and assign labels for the pairs based on interactions. A classic model that uses matrix reconstruction is Bayesian Personalized Rank (BPR) [40], 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.

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. We use s_{input} as the model input, and assign label 1 to the postive 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 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 [43]) 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 to generate up to $n - 1$ training samples out of 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 [43]. GRU4rec, GRU4rec$^+$ and Caser models use variations of the sequence continuation task for training. The training task used by SASRec and NextItNet [50] is slightly different: they work as a sequence-to-sequence models where the target sequence is shifted by one element compared to the input. 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.

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.

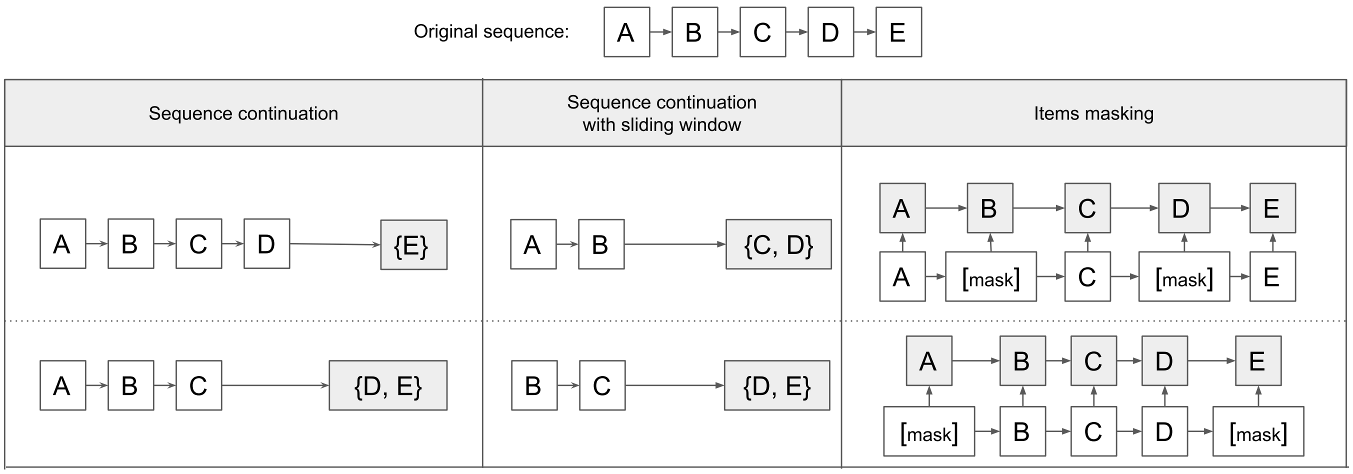


Figure 3: Training sample generation strategies used in existing models. White boxes represent model inputs and filled boxes represent model outputs. In Sequence Continuation, the sequence is split into two parts, with the aim of predicting whether or not an item belongs to the second part based on the 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.

BERT4Rec [42] 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 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. [42] 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 reviewed two main training objectives used in sequential recommendations - 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 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) and choice of loss function (Section 4.2).

Recency-based Sampling of Sequences (RSS) is a training objective that is closely related to the sequential recommendations and allows 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: 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: $f(k)$ is positive ($f(k) > 0$) and monotonically growing ($f(k) \leq f(k+1)$). 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, $f(i)$:

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

We generate the input sequence to the model by removing targets from the original sequence. The full procedure is described in Algorithm 1. One example of a recency importance function that has the required properties is the exponential function:

$$f(k) = \alpha^{n-k} \tag{2}$$

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 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 4 provides an example of the recency importances (for $\alpha = 0.8$) and the generated samples.

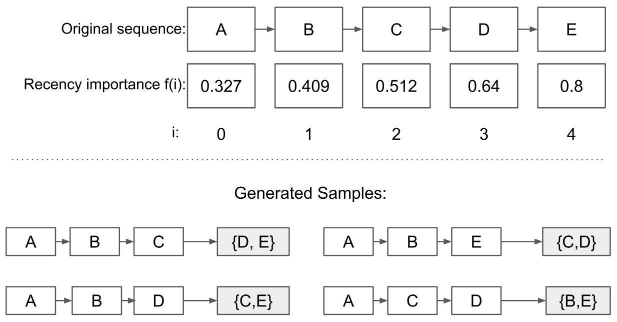


Figure 4: Recency-based Sampling of Sequences. The beginnings of the sequences remains largely unchanged, whereas elements from the end of the sequence are chosen as positive samples more frequently.

4.1 The RSS Training Objective

There are other possible position importance functions, such as linear $f(k) = k + 1$, and the best function may be a property of particular dataset. However, in this work, we only consider the exponential function, leaving other possibilities to future research.

4.2 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 [25]. RSS works with all types of loss functions that support multiple positive samples within each training sample.

GRU4rec$^+$ [15] 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] (or λRank), another listwise optimization loss function. λRank has been widely deployed in training learning-to-rank scenarios [6, 33] for web search. Similarly, λRank has been shown to be advantageous for recommender tasks [24], for example when applied to Factorisation Machines [49] or transformer-based sequential models [32]. λRank [4] uses *Lambda Gradients* instead of objective function gradients in the gradient descent method. The Lambda Gradient for an item $i \in I$ is defined as follows:

$$\lambda_i = \sum_{j \in I} |\Delta NDCG_{ij}| \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} \tag{3}$$

where s_i and s_j are predicted scores, $\Delta NDCG_{ij}$ is a change in NDCG metric in case of swapping items i & j, and σ is a hyperparameter typically set to 1.

In addition to λRank, we also experiment with the standard Binary Cross-Entropy (BCE), following [18, 43], to investigate the effect of the listwise loss and the necessity of both training objective and the loss function in our solution.

Algorithm 1 Recency-based Sampling of Sequences

```
Input: $sequence$ - a sequence of interactions; $\tau$ - 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($sequence, \tau, f$)
      $sampledIdx \leftarrow set()$
      $n \leftarrow length(sequence)$; $c \leftarrow max(1, int(n * \tau)))$
      $prob \leftarrow Array[n]$
      $prob[i] \leftarrow \frac{f(i)}{\sum_{j=0}^{n-1} f(j)}$ 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.

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).

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 (2)) of RSS?
- **RQ4** How do RSS-enhanced models compare with state-of-the-art baselines?

5.2 Datasets

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

MovieLens-20M [14] is a movie recommendation dataset, and is popular for benchmarking sequential recommenders [8, 11, 23, 27, 28, 36, 42, 47, 53]. Note that MovieLens-20M is a ratings dataset, where users rate movies with stars, however following common practice [23, 28, 42] 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 [17, 18, 42] and recent [37, 51, 52] sequential recommendation papers, we conclude that it is well suited for the problem and it is important to include it as one of our benchmarks.

Yelp[1] - is a businesses reviews dataset. It is another popular dataset for sequential recommendations [2, 3, 30, 37, 46, 54]. As for MovieLens-20M, we consider all user reviews as positives.

Gowalla [7] contains user checkins 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 [18, 42, 54], we discard cold-start users with fewer than 5 interactions from each dataset. Table 1 reports the salient statistics of the three datasets.

5.3 Models

5.3.1 Experimental Architectures. We experiment using RSS with three recent model architectures for sequence recommendation: (i) *GRU4Rec* [16] is a sequential recommender architecture based on recurrent networks; (ii) *Caser* [43] applies a convolutional neural network structure for sequential recommendation. For our experiments, we use the basic architecture described in [50]; (iii) *SASRec* [18] is a sequential recommendation architecture based on transformers. 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 TensorFlow version 2 [1].[2] Note that for our experiments we reuse only the architectures of these models and not the training methods or hyper-parameters. 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;

[1] https://www.yelp.com/dataset [2] The code for this paper publicly available in a joint repository with our BERT4Rec replicability paper [32] https://github.com/asash/bert4rec_repro

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

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 [18]. In particular, we limit the length of the user sequences to 50 items for all three datasets, and 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 β_2 parameter set to 0.98. Following BERT4Rec [42], we set the maximum percentage of a sequence to sample, $\tau = 0.2$. Except where otherwise noted, we 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. 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 it is possible to achieve performance comparable with 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 [40]. We use the implementation of this recommendation model from the popular LightFM library [21].

We also use two Transformer-based models as state-of-the-art baselines: (i) *SASRec-vanilla* - the original version of SASRec recommender [18], 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 [42] 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. [19, 22, 26, 31, 33]).

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 BERT4-Rec publication [42] does not report the required amount of training, but we find empirically that reproducing the reported results takes around 16 hours on our hardware [31].

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 original implementation of BERT4Rec infeasible when a dataset has too many items. Indeed, the original BERT4Rec publication [42] only reports results on relatively small datasets with no more than 55000 items and our own attempts to train BERT4Rec on large Gowalla dataset with more than 1 Million items failed because of memory and storage issues (see also Section 6.4). Hence, we do not report BERT4Rec results for Gowalla and leave scaling BERT4Rec to datasets with large number of items for the future research.

5.4 Data Splitting and Evaluation Measures

Following many existing publications [18, 42, 43] 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 maximise 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[3] and Normalized Discounted Cumulative Gain (NDCG). For both metrics, we apply a rank cutoff of 10. To measure the significance of performances differences, we apply the paired t-test, and apply Bonferroni multiple testing correction, following recommended practices in IR [12].

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 [20] 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[4].

6 RESULTS

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

[3] In the context of sequential recommender systems, Recall corresponds to chances of correctly retrieving a single relevant item, and therefore many publications [18, 42, 50], call it Hit Ratio (HR). We prefer the more conventional Recall name for this metric.

[4] Indeed, we also found that conclusions could change using sampled metrics.

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 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 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†*

6.1 RQ1. 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 general magnitudes of the reported effectiveness results are smaller than those reported in [42] - indeed, as stated in Section 5.4, in contrast to [42], we follow recent advice [5, 20] 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 [18, 42]), however, applying RSS significantly improves its Recall@10, both when using BCE (0.153→0.188) and when using 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. NDCG@10 0.102 vs. 0.071 when using sequence continuation). We postulate that the large number of items in the dataset make the training task very hard, and only the combination of RSS with λRank allows to train the model with reasonable quality in the given time limit – this can be investigated further in future work.

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 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 ass Recall@10 in 8 cases out 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

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 (2)), to investigate its effect on effectiveness. In particular, as $\alpha \to 0$, the training task turns to sequence continuation, while large α the training task loses its sequential nature and becomes similar to matrix factorization.

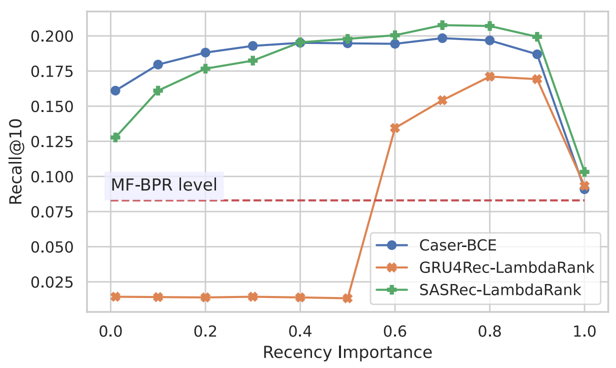
Figure 5 summarizes the impact of α on the model effectiveness. We also present the performance of the MF-BPR [40] baseline. From the figures, we observe that when we set α close to zero, the results match to those we report in Table 2 for the sequence continuation task, illustrating that under small 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 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 predicting the *next* item and predicts *some* item instead. Overall, the general trends visible in Figure 5 suggest that RSS allows to train effective models across a wide range of the α parameter values: for Caser and SASRec, large improvements over sequence continuation training is 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 are 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 $\alpha \leq 0.9$ result in effective performance for all three model architectures.

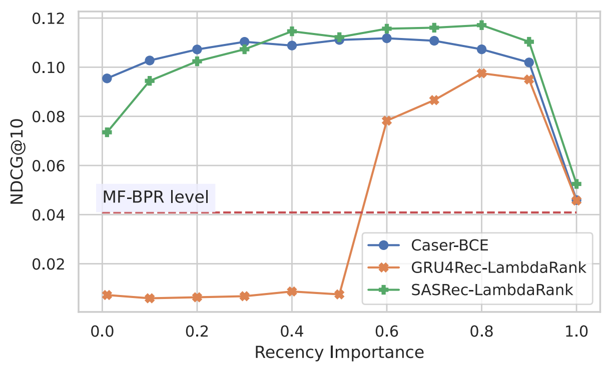
6.4 RQ4. Comparison with Baselines

To address our final research question concerning the comparison with baselines models, we compare the best performing RSS-enhanced model, SASRec-RSS, using both λRank and BCE 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 durations. 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 analysing 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 BERT4-Rec, SASRec-RSS exhibits 16% higher NDCG@10 (the significant



(a) Effect of position recency importance on Recall



(b) Effect of position recency importance on NDCG

Figure 5: SASRec, GRU4rec and Caser performance on the MovieLens-20M dataset, when trained with Recency-based Sampling of Sequences with the exponential importance function $f(k) = \alpha^{(n-k)}$, 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. Training time of all models is fixed at 1 hour.

improvement), despite needing only 6% of the training time (16h→ 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, BERT4-Rec underperforms compared to the SASRec-RSS (λRank or BCE version).

Overall, in answer to RQ4, 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.

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. Through experimentation on four datasets, we found that this relatively simple change in 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, 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. Finally, while we did not apply RSS to BERT4-Rec, due to its requirement of position embeddings (which it inherits from BERT), 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] 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.

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

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

[17] 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.

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

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

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

[21] 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.

[22] 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).

[23] 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).

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

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

[26] 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.

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

[28] 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.

[29] 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.

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

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

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

[33] 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*.

[34] 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.

[35] 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.

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

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

[38] 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.

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

[40] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proc. CUAI*. 452–461.

[41] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proc. WWW*. 811–820.

[42] 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.

[43] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proc. WSDM*. 565–573.

[44] 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.

[45] 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.

[46] Chenyang Wang, Weizhi Ma, and Chong Chen. 2022. Sequential Recommendation with Multiple Contrast Signals. *ACM Transactions on Information Systems (TOIS)* (2022).

[47] 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.

[48] 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).

[49] 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.

[50] 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.

[51] 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.

[52] 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).

[53] Pengyu Zhao, Tianxiao Shui, Yuanxing Zhang, Kecheng Xiao, and Kaigui Bian. 2021. Adversarial oracular seq2seq learning for sequential recommendation. In *Proc. IJCAI*.

[54] 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.

[55] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. 2001. Using temporal data for making recommendations. In *Proc. UAI*. 580–588.