DELFT UNIVERSITY OF TECHNOLOGY

MASTERS THESIS

Neural Session Recommendation

Author:
Jesse Harte

Supervisor:
Dr. Asterios Katsifodimos

Company supervisor:
Dr. Marios Fragkoulis

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Web Information Systems Group
Software Technology

Student number: 5637848
Thesis commitee: Dr. Asterios Katsifodimos
Dr. Merve Gürel
Dr. Marios Fragkoulis

November 23, 2023

DELFT UNIVERSITY OF TECHNOLOGY

Abstract

Electrical Engineering, Mathematics and Computer Science
Software Technology

Master of Science

Neural Session Recommendation

by Jesse Harte

In this thesis we aim to research and design different neural models for session recommendation. We investigate the fundamental neural models for session recommendation, namely BERT4Rec [87], SASRec [47] and GRU4Rec [40, 39] and subsequently use our findings to design a simpler but performant neural model.

Firstly, we address methodological errors made in the training and evaluation process of BERT4Rec, SASRec and GRU4Rec [38, 37, 71, 73, 52, 64] in order to create a fair comparison of the models' performance. We analyze the effect of several design decisions other than the architecture, and subsequently standardize these design decisions for all neural models. In the process, we discover that we can enhance the performance of the original implementations up to 250% in NDCG@10.

Having isolated the effect of the model architecture, we find that models are more similar in performance than previously reported [87, 47]. In addition, there is no consistently superior model. Moreover, we find that frequently-omitted non-neural baselines like SKNN [44, 64] and ItemKNN [81] can be extremely competitive and outperform our neural models on 2 out of 4 datasets. The neural models significantly outperform the non-neural baselines on the remaining 2 datasets.

Furthermore, we compare the recommendation behaviour between different neural models, and find that the recommendation behaviour of different models is both similar and predictable depending on the dataset. We use our baseline models to explain the behaviour of the models, and find that up to 80% of the top-ranked recommendations were also recommended by ItemKNN [81] or a simple first-order Markov Decision Process (MDP) [83].

Therefore, we explore how much performance we can maintain while simplifying the model architectures. We propose a simple embedding-based model, and find that it can equal or even outperform the neural models with up to 48% on NDCG@10 on 3 out of 4 datasets while being significantly easier to understand, faster to train and easier to tune. Moreover, the results show that we can capture almost all performance with a small subset of the original architectural components.

Our optimized implementations are open-sourced as part of our publication on using a large-language model to initialize item embeddings in [32].

Contents

List of Figures

List of Tables

List of Symbols

General symbols

s	A session.
t	A timestep. It can be used to index a session s, so that s_t represents the item at timestep t in session s.
u	A user.
i	An item.
$\mathcal{U}$	The set of users.
$\mathcal{I}$	The set of items.
$\mathcal{N}_s$	All neighbouring sessions of session s. These are all sessions that have an item in common with session s.
$\mathbf{1}$	The indicator function.

Symbols related to the neural models

N	The uniform length of the sessions to be fed to a neural model. In this thesis, all neural models process their sessions to be length N. Shorter sessions contain special padding items at the start of the session. Longer sessions are truncated so that only the last N items are kept.
R	A matrix of shape $(N, \lvert\mathcal{I}\rvert)$ where each entry $R_{t,i}$ denotes the predicted score that item i is the item at timestep $t+1$. In this thesis, the vector R_N has special significance because it denotes the predicted scores for the item at timestep $N+1$. The items with the highest scores in R_N will be recommended to the user.
e	The embedding dimension.
E	The item embedding matrix of shape $(\lvert\mathcal{I}\rvert, e)$. All neural models represent items through an embedding.
e_i	The embedding of item i.
E_s	The embeddings of the items in session s in matrix format.
W	A weight matrix.

Symbols related to transformer-based models

l	A transformer layer.
S	The output of a layer in a transformer. We use S^0 to denote to the output of the embedding layer of the transformer. Similarly, we use S^l to denote the output of transformer layer l.

Symbols related to GRU-based models

h_t	The hidden state vector of a GRU layer on timestep t.
$\hat{h}_t$	The candidate hidden state vector of a GRU layer on timestep t.
z_t	The update gate vector of a GRU layer on timestep t.
r_t	The reset gate vector of a GRU layer on timestep t.

Glossary

The following list consists of the most important terms that are used extensively throughout the thesis. The list is sorted alphabetically.

Bi/uni - directionality	The directionality of a neural model indicates whether it is restricted during training to exploit items succeeding timestep t to produce their predictions R_t for timestep t. A unidirectional model can only use the items at timesteps t and before. A bidirectional model can use all timesteps except t itself.
Context	A general term for the circumstances surrounding an item interaction. It could represent anything like the user's intent, temporal aspects like time of year, or the availability of an item. Since we only consider sessions as the source of information, context can only be deduced from the item interactions themselves.
Homo/hetero - geneity	A characteristic of a neural model. A homogeneous neural model assumes that item i is a good recommendation for item j if item j is a good recommendation for item i. In contrast, a heterogeneous model assumes that this implication does not hold.
Embedding	The numerical representation of an item or position, typically learned through backpropagation by the neural models. A *session embedding* is the numerical representation of a session computed from the items in the session, so note that these are not stored like the item embeddings. All neural models introduced in this thesis differ in how they compute the session embedding, but all of them simply multiply the session embedding with the item embedding matrix to produce scores for each item.
Interaction	An interaction between a user and item. It is a general term to refer to anything like a click, purchase or watching a video for longer than x seconds. We use an interaction as a signal of preference of a user for an item.
Interaction vector	A binary multi-hot vector of dimension $\|\mathcal{I}\|$ that indicates the items that a user or session interacted with. Note that this representation does not contain information on the ordering of the interactions.
Item catalogue	The set of all items, denoted by $\mathcal{I}$.

Next item	The withheld last item of the session. We evaluate models by checking whether the next item is in the recommendation slate for the session. The next-item objective is the training task that trains a neural model to predict the item s_{t+1} at position t with its output R_t.
Offline/online	The context in which models are evaluated. Online evaluation entails measuring business metrics from a recommender system in production. Offline evaluation entails evaluating models by comparing their recommendation slates with withheld items from the evaluation sessions. Offline evaluation is used as a proxy to estimate online performance.
Popularity	The amount of times an item appeared in the set of training sessions.
Rank	The rank of an item in a recommendation slate is the index at which it appears in the list. A rank of 1 indicates the recommended item with the highest confidence by the model.
Recency	The recency of an item interaction refers to the relative position of the item interaction in the session. A recent item is an item that appears later in the session. In contrast, old or historical item interactions appeared earlier in the session. Recent items are generally more indicative of the next item.
Session	An ordered sequence of interactions. An *input session* is session in the form that it is fed to a neural model. A *training session* is an input session to be used for training a model. Likewise, an *evaluation session* is an input session to be used for evaluation. The models are tasked with predicting the identity of the withheld last item of the evaluation session.
Slate	A ranked list of recommendations. The item at index 1 in the slate indicates the recommended item with the highest confidence by the model.
Target item	A term to refer to the ground-truth item for prediction R_t at timestep t. R_t is compared with the one-hot vector indicating the target item in order to compute the loss. In the next-item objective, the target item for position t is item s_{t+1}. In the Cloze objective, the target for position t is item s_t.
Timestep	The index of an interaction in the session. We use s_t to denote the item interaction at timestep t in session s. For example, s_1 is the first item of the session.

Chapter 1

Introduction

A Recommender System (RS) is a system that selects a subset of items from a catalogue to be presented to the user. In this definition, *item* is used as a general term and refers to anything that we might want to recommend to the user. For example, a recommender system on an e-commerce platform recommends products to buy whereas a recommender system on a streaming platform recommends movies to watch. In its simplest form, a recommender system bases its recommendation decisions on the previously consumed items by the user. This thesis will focus on using neural networks for session recommendation, which is a specific recommendation task that aims to predict the next item interaction given only a short ordered sequence of previously interacted items.

1.1 Motivation for recommender systems

Recommender systems are necessary to condense or *filter* the item catalogue to a subset whose size is presentable to the user. Simply presenting every item in the catalogue typically overwhelms the user simply due to the large number of items available [78]. Business psychologists refer to this phenomenon as *information overload* or *the paradox of choice*, and consense that having an excessive number of choices leads to worse decision-making and user well-being in general [82, 86, 80]. Hence, recommender systems are inherently necessary to facilitate the user's decision-making process. From a business perspective, recommender systems are also effective tools to increase item consumption [1], as users are more likely to consume items that relate to their preferences. Moreover, the user experience is also enhanced, resulting in more user retention and more item consumption in the long run [1].

1.2 Recommendation tasks

Recommendations are typically computed using the user's previous item *interactions*. The specific definition of an interaction depends on the use-case, but examples include a click on the item, a purchase of the item, or watching the item (a short video) longer than x seconds [107]. Given the set of interactions from a user, we can then compute recommendations by exploiting the interactions from users with similar interactions. The problem of computing recommendations using only the set of item interactions is commonly referred to as *collaborative filtering*.

In recent years, more elaborate problem formulations came into existence, most importantly including *sequential* recommendation. In sequential recommendation, the order at which a user interacted with the items is also known, which allows for modelling dynamic user preferences and behaviours [87]. A subvariant of sequential recommendation is *session* recommendation, where the system does not have access

to all historic user interactions, but instead only to the interactions from the current session, i.e. the most recent ones [40]. In this thesis, a shopping cart will be the prime example of a session. The purchased products are then the items in the session, and the ordering of the session is the ordering at which the products were added to the shopping cart. Important to note is that the user is anonymous in session recommendation, and so all the system can exploit for computing the recommendations is a short sequence of interactions. The rationale behind session recommendation opposed to sequential recommendation is that the recommendations are usually more relevant to the user's current intent. Furthermore, as the system can not bias towards recommending older historic interactions[1], diversity and exploration of the items is enhanced [103].

1.3 Collaboration with Delivery Hero Research

This thesis was written in collaboration with Delivery Hero Research. This collaboration allows us to evaluate recommendation models on real session datasets. Our unified goal is to improve the performance of recommendations on their checkout pages, visualized in Figure 1.1. Hence, in this use-case sessions represent shopping carts. The rationale for session recommendations holds for this use-case as well, namely that session recommendation focuses on the user's current intent, which in turn enhances diversity and exploration of the items. Furthermore, we believe it to be good practice to optimize CTR with a minimal amount of information as possible (the current session) before adding other sources of information like user identity or historical user interactions.

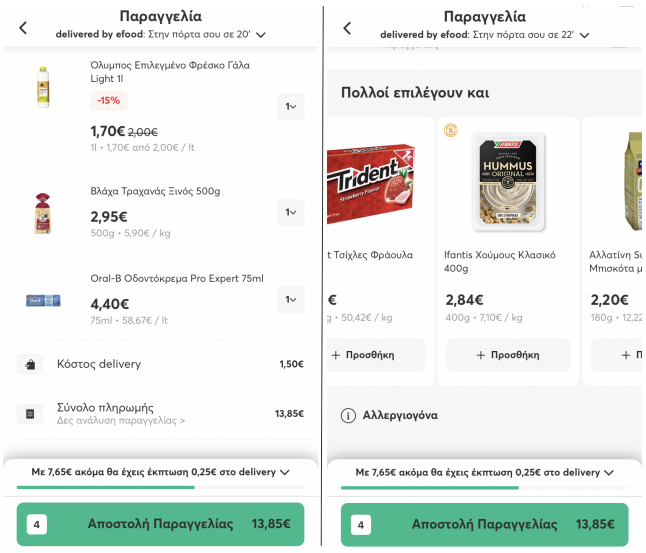

FIGURE 1.1: The checkout page of DH Greece. The left figure shows an order to be paid. The right figure shows the recommendations at the bottom of the same page. The thesis will aim to improve the quality of these recommendations.

[1] We have found the *Previously-bought* baseline to be an extremely competitive baseline.

1.4 Evaluating recommender systems

Given a set or sequence of item interactions, the *recommendation model* is tasked with computing a ranked TOP-K [2] list of items that the user will likely be interested in, the so-called *recommendation slate*. In an ideal scenario, we would compute and present the recommendation slates *online* to actual users in order to test the quality of a model. However, this may be costly, as bad recommendations may harm user experience and decrease profit directly [1]. Instead, recommender systems are typically first evaluated *offline*. To emulate the online setting, we typically withhold a subset of a user's interactions from the input to the recommender system, and compute offline metrics by comparing the TOP-K recommendation slate with the withheld subset of interactions. The core assumption of offline evaluation is then that good recommender systems will have the withheld items in their recommendation slates. We will further formalize the recommendation task in chapter 2 and discuss offline metrics in detail in section 3.3.

1.5 Models for recommendation

Traditionally, approaches like Matrix Factorization [42, 51] and k-Nearest Neighbours [81] have been at the center of the research field and served as the core of many successful variants [88]. Later, the advent of deep learning reached the field of recommender systems, and many neural-based recommender models were introduced [11, 33, 14]. For sequential recommendation, we observe a strong pattern where innovations in Natural Language Processing (NLP) are adapted to fit the sequential recommendation task. For example, the invention of the GRU layer [12] was adapted for sequential recommendation in GRU4Rec [40]. Once the novel transformer architectures were introduced for NLP [93, 16], SASRec [47] and BERT4Rec [87] were proposed. The publications behind these models have each claimed superiority over their respective baselines, and have thus sparked a multitude of variations that again claim to be state-of-the-art.

1.6 Motivation and scope

However, numerous studies have pointed out methodological errors in many of the aforementioned studies, and found that traditional models often outperform neural models when properly tuned [71, 64, 37, 38, 84, 24]. As a result, there is no consensus on what can be considered the state-of-the-art. Therefore, the first goal of this thesis will be to establish the state-of-the-art for session recommendation. We will focus on the fundamental neural models for this use-case, namely GRU4Rec [40], SASRec [47] and BERT4Rec [87] in addition to the competitive baselines SKNN [64], ItemKNN [81] and a first-order Markov Decision Process (MDP) [83]. Moreover, some of the methodological errors in evaluation can be attributed to the complexity of the models, which might cause researchers to adopt faulty third-party implementations of baselines out of convenience [37] and/or do not properly tune all hyperparameters for each baseline [37, 84]. Therefore, the second goal of this thesis is to expose the inner workings of the three vanilla models in order to explain why and when (i.e.

[2] Naturally, the value of K depends on the use-case. For example, in next-track music recommendation, models are specialized in TOP-1 recommendation, as a user can only listen to one song at the time. In contrast, a user can usually view multiple products on an e-commerce webpage, and so higher values for K (i.e. 10) are more suitable.

on what datasets) these models are effective. Lastly, we aim to leverage these new insights to devise simpler models while maintaining or improving performance.

1.7 Research questions

Given the motivation and scope, we devise the following research questions.

1). What are the methodological errors made in the design and evaluation of the models, and how can we create an evaluation setup that does not suffer from these issues?

2). What models are most effective for sequential recommendation on our datasets?

3). How do these models' recommendations compare on various dimensions?

4). Given these findings, can we devise simpler models while maintaining or improving performance?

1.8 Thesis outline

The rest of this thesis is structured as follows. We discuss the background on the recommendation tasks, incumbent session-based recommendation models and their variants, and reproducibility studies in chapter 2. We then introduce our setup for evaluating the recommendation models in chapter 3. We then discuss all steps necesssary to produce a fair evaluation of the models, including the implementation details, hyperparameters, their search space and baselines, and a discussion on the results in chapter 4. We then continue with an analysis of the models' recommendations in chapter 5. Given these results, we create a performant but simple model called LastEmbedding, which we then use to analyze the performance contribution of several architectural components from the original models in chapter 6.

Chapter 2

Background and related work

We prelude our background work with a general introduction to the various recommendation tasks, including sequential and session recommendation. We then continue by introducing the current state-of-the-art (SOTA) session-based recommendation models, namely SKNN [44, 64], GRU4Rec [40], SASRec [47] and BERT4Rec [87]. The aforementioned models are the ones we will implement, analyze and aim to improve. We will discuss these models and the formulas that govern them in depth, as these specificities will facilitate our analyses in chapter 5 and chapter 6.

2.1 Matrix-based recommendation

Explicit feedback In its infancy, the research field of recommender systems was concerned with rating prediction. This entails predicting the missing values in a rating matrix of shape $(|\mathcal{U}|, |\mathcal{I}|)$, where $\mathcal{U}$ denotes the set of users and $\mathcal{I}$ denotes the set of items. Each user has rated a subset of the items on a certain numerical scale. This type of feedback is called *explicit feedback*, because the user has explicitly and consciously provided the ratings on the items. The recommendation task is then to predict the numerical rating the user would give to items he/she has not rated, with the end goal of recommending the unrated items with high predicted ratings to the user [88]. Algorithms like ItemKNN [81] and Matrix Factorization [42, 51] have proven to be very successful at this task [88], and rating prediction is generally considered to be solved by the community [33].

Implicit feedback However, explicit feedback is generally difficult to gather as it requires explicit effort from the user. Instead, research has shifted towards *implicit feedback*, which is a form of feedback that is unconsciously provided by the user [33]. It works on the basis of *interactions* between users and items, which could be anything ranging from clicks, purchases or watching a video longer than x seconds [107].

Evidently, implicit feedback is more readily available as it takes no additional effort for the user to indicate their preferences [33]. However, this type of feedback is noisy in the sense that an interaction does not necessarily imply that the user likes the item. Conversely, the absence of an item interaction does not imply that the user dislikes the item, as it could very well be the case that the user is unaware of the item. Furthermore, though implicit feedback provides more information per user and item than explicit feedback in absolute terms, the scale at which recommender systems are deployed has grown substantially too. For example, large e-commerce sites can nowadays have catalogues of millions of items [104]. As a result, recommender systems must find relevant recommendations from larger pools of candidates with relatively less information [43]. Moreover, processing data at such scales demands

more compute and memory requirements. These demands render many traditional machine learning techniques infeasible, as they will be too demanding for typical hardware [43].

In matrix-based recommendation, implicit feedback is stored as a matrix where an entry on row (user) u and column (item) i is a binary variable to indicate whether user u has interacted with item i. The task for matrix-based recommendation with implicit feedback is roughly similar to the task of rating prediction in the sense that the matrix needs to be reconstructed so that the items with the highest predicted entries will then be recommended to the user.

2.2 Sequential and session recommendation

However, this matrix-based representation of implicit feedback disregards the ordering in which these interactions occurred. In some use-cases this ordering might be informative, which necessitated the formalization of a new recommendation task, namely sequential recommendation. Given an ordered item-interaction sequence $s = (s_1 \dots s_n)$ of n timesteps, sequential recommendation models are tasked with producing the probability over all the items for the identity of the item on timestep $n+1$, denoted $P(s_{n+1}|s)$ [87]. In recommendation literature, the item on timestep $n+1$ is commonly referred to as the *next item*. To reiterate, the items with the highest predicted probability will be recommended to the user.

The task formalization of session recommendation is exactly the same, but the lengths of the sequences are typically shorter because they include only the most recent interactions of a user. Still, there is no precise definition on what differentiates session from sequential recommendation from a formalization perspective, and we find that models are typically evaluated on both long and short sequences [87, 23, 47]. Therefore, we decide to adopt a variety of datasets to cover both sequential and session recommendation in chapter 3.

2.3 Models for session recommendation

2.3.1 SKNN

Session-k-Nearest-Neighbours (SKNN) [44, 64] finds items to recommend by considering sessions similar to the input session. The model considers the top-K most similar neighbours $n \in \mathcal{N}_s$ of a session s with a given similarity function $\text{sim}(n, s)$, and computes the score of an item i with

$$\text{score}(i, s) = \sum_{n \in \mathcal{N}_s} \text{sim}(n, s) \cdot \mathbf{1}(i \in n) \tag{2.1}$$

Here, $\mathbf{1}$ denotes the indicator function. The scores from Equation 2.1 can be normalized to produce the probability distribution $P(s_{n+1}|s)$. Though the similarity function is intentionally left general, cosine similarity between the sessions' binary interaction vectors has proven to work well in practice, and is therefore considered to be the default similarity function. SKNN has been shown to work well with relatively few data points and can therefore also be leveraged to model trends by only considering recent sessions [44].

Several variations of SKNN have been proposed [44, 64], including

- Sequential SKNN (S-SKNN). This variant considers the ordering of the items. It weighs the indicator function using the relative position of the item i in the

neighbouring session n with weight $w(i_x, n) = i_x/|n|$, where i_x is the position of i in session n, and $|n|$ denotes the length of the session.

- Vector Multiplication SKNN (V-SKNN). This variant also exploits the ordering of the items in the session by weighing down items earlier in the session in the interaction vector. As a result, the later (more recent) items gain more influence on the similarity score between two sessions.
- Sequential-Filter (SF-SKNN). This variants filter items from the recommendation slates if it has not occured directly after the last item of the session at least once in the training data.

K-Nearest-Neighbours has repeatedly been shown that it is a very competitive baseline, and often beats neural models if properly tuned [66, 44]. In fact, we will see how SKNN is also a competitive recommendation model for our use-cases, despite its relative simplicity in chapter 4.

2.3.2 GRU4Rec

Arguably the first work to use a deep learning method for the session-based recommendation use-case, GRU4Rec [40] employs a recurrent neural network based on the Gated Recurrent Unit (GRU) [12]. We visualize GRU4Rec in Figure 2.2. We first introduce the training task of GRU4Rec before we can discuss the architecture.

Training task GRU4Rec is trained by sequentially processing sessions, where on each timestep it must use the current item in the session to predict the item at the next timestep. Hence, the input to the model at each timestep t is the current item in the session s_t. Its output is then a score for each item, which we will denote by R_t for the remainder of this thesis. So, R_t represents the model's confidence for each item that it will be the item in the session on timestep s_{t+1}. Using R_t we can compute loss or, by taking the items with the TOP-K scores on the last timestep, compute next-item recommendations. Instead of training on a single session at a time, Hidasi et al. [40] construct *session-parallel mini-batches* so that GRU4Rec can train on multiple sessions in parallel. We explain the session-parallel mini batch construction by means of an example in Figure 2.1.

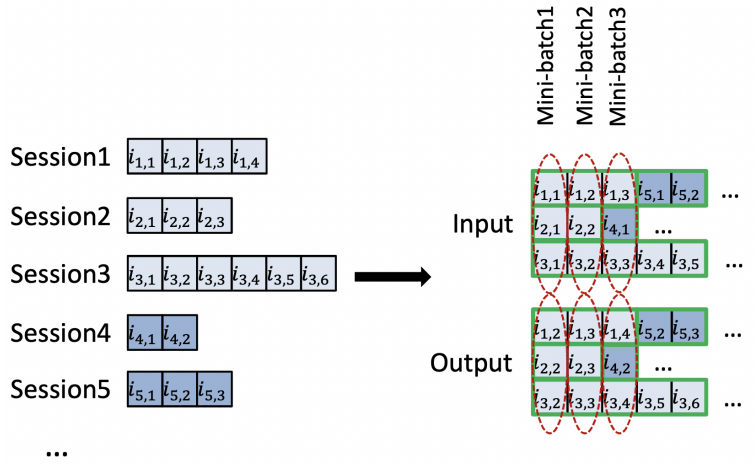

FIGURE 2.1: GRU4Rec's session-parallel mini batch construction. Figure from [40]. Sessions 1, 2 and 3 are initially sampled and their first items make up the first mini-batch (input). GRU4Rec is then tasked with predicting the second item of each session (output). The second batch consists of the second item of each session, with the task of predicting the third item of each session. Session 2 only has two items and is therefore replaced by session 4 in the third batch. Note that the last item of each training session can not be used as input, since there does not exist a next-item for the last item in the training set.

Architecture The precise architecture of GRU4Rec is kept general in [40], but it essentially consists of an item embedding layer, one or multiple GRU layers and a prediction network.

The item embedding layer consumes the item ID s_t at the current timestep t and returns its associated embedding e_{s_t} from the item embedding matrix E. This item embedding matrix is of shape $(|\mathcal{I}|, e)$, where we use e to denote the embedding dimension.

The embedding e_{s_t} is fed to a GRU layer [12]. A GRU layer is stateful, meaning that it maintains a *hidden state* h_t while processing a session. This allows the model to use information from the previously seen items $s_{0\ldots t-1}$ next to item s_t to predict s_{t+1}. To compute h_t, the GRU layer does not only use s_t, but also its previous hidden state h_{t-1} (which contains information on previously-seen items). We present the equations that govern the GRU layer in Equation 2.2. We refer to the vectors z_t as the *update gate*, $\hat{h}_t$ as the *candidate activation* and r_t as the *reset gate*.

$$\begin{aligned} h_t &= (1 - z_t) \otimes h_t + z_t \otimes \hat{h}_t \\ z_t &= \sigma(W_z e_{s_t} + U_z h_{t-1}) \\ \hat{h}_t &= \tanh(W e_{s_t} + U(r_t \otimes h_{t-1}) \\ r_t &= \sigma(W_r e_{s_t} + U_r h_{t-1}) \end{aligned} \tag{2.2}$$

Here, W_z, H_z, W, U, W_r and H_r denote different weight matrices and $\otimes$ denotes the Hadamard (element-wise) product. At each timestep, the prediction network consumes the hidden state h_t to produce the scores R_t for the target items.

In [40], a simple feed-forward network is suggested for the prediction network, where the last layer's output is R_t of dimension $|\mathcal{I}|$. However, no exact specification of the prediction network is given in [40].

Loss and negative sampling Besides the architecture, the loss function is also kept general. Hidasi et al. [40] experiment with standard categorical cross-entropy, Bayesian Personalized Ranking [77] and TOP1 loss [40]. In all cases, the *target vector* at timestep t is a one-hot vector indicating the *ground-truth* item s_{t+1}. The ground-truth item is also commonly referred to as the *positive* item. All other items are referred to as the *negative* items for which the target is zero.

Note that for small datasets it is feasible to produce the score R_t for each item in $\mathcal{I}$. However, on larger datasets this becomes a scalability bottleneck. Instead, *negative sampling* is used to train GRU4Rec. This entails sampling a subset of the negative items and only computing R_t for this subset. As a result, the loss and backpropagation become much faster to compute because their computational load becomes independent of the number of items in the dataset.

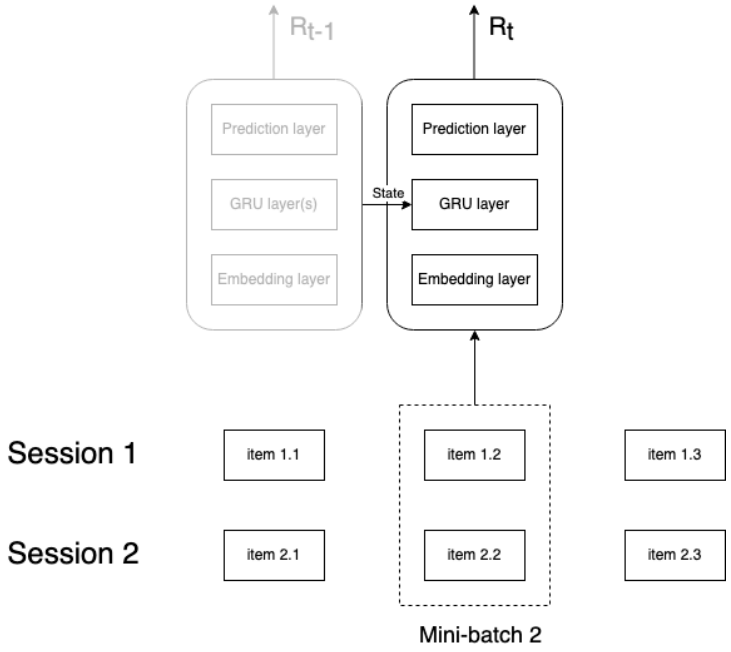

FIGURE 2.2: GRU4Rec's architecture and training task. At mini-batch 2, we feed the model with the items s_2^1 and s_2^2. In addition, the state resulting from mini-batch 1 is fed to the GRU layers during mini-batch 2.

2.3.3 GRU4Rec variants

Vanishing gradients Hidasi and Karatzoglou [39] revisit GRU4Rec and address the limitations of the initial version proposed in [40]. They find that the BPR and TOP1 loss functions suffer from vanishing gradients so that learning prematurely stops, and propose the adjusted BPR-Max and TOP1-Max as mitigations. It was also found that cross-entropy is most consistently a good loss function when no negative sampling is used. Furthermore, they reduce model size and overfitting by re-using the item embedding matrix E to compute the output target scores, which effectively replaces the last layer of the prediction network. *GRU4Rec* is now commonly used to refer to the variant in [39] instead of [40].

Exploiting future timesteps NARM (Neural Attentive Recommendation Machine) [56] enhances GRU4Rec by not only using the current activation h_t for predicting

R_t, but also the weighted average of other timesteps $\sum_{t'} a_{t't} h_{t'}$. The weights $a_{t't}$ are calculated from an attention mechanism between h'_t and h_t. Given the current activation and the weighted average of other activations, their concenation is then fed to the prediction network to compute R_t. Tan, Xu, and Liu [89] also try to improve the exploitation of other timesteps by randomly dropping items in the sessions. Intuitively speaking, dropping item s_t makes the model learn a stronger connection between items s_{t-1} and s_{t+1}, which can be beneficial for sparse datasets. They also train a GRU model with the reversed sessions, and use its predictions for R_t as a target vector to soften the loss for the forward model. This effectively allows the model to exploit information from timesteps after t.

Ensembling GRU4Rec with SKNN Jannach and Ludewig [44] find that the original GRU4Rec model (from [40]) is often outperformed by a properly-tuned SKNN model [44, 64]. However, a weighted ensemble between GRU4Rec and SKNN is more effective than the standalone models, supporting the hypothesis that GRU4Rec (and neural models in general) can be leveraged to learn patterns beside simple co-occurence counts like in SKNN [44, 64].

2.3.4 SASRec

SASRec (Self-Attentive Sequential Recommendation) [47] is the first transformer architecture [93] designed for the sequential recommendation task. We visualize the model in Figure 2.3.

Training task First of all, transformer architectures require their input to be *rectangular*. Therefore, Kang and McAuley [47] introduce hyperparameter N, so that all sessions are either truncated to only contain the last N items, or left-padded (added to the start of the session) to become length N with special `<padding>` items. Like GRU4Rec, the model is trained by predicting the identity of s_{t+1} for each timestep t, but in contrast to GRU4Rec, SASRec computes R (of shape $(N, |\mathcal{I}|)$) for all timesteps $t \in [1, N]$ at once. As a result, SASRec's training batches consist of whole (but processed to be length N) sessions. The last item of each training session can not be used as input, since there does not exist a next-item for the last item in the training set (See Figure 2.3). We will refer to this training task as the *next-item objective*.

Architecture The architecture of SASRec consists of an embedding layer, one or multiple transformer layers, and a prediction network. The transformer layers are based on the encoder stack introduced in [93]. In short, each of the L transformer layers consists of a self-attention and a feed-forward module. We will first introduce the two different modules, after which we will explain how they are used in the transformer layer. Furthermore, we will denote the output of each transformer layer l with S^l (of shape (N, e)). Likewise, we denote the output of the embedding layer with S^0.

Self-attention module We can now detail the self-attention module SA, which is described by Equation 2.3. Given a sequence S, either from the embedding layer or a preceding transformer layer, the self-attention module SA is governed by Equation 2.3.

$$\begin{aligned} \text{SA}(S) &= \text{Attention}(SW^Q, SW^K, SW^V) \\ \text{Attention}(Q, K, V) &= \text{softmax}\left(\text{CausalMask}\left(\frac{QK^T}{\sqrt{d}}\right)\right)V \end{aligned} \tag{2.3}$$

Here, W^Q, W^K, W^V are called the query, key and value matrices respectively, and are all of the shape (e, e). The intuition behind Equation 2.3 is that the self-attention module essentially takes a weighted average of all the embeddings in the sequence, for each timestep in the sequence. To see this, note that $\text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$ (the *attention matrix*) is a matrix of shape (N, N). These weights are then multiplied with S, which constitutes taking the weighted average. The result is finally projected with matrix W^V. The CausalMask operator sets all attention weights above the diagonal to zero, which is necessary to ensure causality and prevent information leakage. Without this constraint, the model can cheat by using the embedding at timestep $t+1$ on timestep t to predict the item at timestep $t+1$. In Figure 2.3, the CausalMask was responsible for removing the arrows from s_2 to timestep 1 and the arrows from s_2 and s_3 to timestep 2.

Positional embeddings Furthermore, note that the multi-head attention module is inherently unable to recognize position, as $QK^T = (SW^Q)^T(SW^K)$ is invariant to the ordering of the rows. The entry i, j in the attention matrix only depend on S_i and S_j, but not on the specific values of i and j. Intuitively speaking, this should be sub-optimal, as neighbouring items in a sequence should be more related. To inject positional information into the module, the embedding layer of SASRec adds a learnable positional embedding matrix P to E_s.

Feed-forward module The feed-forward module is defined by Equation 2.4. Given a sequence of embeddings S, it processes each embedding S_i at position i separately with a shared two-layer feed-forward network.

$$\text{FFN}(S_i) = \text{ReLU}(S_i W^{(1)} + b^{(1)})W^{(2)} + b^{(2)} \tag{2.4}$$

Transformer layer Given the definitions of the self-attention and feed-forward modules, every transformer layer l transforms the sequence S^{l-1} into a new sequence S^l following Equation 2.5.

$$\begin{aligned} S^l &= g(\text{FFN}, g(\text{SA}, S^{l-1})) \\ g(f, x) &= x + \text{Dropout}(f(\text{LayerNorm}(x))) \end{aligned} \tag{2.5}$$

Hence, besides the core component of each module, every module additionally contains layer normalization [4], dropout and a residual connection. These components are necessary to reduce overfitting [47].

Prediction network Given the output sequence of embeddings S^L of the last transformer layer, the prediction network of SASRec projects this sequence into scores over all the items. To do this, the prediction network reuses the item embedding matrix E from the embedding layer, and computes item scores according to 2.6.

$$R = S^L E^T \tag{2.6}$$

Here, R is a NxI matrix, where $R_{t,i}$ denotes the models' confidence that item i will be the item at position $t+1$. For next-item prediction, we therefore use R_N, which denotes the vector containing the scores across items to be the item *succeeding* the last item in the session. Note that R_N is computed from S_N^L. At this point, we note that both SASRec and GRU4Rec compute an embedding (the hidden state h_N for GRU4Rec, S_N^L for SASRec) that is eventually multiplied with the transpose item embedding matrix. In recommendation literature, this embedding is referred to as the *session embedding*. All publications that we discuss in this chapter essentially assume that the models learn to contain all information about the session into this session embedding, after which a simple dot-product similarity computation is used to compute the scores R_t and the TOP-K recommendation slate.

Loss SASRec is trained using binary crossentropy (BCE) [29], using the positive sample and a single negative sample. So, for each timestep t, we take the score for the item s_{t+1} at $R_{t,s_{t+1}}$ (the positive sample), and the score for a random item i at $R_{t,i}$ (the negative sample). The negative sample is necessary to have a downward foce on the scores, as without it, the model could simply learn to always output 1 for all items with zero loss.

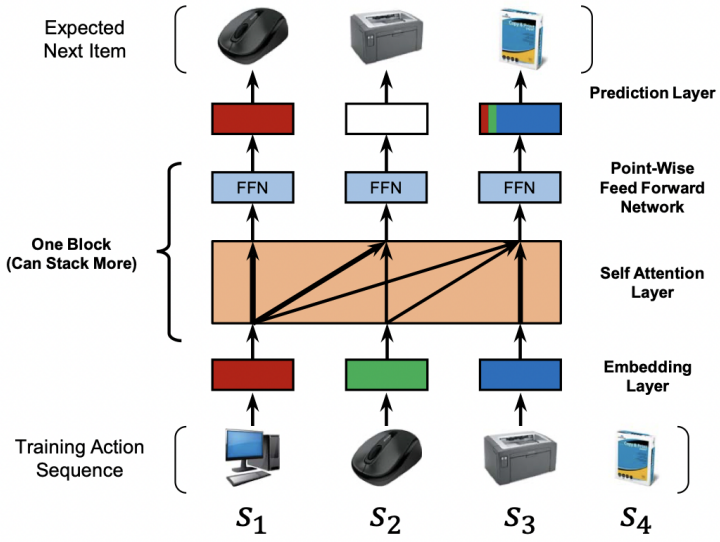

FIGURE 2.3: SASRec's architecture and training task. Figure from [47]. To predict item s_3, SASRec can use the item embedding from items s_1 and s_2, but not s_3 (or potentially s_4) itself because this connection is prohibited by the CausalMask operator. Important to note is that the FFN is shared across timesteps, and that the residual connections, dropout and layer normalizations are not visualized in the figure.

2.3.5 BERT4Rec

Motivated by the success of BERT for NLP tasks [16], BERT4Rec (Bidirectional Encoder Representations from Transformer for Recommendation) [87] applies BERT's innovations to the sequential recommendation problem. We visualize the model in Figure 2.5.

Training task In essence, BERT4Rec is very similar to SASRec. The main difference is that BERT4Rec drops the causality constraint in the attention module, and instead masks items in the input. More specifically, BERT4Rec is trained using the *Cloze* objective [92, 16], which involves masking random items in the sessions by replacing them with special `<mask>` items [1]. The model is then tasked with predicting the identity of the items at the masked positions. The Cloze objective allows BERT4Rec to use items succeeding a masked item to predict its identity without suffering from information leakage. This makes BERT4Rec a *bidirectional* model, as opposed to SASRec which is a *unidirectional* model due to the causality constraint.

Every training epoch, we convert a training session into one or multiple *input sessions* [2] with randomly-masked items. This requires two additional hyperparameters, namely the masking probability ρ at which an item is masked, and the number of randomly-masked input sessions. The latter is not further specified in the paper, but in the official open-source implementation[3], we find that its equal to 10 by default.

Sun et al. [87] note that the Cloze task does not directly correspond to the prediction task. During evaluation we have access to all items in the session instead of the masked session during training. Therefore, Sun et al. [87] also add an input session where only the last item is masked in order to emulate the evaluation setting. We visualize this construction in Figure 2.4. Note that during prediction, the `<mask>` item is simply appended to the sequence.

In [87], Sun et al. explain that the Cloze objective yields more training inputs than the conventional next-item objective (in SASRec and GRU4Rec). As such, they hypothesize it alleviates overfitting and thus adds to performance. To confirm this, Sun et al. [87] construct an experiment where only one item per training session is masked (thus yielding as much / fewer training inputs than the next-item objective), and still observe a performance improvement in comparison to SASRec. We will later revisit this conclusion in chapter 4.

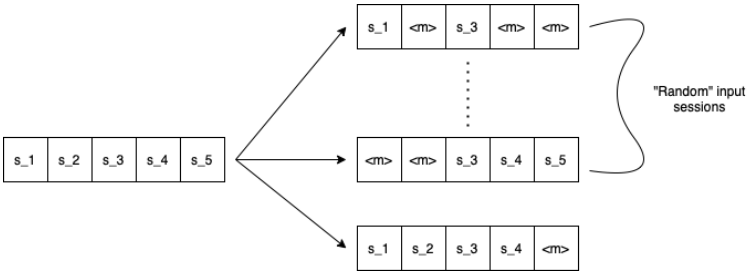

FIGURE 2.4: BERT4Rec's conversion of a training session into input sessions. Every training session is converted into one or multiple input sessions with randomly masked items, and a single input session where only the last item is masked.

Architectural differences with SASRec Besides the new training objective, BERT4Rec is slightly different from SASRec in terms of architecture. Most notably, BERT4Rec uses *multi-head attention*. This involves a slight adaptation of Equation 2.3 so that

[1] The Cloze objective is more commonly-known as "Fill in the gaps", and originally designed for sentence reconstruction. Given a sentence, "Jesse did not feel like _ his thesis, so he grabbed some _ for motivation", the task of Cloze is to predict the missing works, namely "writing" and "coffee".

[2] With *input sessions* we mean sessions in the format that they are expected by the model.

[3] `https://github.com/FeiSun/BERT4Rec/blob/master`

$$\begin{aligned} \text{SA}(S) &= \text{concat}([\text{SA}_1(S) \ldots \text{SA}_h(S)])W^O \\ \text{SA}_i(S) &= \text{Attention}(SW_i^Q, SW_i^K, SW^V) \\ \text{Attention}(Q,K,V) &= \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \end{aligned} \tag{2.7}$$

where each SA_i represents one of the h attention heads. In order to ensure $\text{SA}(S)$ has shape (N, e), each attention head has shape $(e/h, e/h)$ meaning their respective query, key and value matrices have this shape. The concatenation of the attention heads' outputs then results in the correct dimensions for $\text{SA}(S)$. Intuitively, the multi-head attention mechanism should enable the model to look at different parts of the session, making the model more expressive and thus enhancing performance [87].

A small detail is that BERT4Rec uses a slightly different transformer architecture, as the ordering of the layer normalization, dropout and residual connection is different. In BERT4Rec, the function g from Equation 2.5 is replaced by the definition in Equation 2.8..

$$g(f,x) = \text{LayerNorm}(x + \text{Dropout}(f(x)) \tag{2.8}$$

Lastly, the prediction network in BERT4Rec is slightly more involved. Equation 2.9 shows it contains an additional feed-forward layer and a bias, next to the reuse of the item embedding matrix like SASRec.

$$R_t = \text{softmax}(\text{GeLU}(S_t^L W + b^p)E^T + b^o) \tag{2.9}$$

Note that we define R_t (the predictions for timestep t), not R (predictions for all timesteps), as BERT4Rec only predicts for masked positions. Also, GeLU (Gaussian Error Linear Unit) is a smoother variant of the ReLU activation function [36].

In chapter 4, we will examine the effects of these architectural differences on the model performance.

Loss In contrast to SASRec, BERT4Rec does not use negative sampling and instead computes categorical cross-entropy (Softmax) loss over all the items for all masked positions. A seemingly-minor but vital detail is that for timestep t, the scores R_t represent the model's confidence on the identity of masked item s_t, instead of s_{t+1} in SASRec.

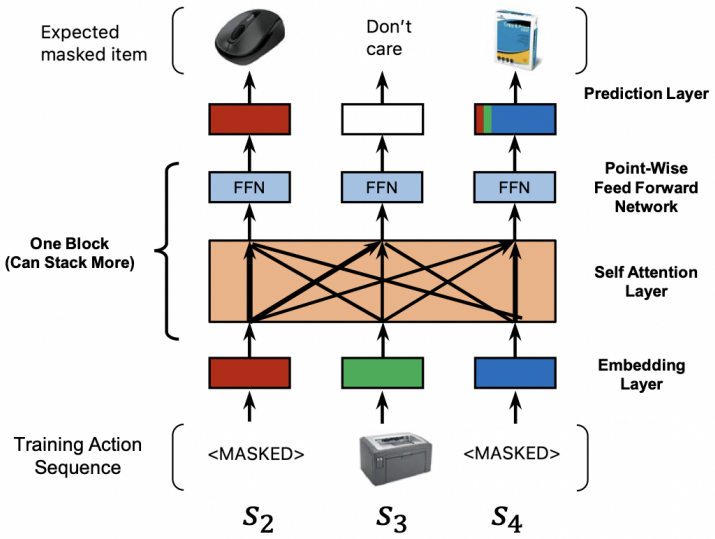

FIGURE 2.5: BERT4Rec's architecture and training task. Figure adapted from [47]. BERT4Rec needs to predict all masked items, in this case s_2 and s_4. Since these items are masked, there is no information leakage and the attention module can attend to all items in the session. Important to note is that the FFN is shared across timesteps, and that the residual connections, dropout and layer normalizations are not visualized in the figure.

2.3.6 SASRec and BERT4Rec variants

There is a variety of work that build upon the transformer-based SASRec and BERT4Rec models. We roughly categorized the different streams into variations that (1) introduce architectural improvements, (2) introduce training and loss improvements, and lastly, (3) employ contrastive learning. We also note there is a plethora of works on exploiting side-information for sequential recommendation using transformer architectures [108, 106, 75, 76, 9, 98, 60, 55, 105, 91], but for the sake of brevity and relevance we refrain from discussing those here.

Architectural improvements LSAN [59] aims to reduce the parameterization and complexity of SASRec. First of all, an item embedding is the sum of several shared base embeddings, where the base embedding matrix is much smaller than the original item embedding matrix of SASRec. Instead of multiple transformer layers, they use a single self-attention and convolutional head, allowing for a focus on both long-term and short-term context respectively. We will confirm that reducing the parameterization may enhance performance in chapter 6. Similar to LSAN, Locker [35] aims to promote short-term context in BERT4Rec by replacing some heads in the multi-head attention mechanism with other mechanisms focusing on exploiting *local* information. In recommendation literature, the *local* context is often used to refer to the items close to the current timestep, whereas the *global* context is often used to refer to the whole session. The proposed mechanisms in [35] include a GRU encoder [12], a fixed mask between distant items, an initialization approach to promote zero attention between distant items, and an additional neural network to make the attention mechanism more expressive. We present similar ideas to reduce the effect of distant items in chapter 6.

Rec-Denoiser [8] is a plugin for the transformer models that learns a binary but differentiable attention masking mechanism. As such, the model can effectively set attention weights close to zero to actual zero values, effectively denoising the attention scores. Similarly, [54] STRec only keeps the TOP-L attention scores for a given L for both accuracy and memory gains. This is similar to Locker [35] in the sense that it reduces the amount of items included in the attention, but Locker focuses attention on items in the proximity of the timestep, whereas STRec focuses attention on the most-similar items.

In another line of work, CORE [41] adapts the transformer architecture to output a sequence of weights, after which the session embedding is the weighted average of the item embeddings. This is in contrast with SASRec, which produces the session embedding from the transformer architecture directly.

Similarly, ElecRec [10] also outputs weights, but in their application the weights represent the model's confidence that the item was part of the sequence. ElecRec is trained with a combination of the original sessions and sessions where some items have been replaced with a random item. The TOP-K recommendations can be computed by taking the items with the highest confidence in the last position of the sequence.

To model the uncertainty of the relevance of an item in a session, DT4SR (Distribution-based Transformer for Sequential Recommendation) [22] learns elliptical Gaussian distributions as item embeddings. This entails learning two separate embeddings per item, namely the mean and covariance embedding, which are then processed by two separate transformers. The session embedding of the mean and covariance transformer are then used to find recommendations with 2-Wasserstein distance [3]. Fan et al. [23] improve upon DT4SR with STOSA (Stochastic Self-attention), which introduces a Wasserstein self-attention layer. This mechanism is more appropriate for calculating the self-attention as it satisfies the triangle inequality [13][4].

Training and loss improvements RSS (Recency-based Sampling of Sequences) [72] improves the accuracy and training time of GRU4Rec and SASRec by devising a new training task. Instead of letting the model predict the item at position t given the items at positions $1 \dots t-1$, RSS extracts items from the input sequence proportional to their position. Hence, more recent items are more likely to be extracted from the sequence. The new task is then to predict the items that were withheld. Besides Binary Cross-Entropy, Petrov and Macdonald [72] experiment with LambdaRank [6] which exhibits better performance for some datasets.

In parallel with our work in chapter 4, two very recent publications [49, 73] independently discovered that the single negative sample in SASRec is not sufficient for training, and that SASRec can attain similar or even better accuracy than BERT4Rec using the Softmax loss function without negative sampling. We will reconfirm this finding in chapter 4. Since cross-entropy over all the items can become a scability bottleneck for large datasets, Petrov and Macdonald [73] additionally deduce a variant of BCE that takes the number of negative samples into account, significantly improving both accuracy and training time.

ASReP (Augmenting Sequential Recommendation with Pseudo-prior items) [62] notes that the model performance improves as the length of a session increases, and hence proposes to enhance the input sequences with *pseudo prior* items. This is done by first training SASRec in the reverse direction to augment sequences with items

[4]Intuitively, the advantage of a similarity measure that satisfies the triangle inequality is that if embedding e_1 is considered similar to e_2, and e_2 is considered similar to e_3, then e_1 is also considered similar to e_3 by the model. This typically enhances convergence.

that would likely precede the sequence, after which the pre-trained SASRec and the augmented sequences are used in the conventional training phase of SASRec. We revisit the motivation for ASReP in chapter 5, where we find that on some datasets the session length is not exclusively correlated with performance of transformer models. Instead, we will find that longer sessions are easier to predict for in general.

Contrastive learning A prominent stream of work is on the application of contrastive learning (CL) to the transformer-based sequential recommenders [74, 18, 99, 108, 63, 68, 102]. Contrastive learning is a self-supervised machine learning technique that aims to pull the representation of positive pairs closer, and push the representation of negative pairs further away [30]. For the application of CL in a particular domain and model, the definitions of representation, positive pairs, negative pairs and the push/pull mechanism need to be defined. In the case of transformers, the representations always regard the sequence embedding of the last layer S_L. Pairs are considered to be pairs of input sessions. The novelties in [74, 18, 99, 108, 63, 68] constitute innovations in the construction of positive and negative pairs, or innovative loss functions to pull or push the sequence embeddings closer or further away. Having a secondary signal to learn the embeddings besides the standard training objective should alleviate overfitting and make the models more effective on sparse datasets [74].

Models that build upon SASRec using CL typically duplicate each sequence s in the mini-batch and apply several non-deterministic augmentation techniques to the two duplicates, resulting in s' and s'' [99, 63]. The two, now slightly different, duplicates are considered to be a positive pair, whereas all other pairs with either s' or s'' are considered to be a negative pair. The specific augmentation techniques consist of random cropping[5], item masking, random reordering [99], and additionally item substitution or insertion [63]. Qiu et al. [74] have a slightly different approach, where for each sequence in the batch they sample a sequence with the same last item to form positive pairs. CBiT [18] is the first out of the aforementioned CL approaches that uses BERT4Rec as the base model, and leverages the randomly-masked sessions from the Cloze objective as positive pairs. In [18, 74], additional dropout stimulates different representations of positive pairs, which are then drawn together. In all cases, the contrastive loss (added to the standard training loss) promotes similarity in the sequence embeddings of the positive pairs relative to the negative pairs [99, 63, 74, 18].

Ma et al. [68] create positive pairs by splitting sequences on a particular position into a preceding and a succeeding subsequence. They propose a contrastive loss to compare the sequence embeddings according to their similarity in *intention*. The intentions behind a subsequence are computed by mapping the sequence embeddings S_L to K-dimensional vectors, where K represents the number of intentions.

In this thesis we will not consider any possible improvements with contrastive learning and will instead focus on simplifying the architecture of the sequential recommendation models.

2.4 Reproducibility issues

Most of the publications in subsection 2.3.6 claim superiority over their baselines and the GRU4Rec, SASRec and BERT4Rec models. However, recent studies [71, 64, 37, 38, 84, 24] have shown that there exist widespread flaws in the evaluation of new

[5]Random cropping refers to extracting a contiguous subsequence of the session.

recommendation models and question to what degree these flaws have affected the field's notion of the current state-of-the-art. We continue this section by discussing the publications on reproducibility specific to the sequential and session recommendation domain.

Petrov and Macdonald [71] observe that BERT4Rec's superiority over SASRec in accuracy is not consistently shown throughout the aforementioned studies. BERT4Rec was superior in 85 out of 134 comparisons, it was a tie in 16 comparisons, and SASRec was superior in 32 out of 134 comparisons. Though some inconsistencies may be explained by differences in datasets, there are also inconsistencies in comparisons where the dataset is exactly the same. Petrov and Macdonald [71] attribute this to poor default configurations of the open-source implementation of BERT4Rec. They then properly tune BERT4Rec and show similar or superior performance compared to the models introduced in [74, 15, 21, 60]. Therefore, it is unclear whether the increased performance of the model variations presented in subsection 2.3.6 actually substantiate. Similar conclusions on BERT4Rec's performance were drawn in [27]. For GRU4Rec, Hidasi and Czapp [37] found that its poor performance in following works is likely to be the result of a combination of poor hyperparameter tuning and faulty third-party implementations. Lastly, Ludewig and Jannach [64] show that simpler session-based models like SKNN often reach similar or superior performance compared to neural approaches depending on the dataset, and should therefore always be included as a baseline in order to ensure progress.

Chapter 3

Preliminaries

In this chapter, we introduce the preliminaries required for evaluating and tuning our models. We will introduce the datasets used in section 3.1 and their train-validation-test splits in section 3.2. We then detail our evaluation setup in section 3.3 and finally our hyperparameter optimization process in section 3.4.

3.1 Datasets

In this section, we introduce and discuss the datasets that will be used throughout this thesis. We will discuss the open datasets in subsection 3.1.1 and the proprietary Delivery Hero datasets in subsection 3.1.2. Finally, we present several dataset statistics in subsection 3.1.3.

3.1.1 Open datasets

We adopt two sequential recommendation datasets often used in the literature, namely Amazon Beauty [69] and MovieLens-1M [31]. Amazon Beauty is a dataset consisting of reviews on the Amazon Beauty catalogue. It has been processed so that the presence of a review is considered an interaction. All interactions by a single reviewer constitute a single session [69]. Similarly, MovieLens-1M is a dataset consisting of ratings by users on the MovieLens platform, where each rating is considered an interaction. All interactions of a single user constitute a single session [31]. Besides their widespread adoption, the Beauty and ML-1M dataset have very different dataset characteristics, which will allow us to provide a more comprehensive view on the performance and behaviour of the models. We will further discuss these characteristics in subsection 3.1.3.

Preprocessing We preprocess the Beauty and ML-1M dataset exactly as [87], namely with a 5-filter. This constitutes removing all users and items with less than five interactions. This is the most common preprocessing technique for the two datasets, and widely adopted BERT4Rec, SASRec and their variants [87, 47, 23, 62, 18].

Discussion We note that these datasets have shortcomings. Most importantly, the sessions in both datasets virtually span an unlimited time period. For example, the longest time span of a session in our processed Beauty dataset is almost 14 years. We argue that the relatedness of two consecutive interactions decreases with their time interval, so that these datasets may be limited in indicating a model's ability to exploit short-term context for recommendation. Still, both of these datasets can be considered the most popular benchmarks for evaluating sequential and session-based recommender systems [47, 73, 87, 23, 18, 62]. We therefore choose to include these datasets to allow for reproducibility and comparison with previous works.

3.1.2 Proprietary datasets

From Delivery Hero we use two datasets from different countries, appropriately named DH Greece and DH Singapore. The sessions in these datasets represent shopping carts from their Q-Commerce platform. The items in the session represent the purchased products, and the ordering of the session is the ordering in which the products were added to the shopping cart.

Preprocessing We do not process these datasets in order to represent reality as much as possible.

Timeframe In contrast to the open datasets, we can select the amount of data ourselves. For DH Greece and DH Singapore we take 3 months and 1 month of data respectively. These timeframes were found by evaluating our models on increasingly larger timeframes until all models' performance stabilized. We do not choose longer timeframes because it does not improve performance, most likely because of seasonality effects. For example, a seasonality effect could be that ice cream is much more often bought in the summer. Hence training data from the winter, where ice cream is bought less often, is less relevant and might even reduce a model's accuracy. On the other hand, our process for choosing the timeframes itself is not utterly robust, since larger models (than the ones we used to choose the timeframe) might benefit from even more data in order to converge. In an ideal case we would include the timeframe as a hyperparameter to model training, but this added too much overhead in terms of implementation and hyperparameter tuning. However, we believe the models' performance on the chosen timeframes to be very close to optimal.

3.1.3 Data statistics

We summarize the data statistics of the four datasets in Table 3.1 below. Furthermore, we visualize the session length distribution in Figure 3.1 and the item popularity distribution in Figure 3.2.

Dataset	# Sessions	# Items	# Interactions	Avg. length	Density
Beauty	40.226	54.542	353.962	8.8	0.016%
ML-1M	6.040	3.416	999.611	165.5	4.845%
DH GR	387.270	6.872	2.061.278	5.3	0.077%
DH SI	258.710	38.246	1.474.658	5.7	0.015%

TABLE 3.1: Statistics of our dataset suite.

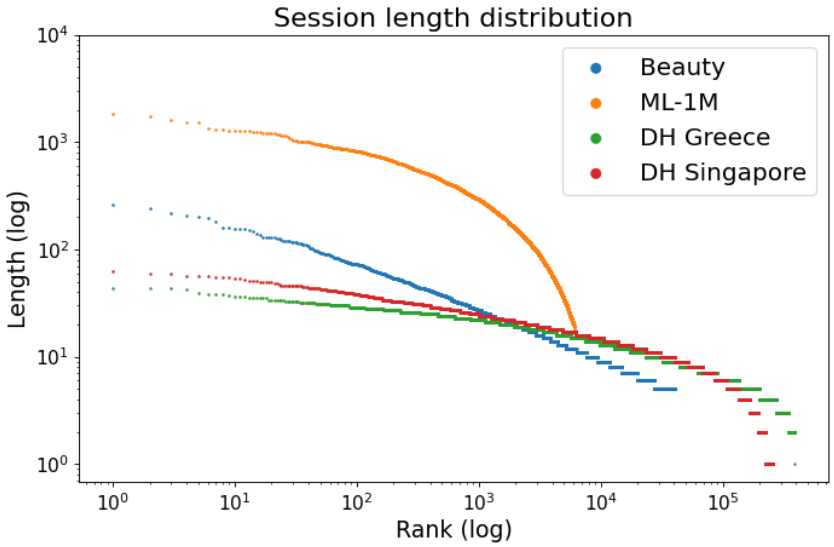

FIGURE 3.1: Distribution of session lengths of our dataset suite.

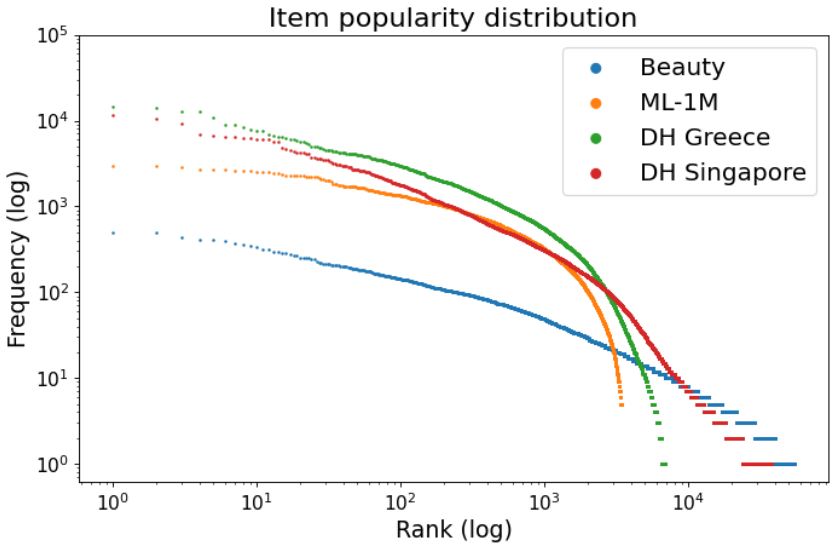

FIGURE 3.2: Distribution of item popularity of our dataset suite.

We selected the Beauty and ML-1M dataset because of the strong differences in dataset characteristics. The Beauty dataset is characterized by its low density and short sequences. In contrast, the ML-1M is characterized by a high density and long sequences. By adopting both in our dataset suite, we hope to gain and provide a comprehensive view of model recommendation performance and prevent ourselves from tailoring experimental models to specific dataset characteristics.

Our proprietary datasets are generally short. The DH Greece dataset is denser than the DH Singapore dataset because of improved product standardization throughout Q-Commerce vendors. This is also reflected in the item popularity distribution, where DH Singapore shows a much longer tail of low-popularity items. Unsurprisingly, the session length distributions of the proprietary datasets are roughly the same because for both datasets, sessions represent Q-Commerce shopping carts.

Throughout this thesis we will see how the subtle differences in the characteristics of the two proprietary datasets can affect recommendation performance significantly.

3.2 Train-validation-test split

In order to evaluate our models, we need to separate our datasets into (a) validation set(s) and a test set. Both of these sets contain subsets of sessions for training and evaluating models. We train and evaluate models on the validation set(s) for finding an optimal hyperparameter configuration. We train and evaluate models on the test set for reporting model performance given the optimal configuration found with the validation set. The way the validation set(s) and test set are constructed differs between the open and DH datasets. For the the open datasets, our main goal is to facilitate reproduction. For the DH datasets, our main is to represent reality as much as possible.

3.2.1 Open datasets

For Beauty [69] and ML-1M [31], we follow [87] and take the last item of each session as the test ground-truth item, and the second-to-last item of each session as the validation ground-truth item. Hence, in the validation phase, we train the model on all sessions with their last two items withheld, and evaluate them by predicting the second-to-last item with the preceding items as input. Then, during the testing phase, we train the model on all sessions with the last item withheld, and evaluate them by making them predict the last item with the preceding items as input. We visualize the approach below.

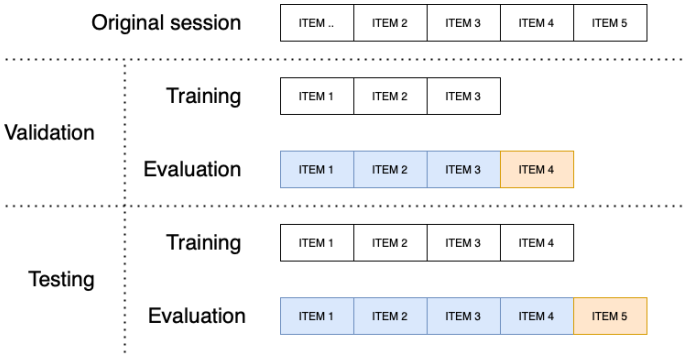

FIGURE 3.3: Train-validation-test split for open datasets. For evaluation in the testing and validation phase, we highlight the input session with blue and the ground-truth next-item with orange.

Discussion A shortcoming of this train-validation-test split is that there is a discrepancy between the average lengths of the sessions between the validation and testing phase. If the original sessions are on average length X, then the sessions on which the models are trained during the validation phase are on average length X - 2. As a result, the hyperparameter configuration that we find to be optimal for sessions of average length X - 2 might not be optimal for sessions of average length X. Furthermore, the train-validation-test split only results in one validation split. As a result, we will not be able to evaluate the variance of a model's performance before testing. On the other hand, the split is deterministic which facilitates reproduction. This is important because there have been some cases where non-deterministic splits

seem manufactured by hand, or at least extremely unlikely [24]. The split is also the most data efficient out of all conventional train-validation-test splits, since we use every available session during evaluation.

3.2.2 Delivery Hero datasets

Next, we process the sessions of the DH datasets into multiple validation sets and a test set. For both DH datasets, we perform a global timesplit to split the sessions for training/validation and testing. More specifically, for the DH Greece dataset we withhold the last two weeks of sessions for evaluation (roughly 50.000 sessions), and use all preceding sessions for training. Similarly, for the DH Singapore dataset, we withhold the last week of sessions for evaluation (also roughly 50K sessions). This is the best approximation of a model's performance in production, as the model is trained on sessions up to a certain point in time, and subsequently deployed without any retraining.

For validation, we use cross-validation with three validation folds, meaning we create three pairs of (train, evaluation) sets of sessions. We do not make use of temporal information here, and simply create the folds with random sampling. The reasoning behind not using a temporal split for our validation phase is because it is not possible to create more than one mutually-exclusive fold with a temporal split. We want multiple validation folds in order to increase the robustness of our results.

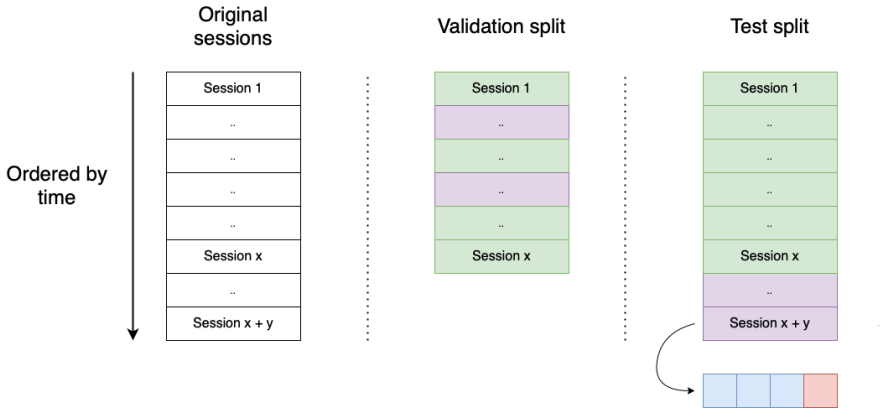

FIGURE 3.4: Train-validation-test split for DH datasets. We highlight the sessions for training with green and the sessions for evaluation with purple. The main difference with Figure 3.3 is that sessions are not shared between the validation and test set. We feed the model with all but the last item of an evaluation session (highlighted in blue), and make it predict the withheld last item (highlighted in orange).

3.3 Evaluation setup

In this section we introduce our approach to evaluating the models' performance. We discuss the widespread but flawed *sampled metrics* evaluation task and its methodologically correct alternative of *full ranking* in subsection 3.3.1. We then continue with the accuracy metrics in subsection 3.3.2, and finally discuss the *beyond-accuracy* metrics in subsection 3.3.3.

3.3.1 Evaluation task

On sampled metrics We divert from the task setting in [87] due to the criticism on *sampled metrics* in recent publications [61, 52, 15, 38, 7]. In short, sampling metrics refers to an alternative way of evaluating models by making each model rank 101 items from the item catalogue. The first 100 are items sampled uniformly or proportional to their popularity (excluding items that are already in the session). The 101-th item is the ground-truth item. The "TOP-K" items are then considered to be the TOP-K items of this ranked list. This process was originally designed to speed up evaluation as it does not require sorting the item scores of all items in the catalogue, and can instead be computed in constant time independent of the number of items. This evaluation task has been adopted extensively in various works, including BERT4Rec [87, 71], SASRec [47], several suggested improvements for BERT4Rec and SASRec [18, 63, 74, 22], and other influential works on recommendation [34, 91, 20, 50]. Publications criticizing this approach show that sampling metrics could lead to arbitrary model leaderboards, especially for target sets containing just 101 items [61, 52, 15, 38, 7]. To foreshadow, we took the original SASRec implementation and trained it on the Beauty dataset. Whereas SASRec outperformed all models in [47] , we find that the original implementation barely beats the popular baseline on the *full ranking* recommendation task.

Full ranking Instead, we evaluate our models by ranking all items in the catalogue, excluding those already in the session. This is inherently the most consistent method for evaluating models [52]. In addition, it is a proper representation of recommendation performance in production, as models recommend the TOP-K items from the whole item catalogue.

3.3.2 Ranking metrics

For each evaluation session, the model produces a TOP-K recommendation slate. There are several metrics available to evaluate the quality of the recommendation slate, and by extension the model that produced the recommendation slates. The main accuracy metrics are Normalized Discounted Cumulative Gain (NDCG), Mean Reciprocal Rank (MRR) and HitRate (HR) [64]. For a given TOP-K recommendation slate and a ground-truth item, all metrics evaluate to zero if the ground-truth item is not included in the recommendation slate. Otherwise, the metrics are calculated with the formula below, given the relative rank x of the ground-truth item in the TOP-K recommendation slate. The top-ranked item gets rank 1, the lowest-ranked item gets rank K.

$$\begin{aligned} \text{HR(x)} &= 1 \\ \text{MRR(x)} &= 1/x \\ \text{NDCG(x)} &= 1/\log_2(x+1) \end{aligned} \tag{3.1}$$

Note that these formulas are simplified variants from their conventional formulas [46]. The simplification is possible because there is only one ground-truth, and the graded relevance score of an item is simply 0 (not ground-truth) or 1 (the ground-truth). We visualize the formulas in Figure 3.5.

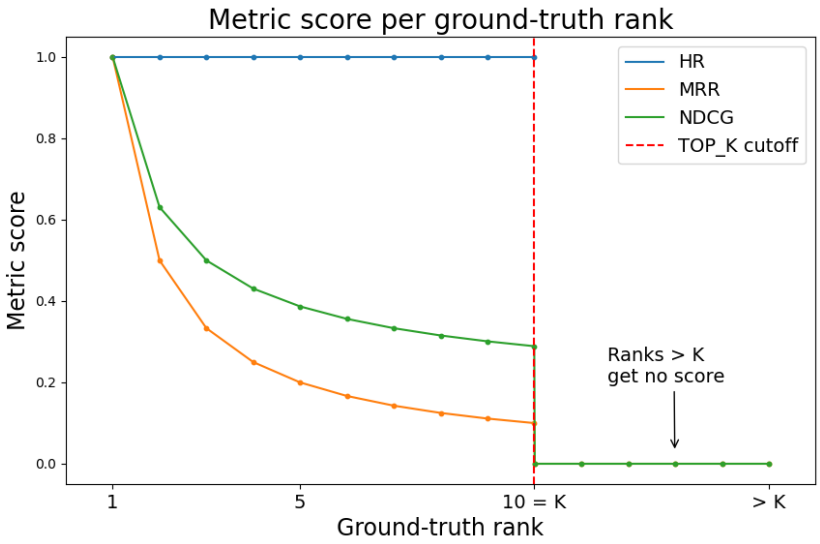

FIGURE 3.5: Metric score against ground-truth rank. In this figure, $K = 10$, meaning we only consider the TOP-10 recommendations from a model. If the ground-truth has a higher rank than K, a score is considered zero.

In this work we will be presenting the HR and NDCG ranking metrics with $K = 10$. The HR and NDCG metric together provide a view of the models' ability to retrieve (recall) relevant items and their ability to rank these relevant items respectively. We prefer NDCG over MRR because the tail of the curve of NDCG is less steep. Intuitively, we assume that if a user will scroll through the recommendations (beside the first few that are shown by default), the user will view all. Hence we want the scores of the lower ranks to be roughly equal. This intuition is best reflected by the NDCG metric. The value of 10 is chosen for K because it is both conventional in recommendation literature, and it is the maximum amount of items shown on the checkout pages of Delivery Hero.

3.3.3 Beyond-accuracy metrics

Besides the ranking metrics, we consider the following *beyond-accuracy metrics* to obtain a more comprehensive picture of the performance and behaviour of the models and their configurations: catalog coverage, serendipity, and novelty. Catalog coverage represents the fraction of catalog items that appeared in at least one TOP-K recommendation slate [45]. Serendipity measures the average number of correct recommendations for each user that are not recommended by the popularity baseline [28]. Novelty computes the negative log of the relative item popularity [109]. Formally, denoting a recommendation slate with r and an item i's popularity with $p(i)$, novelty is computed using Equation 3.2.

$$\text{Novelty}(r) = \sum_{i \in r} -\log\left(\frac{p(i)}{\sum_{j \in \mathcal{I}} p(j)}\right) \tag{3.2}$$

3.4 Hyperparameter optimization

To properly evaluate and compare our models' performance, we have to *hypersearch* (optimize) the hyperparameters. To do this, we employ Optuna [2] for its flexibility in defining search spaces and built-in visualizations. We optimize the average NDCG@10 on the validation folds[1]. For the sake of completeness, we discuss the optimizing process in detail in subsection 3.4.1 and our approach to pruning in subsection 3.4.2.

3.4.1 Optimizing process

We navigate the search space by first sampling 20 random hyperparameter configurations from the search space and evaluating their performance (in hyperparameter-search terminology, the process of evaluating a single hyperparameter configuration is called a *trial*). We then continue with 70 configurations from a TPE sampler [5][2]. The TPE sampler is a data-efficient sampler available in Optuna that supports multivariate optimization. Hence, it is able to take the relations of variables into account. For example, the optimal size of a model is dependent on the optimal amount of regularization and vice versa, and this can only be exploited if multivariate optimization is applied. In short, the TPE sampler uses Gaussian Mixture Models to model the parameter configurations that resulted in a high objective value in $g(x)$, and similarly to model parameter configurations that resulted in a low objective value $l(x)$. It then proposes the configuration that maximizes $g(x)/l(x)$. We precede the TPE sampling with random trials in order to prevent premature convergence to a local optimum.

We use average NDCG@10 across the folds as the objective value, and consider the hyperparameter configuration with the highest average NDCG@10 as the optimal model configuration after the maximum number of trials has been reached. We will then use this optimal configuration to evaluate the model on the test set.

3.4.2 Pruning trails

To save time we make use of the pruning functionality of Optuna [2]. *Pruning* refers to stopping unpromising trials early-on before evaluating the model on all folds. From Figure 3.6 it is evident that the performance on the first fold is already a reasonable indicator of final test performance, so that it is safe to prune the worst-performing trials. Hence, after each fold we check whether the average NDCG@20 so far is not in the bottom 20% of trials. If it is, we prune the trial. Of course, we have multiple folds to increase the robustness of our hypersearch, but we have found that the performance of a model configuration after a single fold is very correlated with the final test performance. Hence, if a trial is in the bottom 20% of trials in terms of average NDCG@10 so far, we can already conclude that it will most likely not be the top-performing configuration. Note that our academic datasets only have one fold, so it is not possible to do any pruning there.

[1] Some configurations were actually found by optimizing NDCG@20. Given that NDCG@20 and NDCG@10 are extremely correlated, we decided to not re-run these hypersearches because of the costs of hypersearching parameters.

[2] We also stop a hypersearch if the objective value has not sufficiently increased for 20 trials.

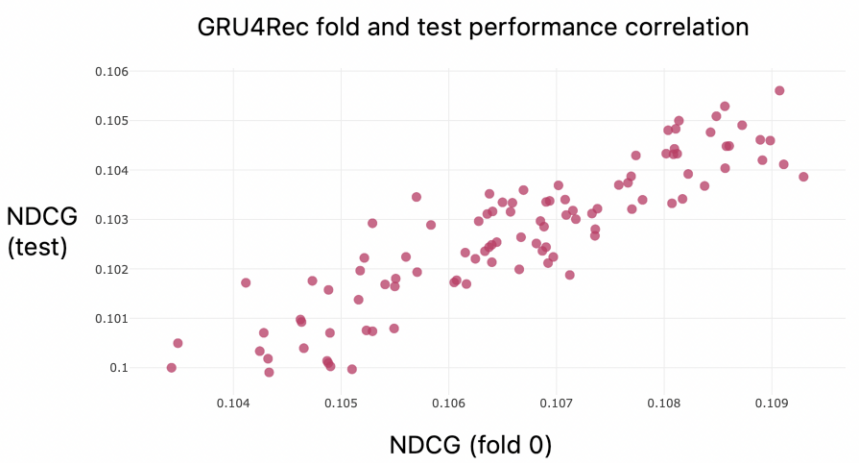

FIGURE 3.6: The correlation between GRU4Rec's fold and test performance on the DH Greece dataset. It shows that the performance on the first fold is already a reasonable indicator of final test performance, so that it is safe to prune the worst-performing trials. In addition, the variance indicates that a single fold is not sufficient to find the optimal configuration. Figure created with Optuna [2].

Chapter 4

Model comparison

In this chapter, we will evaluate the models on our dataset suite. The main goal of this chapter is to produce a fair comparison between models by (1) addressing the methodological errors in implementation and hyperparameter optimization, and (2) standardizing implementation between the different models in order to isolate the benefit of each architecture. We will first discuss our baselines in section 4.1 and the hyperparameter search space in section 4.2. With the hyperparameter optimization process fully specified, we introduce the implementation details and make some preliminary design decisions in section 4.3. Finally, we present the evaluation results in section 4.4.

4.1 Baselines

Besides SKNN [44, 64], we include three other non-neural baselines, namely MostPopular, ItemKNN [81] and NextPopular [83], which we briefly discuss below.

4.1.1 MostPopular

MostPopular always recommends the TOP-K most popular items in order of their popularity. For the sake of clarity, the most popular items are the items with the most interactions in the training data.

4.1.2 ItemKNN

ItemKNN [81, 95] disregards the ordering of the items in the session, and instead uses item-item similarities to compute recommendations. It returns the items with the highest scores defined by Equation 4.1.

$$\text{score}(i, s) = \sum_{j \in s} \frac{r_i r_j}{\|r_i\| \, \|r_j\|} \tag{4.1}$$

Here, the vector $r_i \in \{0, 1\}^{|\mathcal{S}|}$ for an item i is a multi-hot vector indicating the sessions that contain item i. Intuitively speaking, ItemKNN recommends the items that appeared most often with the items in the session. To clarify, ItemKNN uses item-item similarities to find recommendations, whereas SKNN [44, 64] uses session-session similarities to find recommendations.

4.1.3 NextPopular

Our last baseline is NextPopular, a simplification of the Markov Decision Process (MDP) defined in [83]. NextPopular recommends the TOP-K items that have most often been observed as direct neighbours of the last item of the session. Hence,

for each item i, NextPopular counts how often item j occurs as a direct neighbour (meaning it either directly precedes or succeeds in a session) of item i in a session. At prediction time, NextPopular takes the last item of a session and recommends the items that most often neighboured the last item, in the order of the counts. NextPopular will be vital to our analyses in chapter 5 and chapter 6.

4.2 Hyperparameters and search space

We now define the hyperparameters and the search space for BERT4Rec [87], SASRec [47], GRU4Rec [40, 39] and SKNN [64, 44]. All hyperparameters are invariant to the design decisions we will make later in section 4.3.

4.2.1 Limitations of original publications

We note that the search space of the original publications is limited. In [87, 47], BERT4Rec and SASRec's optimal configuration are found through a grid search for various values of the embedding dimension, weight decay and dropout rate. However, hyperparameters like the number of transformer layers, the number of heads, learning rate and the batch size are fixed. After the optimization, the authors vary the number of transformer layers in an ablation study. In [40], GRU4Rec is optimized by fixing the hidden dimension of the model and optimizing the auxiliary hyperparameters[1]. The optimal auxiliary hyperparameters are then used to search the hidden dimension. Though methodologically sound, all three approaches to optimization are biased towards the values of the fixed hyperparameters, namely the number of transformer layers and the hidden dimension. To illustrate, we found that there is a strong interplay between optimal regularization and size of the model. By optimizing regularization with a fixed model size, the optimal model size with the found hyperparameter values for regularization will be biased towards the previously fixed values for model size.

Instead, we rely on the TPE sampler to efficiently sample from all parameters at the same time, which will result in an unbiased optimal configuration of hyperparameters.

4.2.2 Neural model hyperparameters and search space

The precise search space for each model is defined in Table 4.1.

[1] It was not specified what hyperparameters are actually being optimized in [40].

Hyperparameter	BERT4Rec	SASRec	GRU4Rec
Learning rate	1e-4 - 1e-2	1e-4 - 1e-2	1e-4 - 1e-2
Weight decay	1e-5 - 1e-1	1e-5 - 1e-1	1e-5 - 1e-1
Batch size	32 - 512	32 - 512	32 - 512
Embedding dim. (e)	16 - 512	16 - 512	16 - 512
Drop rate	0 - 0.9	0 - 0.9	0 - 0.9
# transformer layers (L)	1 - 4	1 - 4	
# heads (h)	1 - 4	1 - 4	
Transformer layout	{FDRN, NFDR}	{FDRN, NFDR}	
Masking probability	0.05 - 0.9		
Hidden dim			16 - 512
N	95%	95%	95%
# random train cases	10		

TABLE 4.1: Hyperparameters and their search space for the neural models. A missing entry indicates that the hyperparameter does not apply to the model. A single value indicates that the hyperparameter is fixed to the given value.

Shared hyperparameters In short, we have the learning rate, weight decay, batch size, embedding dimension and dropout rate as shared hyperparameters for all three models. The dropout rate is applied to the embedding layer and the output of each transformer/GRU layer.

Transformer hyperparameters The transformer models have additional hyperparameters for the number of transformer layers (L), number of heads (h) and transformer layout. The transformer layout indicates what the precise ordering is of the components inside the transformer layout (See subsection 2.3.4). The original SASRec implementation [47] uses NFDR (Normalization, Function, Dropout, Residual), meaning it first applies the layer normalization, then the function (MHA or FFN), then the dropout and then the residual connection. The original BERT4Rec implementation [87] uses FDRN (Function, Dropout, Residual, Normalization). Instead of fixing to a single layout, we let the hypersearch decide the layout. Also, we note that we do allow for multiple heads in SASRec as opposed to the original publication [47], as multi-head attention generalizes single-head attention anyway.

Masking probability Compared to SASRec, BERT4Rec has one additional hyperparameter, namely the masking probability. This is the probability that an item is masked in the generation of the randomly-masked input sessions (See subsection 2.3.5). We have preliminarily experimented with various values for the number of random train cases (the randomly-masked input sessions), and found that performance stabilized for all datasets for all configurations before 10 random train cases. To limit the search space and training time of BERT4Rec, we therefore fixed the parameter to a value of 10.

Hidden dimension GRU4Rec has an additional hidden dimension hyperparameter that defines the dimension of the GRU layer. This is the dimension of the weights and the output h_t of the GRU layer.

Length of sessions (N) All three models use the whole-session batches, and we fix the length of the sessions to the 95% percentile of the session length distribution. This means that roughly 5% of sessions are truncated. For Beauty, the 95% percentile is 20 items, for ML-1M 556 items, for DH Greece 11 items and for DH Singapore 15 items. We actually found that increasing the percentile did not improve performance, while it did increase training time significantly due to the outliers in session lengths. We assume this is because the longest sessions contain more noise, so that they are less informative for training purposes.

4.2.3 SKNN hyperparameters and search space

For completeness we also provide the SKNN hyperparameters and search space in Table 4.2. We note that we adopt the implementation provided by Delivery Hero Research.

Hyperparameter	SKNN
Sequential weighting	{true, false}
Sequential filter	{true, false}
Sample size	500 - 2000
Sampling	{random, recent}
Similarity measure	{dot, cosine}
IDF-weighting	{true, false}
k	50 - 500

TABLE 4.2: Hyperparameters and their search space for SKNN.

Sequential weighting defines whether the score for an overlapping item between sessions increases with the item's position in the session (S-SKNN, V-SKNN [64]). Sequential filter defines whether no items may be recommended that have not directly succeeded the last item of the session (SF-SKNN [64]). The sample size defines the number of sessions for which we calculate the similarity with the session at hand. These can be sampled uniformly at random, or sampled proportional to their recency [64]. The similarity measure defines whether we use dot-product or cosine similarity for the similarity computation. Parameter k defines the number of most-similar sessions from the sampled sessions that are used to compute the recommendations. IDF-weighting defines whether the weight of an item in the similarity computation should be proportional to its Inverse Document Frequency value [79]. In session recommendation, the frequency of an item is the amount of times it appears in a session in the training data.

4.3 Implementation details

There are differences between our implementation of BERT4Rec, SASRec and GRU4Rec and their original open-source implementations. Our motivation for these differences is two-fold. Firstly, we found that each model contains various details that are not necessarily dictated by their main architecture. This complexifies implementation [37] and prevents us from drawing solid conclusions on the architecture itself. Secondly, we found that seemingly-minor details can have significant effect on the performance. An example of this is that BERT4Rec claimed superiority over SASRec because of its bidirectionality [87], but we will find that it is actually the loss function that made BERT4Rec superior in the original evaluation. Therefore, we aim to

standardize the implementation of the models as much as possible to allow for an analysis on their main architecture, namely the bidirectional or unidirectional transformers and the GRU layer.

We prelude with standardizing the mechanism for training termination in subsection 4.3.1. We then discuss two major differences that positively impact the performance of our implementations, namely the loss function of SASRec and GRU4Rec, and the batching mechanism of GRU4Rec in subsection 4.3.2 and subsection 4.3.3 respectively. Furthermore, we will discuss more minor implementation details in subsection 4.3.4.

4.3.1 Training termination

The approach to determining when to stop training of BERT4Rec, SASRec and GRU4Rec in their respective publications are either erroneous or arguably, wasteful. In [40, 39], the number of epochs for GRU4Rec is simply treated as a hyperparameter. Though this is methodologically sound, it disregards the fact that model performance can be measured across epochs in a single run. By ignoring this, GRU4Rec might be trained and evaluated multiple times with the same model configuration but with a different number of epochs, whereas the model performance across different epochs could have been extracted from a single run. In BERT4Rec and SASRec [87, 47], this is erroneously addressed by evaluating the model on the test set every single epoch. Given the random oscillations in the evaluation metrics throughout epochs [2], this direct involvement of the test set in the training phase allows for opportunistically choosing coincidentally-high evaluation metrics [38, 53]. This effect is further amplified by the randomness of the sampled metrics approach [53].

Early stopping Instead, we choose to withhold a subset of sessions from the training data and validate the model performance on this *early-stopping-validation* set every epoch. Hence, we sacrifice 10% of the training sessions to solve the issue of having to choose the number of epochs. We measure our target metric NDCG@10 on the early-stopping-validation set and stop training if this metric has not increased for a fixed number of epochs, the *patience*. When we terminate training, we restore the model weights of the epoch with the highest NDCG@10 on the early-stopping-validation set. By measuring a metric on the data from our training set, we do not use any information from the test set, making it methodologically sound. Furthermore, early stopping allows us to remove the number of epochs as a hyperparameter, making our model training and hyperparameter-search process more efficient. We do not hypersearch the patience hyperparameter, and instead fix it to the value of 5. From preliminary experiments we found that higher values did not result in better evaluation metrics on any of the datasets. Lower values for the patience resulted in premature stopping.

Reducing the learning rate Now that we can estimate model performance with the early-stopping-NDCG@10, we can reduce the learning rate when the performance plateaus [3]. Reducing the learning rate has often been shown to enable convergence

[2] Random oscillations in performance throughout epochs are inherently caused by the random batching process.

[3] We have found that the loss is often still decreasing while the early-stopping-NDCG@10 has plateaued, indicating that the loss is not a suitable estimator for deciding when to reduce the learning rate. We assume the absence of a good performance estimator is why the original implementations did not make use of this mechanism.

to a better minimum [17]. We reduce the learning rate by a factor of 10 when early-stopping-NDCG@10 has not improved for 4 epochs. Admittedly, while the main goal of this thesis is to simplify models instead of adding even more components, we found this well-known, easily-implementable [4] mechanism to directly improve performance. For example, Figure 4.1 shows the performance trajectory of GRU4Rec on the DH Greece dataset. Evidently, lowering the learning rate on plateau increases performance. An initial learning rate of 1.9e-4 resulted in worse performance demonstrating that GRU4Rec benefits from a high initial learning rate to escape bad local minima from the random initialization, but requires a lower learning rate to fully converge.

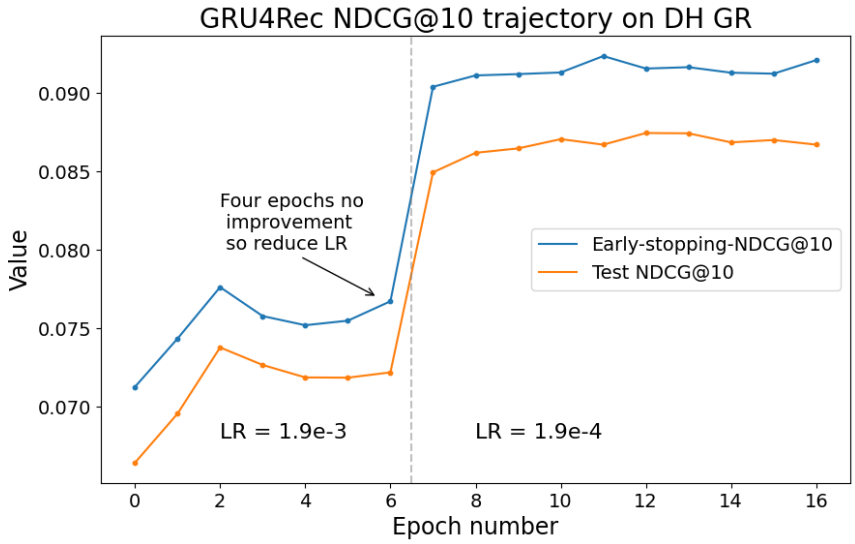

FIGURE 4.1: Performance trajectory of GRU4Rec on the DH Greece dataset. It illustrates how lowering the learning rate on plateau increases performance. Since early-stopping-validation and test sessions are drawn from different distributions in terms of time, the early-stopping-validation NDCG is slightly higher. This also demonstrates why choosing longer timeframes for the DH datasets does not necessarily result in higher performance.

4.3.2 Loss functions (SASRec, GRU4Rec)

We recall from subsection 2.3.4 that SASRec and GRU4Rec are trained with binary cross-entropy (BCE) using negative sampling, which entails including the positive sample (the ground-truth item) and a number of negative samples sampled uniformly at random. Instead, we propose to adopt the loss function of BERT4Rec, which is the categorical cross-entropy loss (Softmax) loss, over all the items. This essentially means that we use all items except the ground-truth as negative samples. The original motivation for a fixed number of negative samples is that the loss computation time is independent of the number of items, and hence allows for training models on larger datasets [47]. In turn, training becomes less demanding compute and memory-wise, but may result in sub-optimal performance [73, 49]. However, we note that it is still feasible to compute the Softmax loss using all items on our suite of datasets. Thus, to fairly compare GRU4Rec and SASRec with BERT4Rec, we implement Softmax loss for these models as well.

[4] ReduceLROnPlateau can be implemented in 2 lines of code in TensorFlow 2.10.

SASRec We will first hypersearch both SASRec variants on the open datasets and summarize the results in Table 4.3.

Dataset	Model	HR	↓NDCG	CatCov	Ser	Nov
Beauty	SASRec (Softmax)	.039	.021	.106	.033	12.1
	SASRec (1 sample)	.012	.006	.002	.005	10.3
	Popular	.012	.005	.000	.001	9.7
ML-1M	SASRec (Softmax)	.307	.176	.677	.300	10.7
	SASRec (1 sample)	.210	.110	.660	.205	10.8
	Popular	.016	.008	.003	.000	8.5

TABLE 4.3: Preliminary results on negative sampling with SASRec. We find that Softmax loss over all items performs considerably better. The table is sorted by NDCG@10.

Evidently, the Softmax loss on SASRec results in better performance. While SASRec with only 1 negative sample still significantly outperforms the popular baseline on ML-1M, the same model barely beats the popular baseline on Beauty. The low catalog coverage and novelty indicate that SASRec with a single negative sample degenerates to consistently recommending popular items. Surprisingly, two very recent parallel works to ours have made the same observation [73, 49]. The analysis in [73] shows that the single negative sample is not enough downward force on the output of the model, so that for every session there are much more than TOP-K items that receive a score very close to one. On a dataset like Beauty with a large item catalogue, this makes the model recommend predominantly popular items, which have a slightly higher score than all other highly-scored items. Petrov and Macdonald [73] call this behaviour *overconfidence*. SASRec suffers less from overconfidence on the ML-1M dataset because ML-1M has a smaller item catalogue than Beauty. Therefore, the items for which SASRec is usually overconfident are more often drawn to zero with the negative samples.

Furthermore, the extreme performance difference between Table 4.3 and the original SASRec evaluation [47] highlights the inadequacy of sampled metrics to evaluate a model. Whereas on the full ranking evaluation task SASRec barely surpasses the popular baseline, SASRec outperformed all its baselines on the sampled metrics evaluation task in [47]. Lastly, both Petrov and Macdonald [73] and Klenitskiy and Vasilev [49] state that BERT4Rec's original superiority over SASRec can be solely attributed to the Softmax loss. We will further discuss this statement in section 4.4.

GRU4Rec Though the loss function for GRU4Rec is not specified in [40, 39], Hidasi and Czapp [37] note that Softmax loss or BPR-max loss introduced in [39] clearly outperform all other loss functions on their datasets. Therefore, we hypersearch the GRU4Rec variants with the two different loss functions, and summarize the results in Table 4.4 below.

Dataset	Model	HR	↓NDCG	CatCov	Ser	Nov
Beauty	GRU4Rec (Cross)	.028	.015	.649	.028	14.6
	GRU4Rec (BPR-max)	.027	.015	.759	.027	15.8
	Popular	.012	.005	.000	.001	9.7
ML-1M	GRU4Rec (Cross)	.194	.100	.953	.190	11.9
	GRU4Rec (BPR-max)	.135	.067	.816	.133	12.5
	Popular	.016	.008	.003	.000	8.5

TABLE 4.4: Preliminary results on various loss functions for GRU4Rec. The table is sorted by NDCG@10.

Interestingly, the GRU4Rec version with cross-entropy clearly outperforms the BPR-Max version on ML-1M, but there is no notable performance difference on Beauty. Therefore, we will adopt the same Softmax loss we use for BERT4Rec and SASRec for the remainder of this thesis. Admittedly, our preliminary analysis is not sufficient to conclude that cross-entropy will result in higher performance on any dataset, but this standardization does allow us to isolate the effect of the architecture. We leave the exploration of various loss functions for future work.

4.3.3 Batch format (GRU4Rec)

Our GRU4Rec implementation differs from the original in its batch format. We recall from subsection 2.3.2 that the original GRU4Rec uses session-parallel mini-batches. Sessions are processed one-by-one, one item per batch, while maintaining an internal state throughout batches. The motivation behind this design decision is that session lengths may be broadly distributed. For example, one dataset used in [40] has session lengths ranging from 2 to a few hundred. However, from Figure 3.1, we can see that the session lengths of our datasets are mostly short, especially on the Delivery Hero datasets. Therefore, we implement GRU4Rec with whole-session batches similar to SASRec and BERT4Rec. To reiterate, this means that we process sessions to be length N, either by truncation or the addition of special padding items. Processing sessions as a whole greatly simplifies implementation. With whole-session batches, we do not have to maintain a hidden state across batches during training, and we can infer recommendations from a single batch during prediction. We hypersearch both GRU4Rec variants and summarize the results in Table 4.5.

Dataset	Model	HR	↓NDCG	CatCov	Ser	Nov
Beauty	GRU4Rec (w.s.)	.052	.029	.231	.048	12.9
	GRU4Rec (s.p.)	.028	.015	.649	.028	14.6
	Popular	.012	.005	.000	.001	9.7
ML-1M	GRU4Rec (w.s.)	.339	.203	.816	.331	10.7
	GRU4Rec (s.p.)	.194	.100	.953	.190	11.9
	Popular	.016	.008	.003	.000	8.5

TABLE 4.5: Preliminary results on the batch design for GRU4Rec. We use (w.s.) to refer to GRU4Rec with whole-session batches, whereas we use (s.p.) to refer to GRU4Rec with session-parallel batches. The table is sorted by NDCG@10.

Evidently, using whole-session batches produces much better results than session-parallel mini-batches. We have found very little literature on this finding. In [49],

Klenitskiy and Vasilev note that they implemented GRU4Rec by replacing the transformer layer of their SASRec implementation with a GRU layer. Since the transformers use whole-session batches, we believe this GRU4Rec implementation to be very close to ours. The evaluation results show that their GRU4Rec implementation is much more performant on the exact same datasets as many other publications that use GRU4Rec as a baseline [87, 23, 47]. However, no comments are made on these specific results. Moreover, a popular variation of GRU4Rec is NARM [57]. This model adds a bidirectional attention mechanism as a *global encoder* next to the unidirectional GRU layer. This attention mechanism necessitates the use of whole-session batches. Given our results and the results in [49], we believe that part of NARM's additional performance over GRU4Rec may be attributable to the batch design instead of the global encoder.

Conjectures We propose several conjectures on why whole-session batches generally perform better. First of all, when processing a session one-by-one, the hidden state becomes out-of-sync because the weights that produced this hidden state are updated at each timestep. This might harm the integrity of the information stored in the hidden state, similar to how federated learning may suffer from instability [58]. Furthermore, in whole-session batches, the feed-forward propagation is still unidirectional, but the gradients are bidirectional. We visualize this conjecture in Figure 4.2. To illustrate, suppose we have a session where the first part of the session is very informative on the second part of the session, with two random noisy items in the middle separating these parts. Upon arriving at the noisy items, the GRU4Rec (s.p.) variant will learn to reset the hidden state as much as possible, since the first part of the session is not at all informative to the recommendations for the noisy items. In contrast, the gradients of GRU4Rec (w.s.) have a full view on the session (since they are calculated for sessions as a whole), and will incorporate the fact that the hidden state is informative for the items after the noisy items. This should lessen the degree to which it will learn to reset the hidden state. As a result, we believe the whole-session batch design to be more robust against noise.

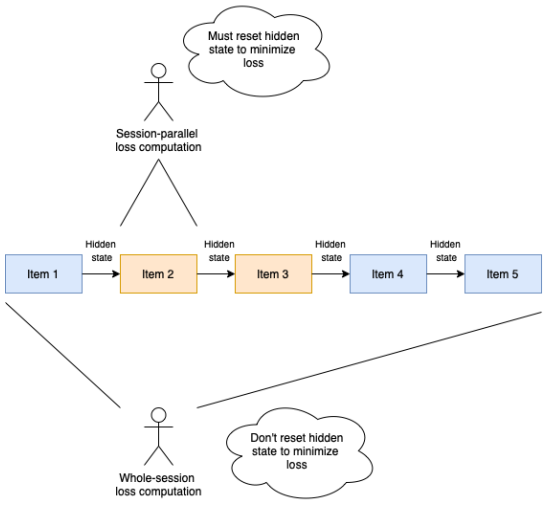

FIGURE 4.2: A visualization of one of our conjectures on why whole-session batches are better to train GRU4Rec. We highlighted "informative" items in blue, and highlighted "noisy" items in orange.

Lastly, the special padding items may act as a means to inform the model on the position of an item. In the case of whole-session batches, the hidden state can theoretically act as a counter for the number of special padding items until it reaches an actual item and subsequently leverage this information to make better recommendations. In the case of session-parallel batches, these special padding items do not exist and the hidden state is zero when the first item of a session is processed. In this thesis, we will refrain from exploring the validity of these conjectures, and instead focus on the analyzing the architectures.

4.3.4 Other details

Two minor differences between our implementation and the original publications are the prediction network and the optimizer. We briefly discuss both below.

Prediction network BERT4Rec's prediction network is slightly different from the one in SASRec. In BERT4Rec, the embeddings S^L produced by the last transformer layer are fed to a dense layer, then multiplied with the transpose of the item embedding matrix, and finally, the results are fed through a bias layer. In SASRec, S^L is immediately multiplied with the transpose of the item embedding matrix without any bias. GRU4Rec does not actually specify its prediction network, but requires at least one dense layer to project the sequence embedding of shape (N, e_{hidden}) to shape (N, e_{item}). Therefore, we adopt the prediction network of BERT4Rec for all three models for consistency.

Optimizer Both BERT4Rec and SASRec use the Adam optimizer [48]. GRU4Rec in [40] uses the AdaGrad optimizer [19]. We briefly experimented with various optimizers, and found Adam to consistently provide the best results. This is in line with conclusions drawn from other domains where deep learning is applied [85]. Besides the learning rate and weight decay, all other auxiliary hyperparameters of the Adam optimizer are fixed to their default value in TensorFlow.

4.4 Evaluation results

Finally, we can compare the model performance in their optimal configuration. We summarize the results in Table 4.6.

Dataset	Model	HR	↓NDCG	CatCov	Ser	Nov
Beauty	SKNN	.064	.037	.540	.058	12.6
	ItemKNN	.052	.030	.741	.048	13.9
	GRU4Rec	.054	.030	.231	.048	12.9
	BERT	.047	.026	.226	.043	13.2
	NextPop	.041	.026	.664	.039	13.5
	SASRec	.039	.021	.106	.033	12.1
	Popular	.012	.005	.000	.000	9.7
ML-1M	GRU4Rec	.339	.203	.816	.331	10.7
	SASRec	.307	.176	.677	.300	10.7
	BERT	.309	.171	.633	.303	10.6
	NextPop	.200	.114	.723	.194	10.5
	SKNN	.154	.077	.502	.148	9.8
	ItemKNN	.053	.026	.060	.038	8.7
	Popular	.016	.008	.003	.000	8.5
DH GR	GRU4Rec	.155	.090	.443	.128	10.1
	SASRec	.153	.090	.486	.128	10.3
	BERT	.150	.086	.375	.121	9.9
	NextPop	.140	.083	.796	.120	10.5
	SKNN	.131	.077	.785	.112	10.5
	ItemKNN	.117	.065	.697	.083	9.4
	Popular	.052	.024	.001	.000	7.5
DH SI	ItemKNN	.175	.100	.362	.136	10.0
	BERT	.171	.097	.125	.139	10.4
	SKNN	.159	.094	.361	.131	10.4
	GRU4Rec	.156	.089	.375	.129	11.0
	SASRec	.158	.088	.292	.128	10.7
	NextPop	.114	.067	.335	.089	10.6
	Popular	.049	.024	.000	.000	7.5

TABLE 4.6: The evaluation results of our vanilla neural models and the non-neural baselines, sorted by NDCG@10.

Non-neural vs. neural model performance Overall, we find that the neural models generally perform better on DH Greece and ML-1M, whereas the non-neural models perform better on DH Singapore and Beauty. We explain this with the density of the datasets. From Table 3.1, we find that DH Singapore and Beauty are relatively sparse with 0.015% density, whereas DH Greece and ML-1M are relatively dense with 0.077% and 4.8% density respectively. The sparsity of DH Singapore and Beauty seems to prevent our neural from outperforming the non-neural baselines. Recall that we chose the timeframe for the DH Singapore dataset by evaluating our models on increasingly larger timeframes until performance stabilized, implying that it is not a lack of data that is causing the inferiority of the neural models on this dataset. The superiority of the non-neural baselines on the sparse datasets also indicate that they can be considered competitive and should therefore always be included as baselines. For example, none of the published evaluation results on the Beauty dataset in [87, 47, 23, 18, 74, 99, 39] uses SKNN as a baseline, while it is the most performant by far.

Inconsistency in neural model ranking Furthermore, we do not find a consistent model ranking between the neural models, even on DH datasets. It demonstrates

that there is no single model that is consistently better than all baselines, as opposed to what is often suggested when a model is introduced [87, 47, 40, 23, 18, 74, 99]. This is to be expected, since each neural architecture has unique properties that make it more suitable for certain datasets. We will discuss these properties in chapter 5.

Performance difference DH Greece and DH Singapore Interestingly, we also find that the accuracy of each model is generally higher on DH Singapore than on DH Greece. This is counter-intuitive, as DH Singapore is a sparser dataset than DH Greece, which generally speaking should result in lower accuracy by the models. Unfortunately, Table 4.6 does not provide an immediate reason for this performance difference. Recalling that the main reason for DH Singapore's sparsity is the lack of product standardization, we believe that this might have resulted in less noise in the user-item interactions, which in turn resulted in higher accuracy values.

Beyond-accuracy metrics Looking at the beyond-accuracy metrics, we find that our neural models tend to have low catalog coverage on Beauty and SH Singapore in comparison to the non-neural models, presumably because of the sparsity. From Figure 3.2 we see that the majority of items have less than 10 interactions, and we assume that that these items are largely ignored by the neural models. We confirm this hypothesis in chapter 5. Serendipity seems to be strongly correlated with HitRate, indicating that models do not degenerate to a popularity baseline. The exception here is ItemKNN, which has a relatively higher share of correctly-predicted popular items than unpopular items.

Chapter 5

Recommendation analysis

We can get a deeper insight into the behaviour of the models by analyzing their recommendations. Firstly, in section 5.1 we compute the pairwise overlap between the recommendations of each model. We then continue with an analysis on several dimensions, namely the value K in section 5.2, item popularity in section 5.3, session length in section 5.4 and item position in section 5.5. The aforementioned analyses act as our motivation for our upcoming experiments in chapter 6. Moreover, the analyses will provide us more insight on how the open datasets are different from the Delivery Hero datasets, and therefore highlight promising avenues of research for Delivery Hero that are arguably underrepresented in the field of research on recommender systems.

We exclude SKNN from our analyses and visualizations below because the main goal of this section is to explain the behaviour of the neural models. We do this by comparing the neural models with ItemKNN and NextPopular because of their relative simplicity in comparison with SKNN.

5.1 How much do the recommendations overlap?

To get a first impression of the models' behaviour, we compare the models' recommendations to each other. We do this in two ways, namely by directly computing the overlap between the recommendations per session (*recommendation similarity*), and by computing the overlap in the sessions that each model correctly predicted (*session-correct similarity*). To be exact, the first approach entails computing the number of items that appear in recommendation slates of both models on the same session, averaged over all sessions. We then normalize by dividing by $K = 10$, which defines the size of the slates, and therefore also the maximum amount of items that can appear in both recommendation slates. For the second approach, given two models m_1 and m_2, we compute the number of sessions that both m_1 and m_2 got correct, normalized by the amount of sessions that m_1 got correct. For both cases, we compare the self-similarity of a model by training a model with the exact same configuration, and subsequently comparing the recommendations and sessions correct with the original model. The results of the two approaches are visualized in Figure 5.1 and Figure 5.2 respectively.

FIGURE 5.1: The models' recommendation similarities.

FIGURE 5.2: The models' session-correct similarities.

Recommendation similarity Firstly, we find that recommendation similarity is very dependent on the dataset. For Beauty, we have recommendation similarities of at most 15% between different models, and 27% between the same model (SASRec). Surprisingly, the recommendations between the two trained GRU4Rec or BERT4Rec models are almost indiscernable from other pairs of neural models. This indicates that the models are highly unstable on this dataset. On the denser datasets ML-1M and DH Greece we do find that instances of the same model and configuration behave similarly, but the difference in recommendation-similarity between the same model and two different models is surprisingly small.

Causes for instability Of course, we have introduced some inherent instability by randomly withholding 10% of the training sessions for early-stopping validation, but in the worse case 80% of the training sessions are the same for any two models. We performed an experiment where we set the seed for making the early-stopping-validation set deterministic, and found no considerable difference to the results in Figure 5.1. Therefore, the non-determinism of the early-stopping-validation set is not responsible for the low self-similarity. The two other sources of randomness in model training, namely the random initialization and the random batching process, appear to play a much larger role in final model behaviour. We will enhance self-similarity and in turn, performance, in chapter 6.

Session-correct similarity At the same time, we have that the session-correct similarity is much higher than the recommendation similarity. Still, it appears that session-correct similarity between two instances of the same model is not significantly distinguishable from the session-correct similarity between different neural models. This indicates that there is some shared driver behind performance. We will further explore this statement in chapter 6.

In addition, we find that the session-correct similarity between the neural models and ItemKNN is close to the neural models' self-similarity on Beauty and DH Singapore. The same holds for the neural models and NextPopular on DH Greece. This demonstrates the flexibility of the neural models, in the sense that they can mimic non-neural models depending on what works best on a given dataset. Given that NextPopular only uses the last item in a session, whereas ItemKNN uses all items in a session, we are interested in how the position of an item in the session affects its influence on the recommendations. We will explore this in section 5.5.

Ensembling models The fact that session-correct similarity is much higher than recommendation similarity indicates that consensus between models on a recommended item is a good predictor for the correctness of this recommended item. Therefore, we create an ensemble where the rank of an item is determined by the amount of models (excluding NextPop and Popular) that recommended the item. The results of this ensemble are summarized in Table 5.1.

Dataset	HR	NDCG
Beauty	0.065 (101%)	0.038 (102%)
ML-1M	0.337 (99%)	0.191 (90%)
DH GR	0.158 (102%)	0.090 (100%)
DH SI	0.186 (106%)	0.104 (104%)

TABLE 5.1: The evaluation results of an ensemble model where the rank of an item is determined by the amount of models (excluding NextPop and Popular) that recommended the item. The percentages in parentheses indicate the metric's value in comparison to the top model per dataset in Table 4.6. The ensemble surpasses the performance of the best model on Beauty, DH Greece and DH Singapore.

Indeed, an ensemble allows us to improve performance of the best model on Beauty, DH Greece and DH Singapore. Unfortunately, an ensemble is not desirable in production as it requires to run multiple models in parallel, where the slowest model will determine the latency. This does not weigh against the marginal improvement by the ensemble.

5.2 How does the K parameter affect model behaviour?

In the previous section we did not differentiate on the model's confidence (a recommended item's rank in the slate). While the value of 10 is most often used as the value for K in the offline evaluation of recommender systems, other use-cases like next music track recommendation might require a different value for K. Therefore, to generalize the results from section 5.1, we aim to explore a model's behaviour in terms of the rankings in the recommendation slates. We do this by taking the value for K into account.

Rank similarity We aim to provide a view similar to Figure 5.1 and Figure 5.2 while taking K into account. For a pair (m_1, m_2) of models, a given rank r and a session, we check whether the recommendation by model m_1 on rank r is present anywhere in the TOP-K recommendation slate of model m_2. We then take the average of these binary variables. We visualize the result in Figure 5.3. For discernability we only visualize the pairs (m_1, m_2) where m_1 is neural and m_2 is not, with the exception BERT as m_2 to show the interplay between the neural models.

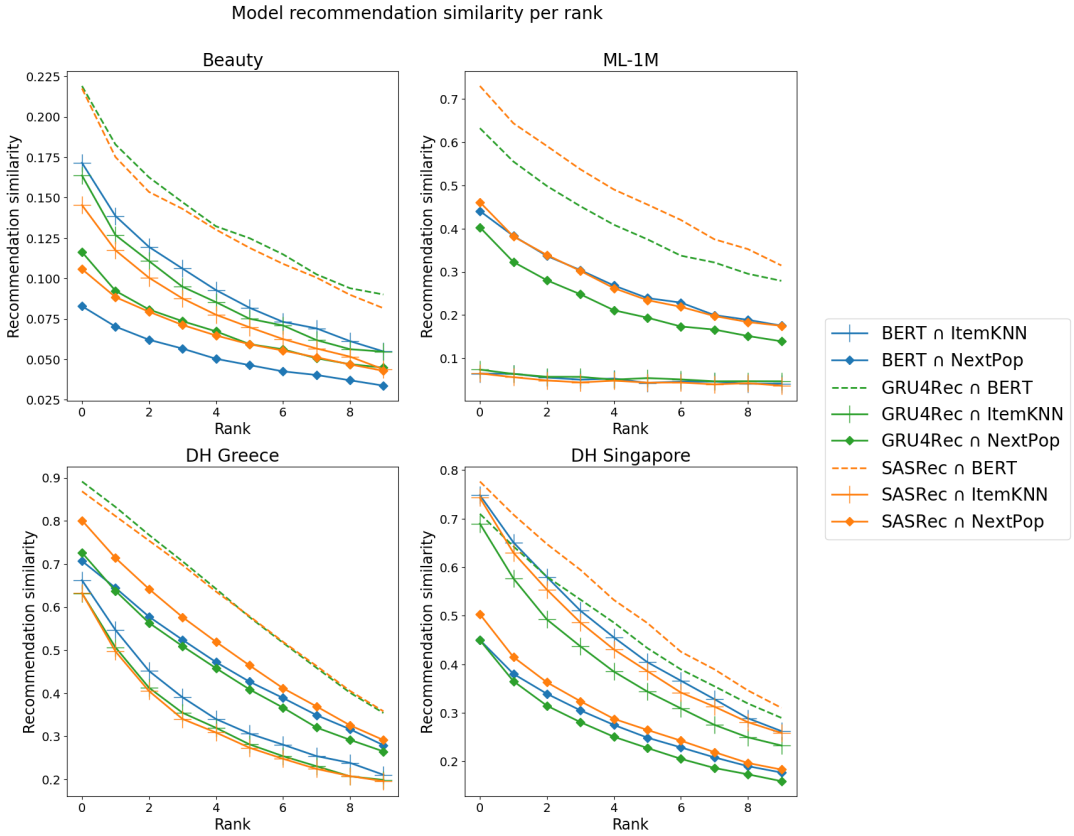

FIGURE 5.3: The models' recommendation similarity per rank. We use colour to denote model m_1, and we use the linestyle to denote model m_2.

General observations Expectedly, we find that there is a strong negative correlation between the rank of an item and the probability that it appears in another model's recommendation slate. So, for all datasets there is more consensus on the highly-ranked items than the lower-ranked items. We find this effect to be the strongest in the proprietary datasets, where more than 75% of the top-rank items can be explained by the extremely simple NextPopular model. In addition, we find that the ordering between NextPopular and ItemKNN as predictors of a neural models' recommendation slate is not consistent across datasets, and instead find that it matches the ordering in the evaluation results of Table 4.6. To illustrate, ItemKNN outperforms NextPopular on DH SI in Table 4.6 and it appears to be a better predictor of the neural recommendation slates on the same dataset. In contrast, NextPopular outperforms ItemKNN on DH GR, and therefore seems to be a better predictor of the slates on this dataset. This highlights the fluidity of the neural models in the sense that they can mimic a variety of models.

Effect of bidirectionality Furthermore, we can see that BERT4Rec is inherently more similar to ItemKNN than the other models, which can be explained by the fact that BERT4Rec is bidirectional. This means it is not restricted by the relative distance between two items to learn associations, like ItemKNN. On the other hand, the unidirectional SASRec and GRU4Rec can only learn an association from an item i to item j if item j succeeds item i. As a result, they can not fully mimic ItemKNN. Since

ItemKNN is the most performant model on DH Singapore, the additional performance of BERT4Rec on this dataset can be partially explained by the bidirectionality. We will further analyze the benefit of bidirectionality in section 6.2.

Rank similarity on sessions correct Besides comparing recommendations directly, we can filter our statistics to only include sessions that m_1 correctly predicted and present the results in Figure 5.4

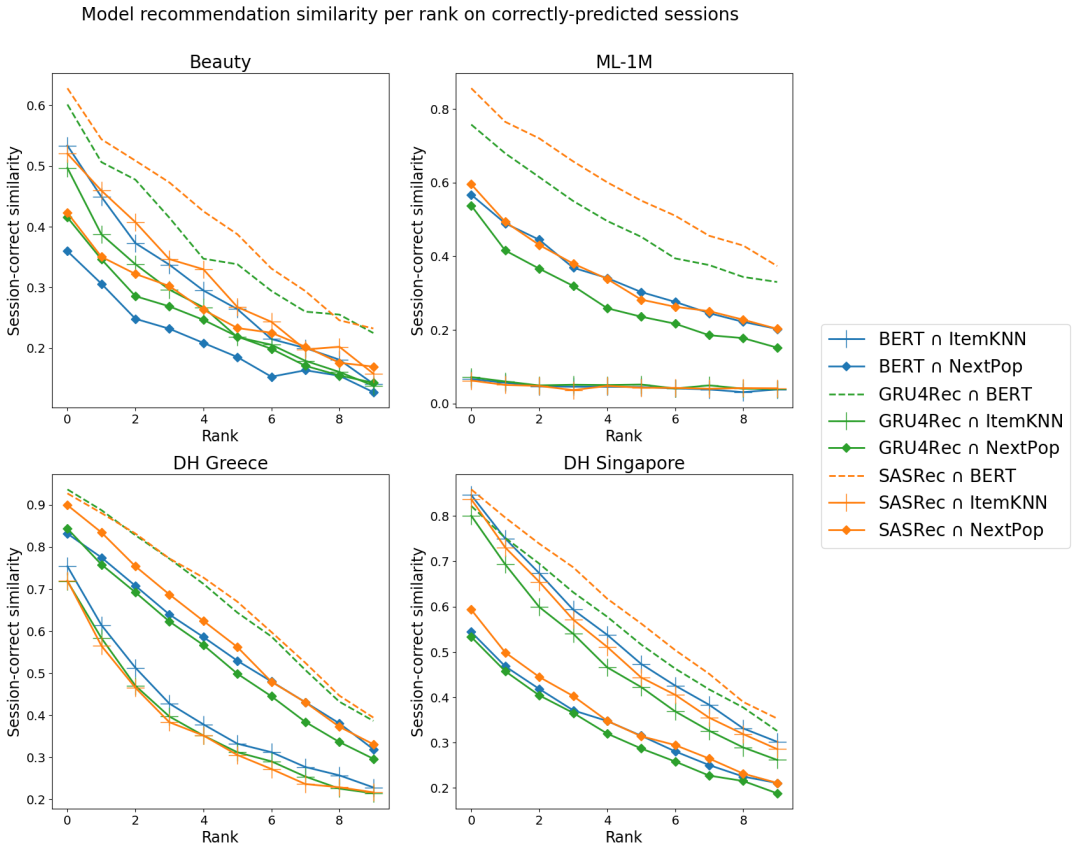

FIGURE 5.4: The models' recommendation similarity per rank on correctly-predicted sessions, meaning we compute recommendation similarity on sessions that model m_1 got correct. We use colour to denote model m_1, and we use the linestyle to denote model m_2.

.

Evidently, the consensus is much larger on recommendation slates where m_1 correctly predicted the ground-truth item. On DH GR we find that close to 90% of the top-ranked items were also recommended by the NextPopular baseline, and roughly 85% on DH SI by the ItemKNN baseline. This refers back to our discussion in section 5.1, where we found that item consensus is a good predictor for performance. With this figure we can conclude that this holds for all ranks roughly to the same degree.

5.3 How does item popularity affect model behaviour?

The most straightforward and well-studied dimension on which we can compare recommendations is item popularity. In recommender system literature, an item's

popularity refers to its number of interactions in the training data. It is crucial to understand a model's behaviour on different item popularities, as it can significantly affect the health of the system as a whole. For example, when users primarily rely on a recommender system to find items, a popularity bias could cause a reinforcing effect to the point where users only see the most-popular items, which in turn become more popular [25]. Therefore, we explore how item popularity affects model behaviour in multiple ways, namely item recommendation frequency against item popularity in subsection 5.3.1 and item precision against item popularity in subsection 5.3.2.

5.3.1 Item recommendation frequency against item popularity

In our initial simplest approach, we compare the number of times an item is recommended by each model against the number of times it occured in the training data (its *item popularity*). For visibility we group items into five equally-sized groups according to their popularity. The result is visualized in Figure 5.5.

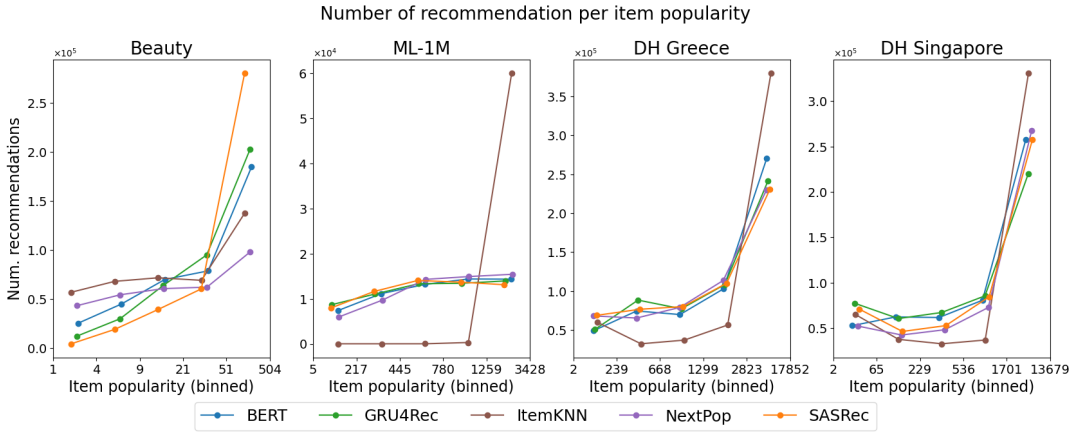

FIGURE 5.5: The models' number of recommendations against item popularity, where the item popularities have been grouped into five equally-sized bins. We added a small amount of horizontal jitter for visibility. We converted the y-ticks to use scientific notation.

General observations We find very little difference in the behaviour of the neural model on different item popularities. The only outlier is SASRec on Beauty, which appears to be recommending relatively more popular items. This is due to the fact that the optimal embedding dimension for SASRec on Beauty is only 32, which means that it is much harder to differentiate between different items. As a result, the best option for SASRec to minimize its loss is to recommend the popular items, as these will generally occur more often as the ground-truth item.

Expected distribution Unsurprisingly, we find that the bulk of recommendations entail the top-80% percent of items for all datasets. Assuming that the training sessions are representative of the test sessions, a well-fitted model should roughly recommend items proportional to the amount of times it appears in the training data. We should therefore observe a linear correlation in Figure 5.5. We note that the grouping process has slightly distorted this trend, but we generally find that all models adhere to this correlation with the exception of ItemKNN. Since ItemKNN recommends the items that have most-often occurred with all items in the session,

it is more prone to a popularity bias. ML-1M consists of long sequences, and so the effect is amplified on this dataset. Virtually no recommendations are made for the first 80% of items.

5.3.2 Item precision against item popularity

Furthermore, we are interested in item precision againt item popularity [1]. The precision of an item is calculated by the number of times an item was correctly predicted to be the ground-truth divided by the number of times the item was recommended. The result is visualized in Figure 5.6.

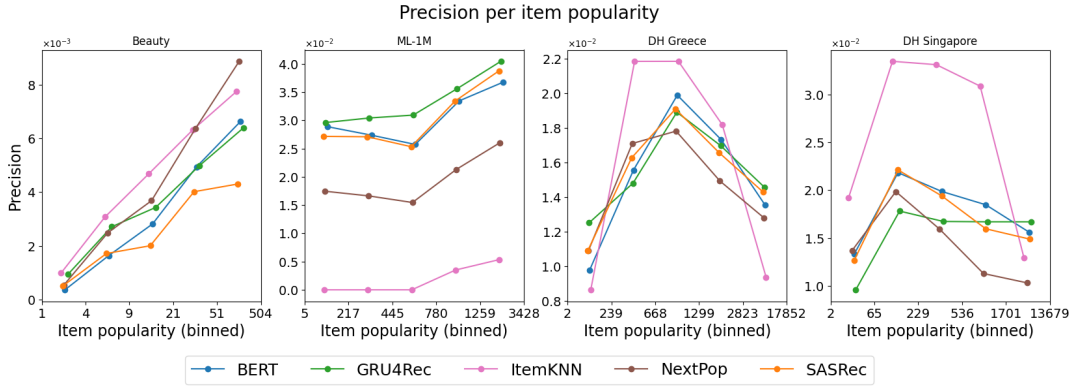

FIGURE 5.6: The models' precision against item popularity, where the item popularities have been grouped into five equally-sized bins. Precision is calculated by the number of times an item was correctly predicted to be the ground-truth divided by the number of times the item was recommended. We added a small amount of horizontal jitter for visibility.

General observations On Beauty and ML-1M, we observe a stable upward trend in precision by the neural models, meaning that the probability of a recommended item to be correct is roughly proportional to its popularity. Naturally, when more data is available for a given item a model can learn to better predict its suitability as the next-item for a given session. However, we see the opposite on the Delivery Hero datasets. In fact, we see that this probability drops significantly, meaning that all models are overconfident in recommending popular items.

Popularity bias While this *popularity bias* is often attributed to deep learning models for recommender systems [11], we also see that our non-neural baselines suffer from the same performance degradation. We attribute this to the fact that the popular items in DH Greece and DH Singapore have a higher *relative* popularity. For example, the most popular item in DH Greece constitutes roughly 0.75% of item interactions, whereas on Beauty the most popular item constitutes roughly 0.2% of item interactions. Therefore, we believe these items occur more often as noise in the training data, to the degree where the models mis-recommend these items relatively more often during the evaluation phase.

[1] We do not visualize the HitRate of an item against its popularity because it is strongly correlated with the number of times it is recommended. The precision of an item is independent of the number of times it is recommended, and therefore provides a more complementary view to Figure 5.5.

Mitigating popularity bias This indicates that reducing this overconfidence (*debiasing*) might lead to performance gains. Therefore, we suggest the implementation of models that address this problem to Delivery Hero. For example, RPβ [70] is an ItemKNN variant that reduces overconfidence in popular items by scaling the score for each item by their popularity raised to the power of β. For neural models, the main stream of work to address popularity bias is through customized loss functions that weigh scores based on popularity [97], or advanced negative sampling strategies [33, 73]. On the other hand, while ItemKNN is most prone to a popularity bias according to Figure 5.5, this same bias also makes it the most effective model on the DH Singapore dataset in Table 4.6. In this thesis, we will not focus on debiasing our models explicitly, given that we focus on the architecture of the models, but we do note that some of our simple models introduced in chapter 6 will be more robust against popularity bias.

5.4 How does session length affect model behaviour?

Besides item popularity, a fundamental dimension on which we can compare models is on session length. Session length is an important statistic that balances between the amount of information available to make the recommendations and the amount of noise by historical interactions that might disturb the quality of the recommendations [96]. To explore how session length affects performance on our datasets, we group sessions by their length by taking each 20th percentile as a bin[2]. We then calculate HitRate for each model per group of sessions. The result is visualized in Figure 5.7.

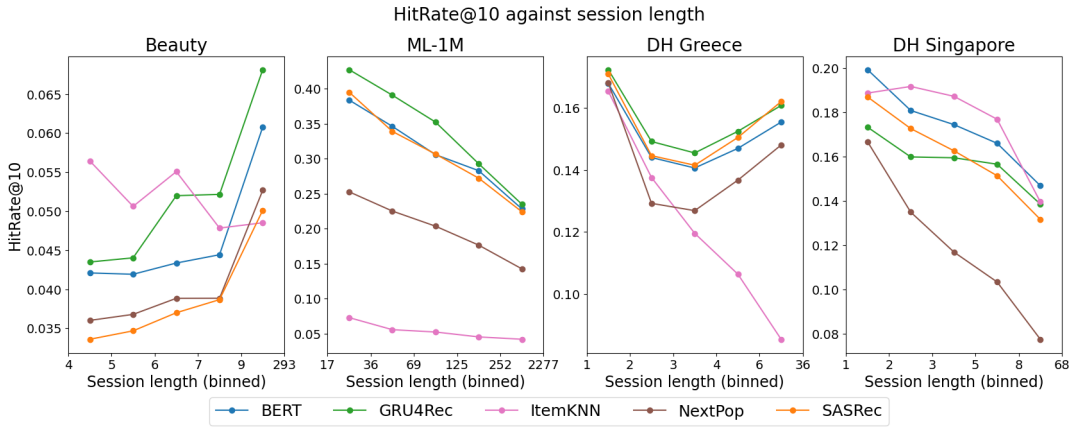

FIGURE 5.7: The models' HitRate against session length, where the session lengths have been grouped into five equally-sized bins.

General observations We observe inconsistency in the trends across datasets. On Beauty, performance seems to decrease slightly on longer sessions for the ItemKNN model, whereas the neural models and NextPopular seem to increase in performance. On ML-1M, performance decreases with session length for all models. On our proprietary datasets, we find that the models are generally most performant on single-item sessions. Interestingly, we have that performance decreases significantly

[2]We also enforce that the groups are mutually exclusive by ensuring that the bin size is at least 1. This was necessary for the Beauty dataset, where the both the 20th and the 40th percentile of the sessions lengths is 4.

with session length on DH SI, but on DH GR this only holds for ItemKNN. The rest of the models seem to be stable or even increase in performance.

Short-term and long-term context At this point, we would like to recall the general belief in the field of recommender systems that the GRU encoder is suitable for *short-term context* modelling [40, 56, 100], whereas the transformer layers are better at *long-term context* modelling [87, 47, 100, 62]. While this may be true for their original applications like machine translation [12, 16, 93, 90], Figure 5.7 shows that it does not hold for our session recommendation datasets. For example, GRU4Rec is most performant on ML-1M, while ML-1M is most often referred to as the dataset that measures a model's ability to model long-term context [87, 100, 101]. Moreover, SASRec outperforms GRU4Rec on DH Singapore on the 60% of lower item popularities, after which GRU4Rec seems to outperform SASRec on the 40% of higher item popularities, completely opposing the aforementioned supposition.

NextPopular's invariance to session length From the results on NextPopular, we can conclude that the last item is less indicative of the next item on DH Singapore than on DH Greece. We can draw the opposite conclusion for Beauty, where NextPopular's performance increases with the length of the session. We believe the latter is due to the fact that Beauty is actually a review dataset with a large timespan. Longer sessions imply that the relative time between reviews may be shorter, so that the relatedness between the last item and next item increases. This would explain the higher performance of NextPopular on the longer sessions. Interestingly, a similar effect can be observed for DH Greece, because the curve of NextPopular is very similar to the curve of the neural models. This indicates that on this dataset, the dip in performance in the middle segments of session lengths can not be caused by the session length itself. Instead, session length seems to be correlated with the last item's relatedness to the next-item. We leave an analysis on the root cause of this phenomenon to future work at Delivery Hero.

As a side note, our observations with NextPopular are relevant to the motivation of ASReP [62], which uses the same curve of Beauty visualized in Figure 5.7 to state that *transformers specifically* are worse on short sessions and conversely, better on longer sessions. However, we found that this is inherent to the dataset instead of the transformer models specifically, since the performance of NextPopular also increases with session length.

5.5 How do items at different positions in the sessions affect the recommendations?

Given the surprising results on the session length, we aim to explain these results further by analyzing how item position plays a role in the computation of the recommendations. We design two approaches to measure each position's effect on the model recommendations, namely session truncation in subsection 5.5.1 and *trivial* recommendation overlap in subsection 5.5.2.

5.5.1 Session truncation

Motivated by the finding that the models seem to mimic certain non-neural algorithms (NextPopular and ItemKNN), we are interested in how much the recommendations are affected when only feeding a truncated part of a session. For example,

given session (s_1, s_2, s_3), we first feed (s_3), then (s_2, s_3) and so forth. We visualize NDCG@10 against the maximum number of items in the session in Figure 5.8. Furthermore, we want to measure how much the recommendations change when we increase the maximum session length. Hence, we visualize the recommendation overlap of the models to their recommendations if only fed the last item in Figure 5.9.

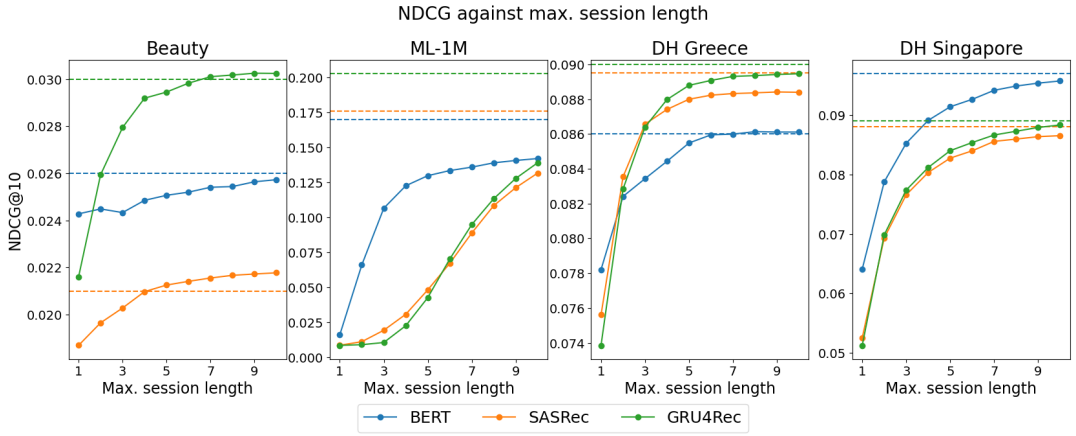

FIGURE 5.8: The models' NDCG against maximum session length, so that all sessions are truncated to be at most the maximum session length. The dashed lines indicate the models' NDCG@10 using all items of the session.

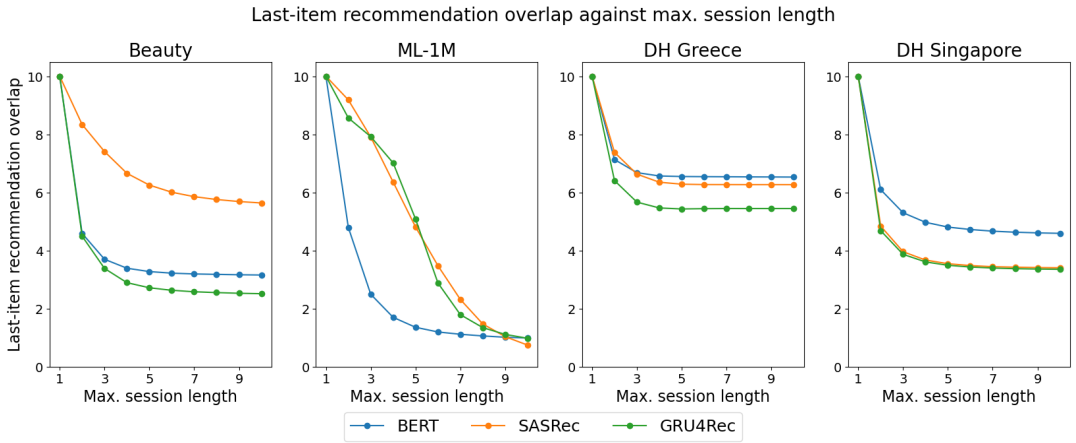

FIGURE 5.9: The models' recommendation similarity with their last-item recommendations against maximum session length. In short, we first let the model produce recommendations feeding only the very last item of the sessions and compute the overlap with the recommendations when the truncated session was provided.

General observations The NDCG@10 attained using just the last item depends on the dataset. On DH Greece we have that the NDCG@10 of the models on just the last item is within 20% of the overall NDCG@10. In contrast, on ML-1M we have that the optimal NDCG@10 is not even within 90% of the overall NDCG@10. Moreover, the maximum session length of 10 does not even approach the overall NDCG@10 yet. This means that the models need more items in order to compute good recommendations on ML-1M.

Training N vs. Prediction N Also, we find that on Beauty, the performance of the models sometimes exceed the overall NDCG@10, meaning that the truncation in this case improves performance. This is counter-intuitive, as we initially chose our parameter N (the maximum session length in all other experiments, and also the maximum session length for the overall NDCG@10) by performing a preliminary hypersearch on this parameter. However, the crucial insight here is that N implicitly balances between the amount of training data a model receives (positive) and the amount of noise it receives during prediction (negative). Hence, the models seem to benefit from a longer N to learn more from the data, but during prediction the models seem to be incapable of properly filtering the noise that comes from the tails of the sessions. As a result, we recommend to split the N hyperparameter to $N^{(\text{training})}$ and $N^{(\text{prediction})}$, where the former denotes the session lengths during training and the latter denotes the session lengths during prediction. Having a lower $N^{(\text{prediction})}$ than $N^{(\text{training})}$ is also positive in the sense that we reduce inference time.

Long vs. short term modelling Again we find that GRU4Rec seems to be more suitable for long-term preference modelling, as its performance up until the maximum session length of 10 is the same as BERT4Rec and SASRec, but subsequently appears to climb to a much higher NDCG using the older items in the session as well.

Ranking vs. recall In addition, we find that the plateau of the recommendation overlap in Figure 5.9 is reached faster than the plateau of optimal NDCG@10 in Figure 5.8. The plateau in Figure 5.9 indicates the maximum session length at which roughly all items in the recommendation slate are determined, because the recommendation overlap does not change. Since these maximum session lengths are lower than the maximum session lengths at which NDCG@10 plateaus in Figure 5.8, we can conclude that the figures indicate that the last items are predominantly used for the recall of the recommended items, after which the tail of the session is used to improve the ranking of these recommended items.

5.5.2 Trivial recommendation overlap

Furthermore, we design an orthogonal way to measure an item's effect on a model's recommendations, in the hopes of explaining the recommendations that were not accounted for in Figure 5.1, and subsequently explain what behaviour causes the performance increase of the neural models compared to the non-neural models. To do this, we compute a fixed set of *trivial* recommendations per item, and compare the overlap between a model's recommendation and the fixed set of recommendations for each item in a session. Hence, for each position in a session, we get the average overlap between the final model recommendations and the fixed set of recommendations of the items that appeared on that position. Being a first-order MDP (meaning it only depends on one item), NextPopular is the obvious choice for generating the trivial recommendations. We visualize the design of the experiment in Table 5.2. Lastly, we also compute the session-correct similarity between the models' and the trivial recommendations. The result of this experiment is visualized in Figure 5.11 [3].

[3]We note that the HitRate@10 of the trivial recommender in Figure 5.11 is not the HitRate@10 of NextPopular in Table 4.6. This is due to the fact that the trivial recommender does not remove items that are already in the session, while NextPopular does.

	Position →			
Session	1. Bananas	2. Spaghetti	3. Beer	4. Peanut butter
Trivial rec's	1. **Apples** 2. Pears 3. Oranges	1. Tomato sauce 2. Ground beef 3. Salt	1. Wine 2. Heineken 3. Coca-cola	1. **Nutella** 2. **Strawberry jam** 3. Bread
Model rec's	1. **Nutella** 2. **Strawberry** jam 3. **Apples**			
Overlap	1	0.0	0.0	2

TABLE 5.2: Example calculation of the trivial recommendation overlap per position. In this simplified example, $K = 3$.

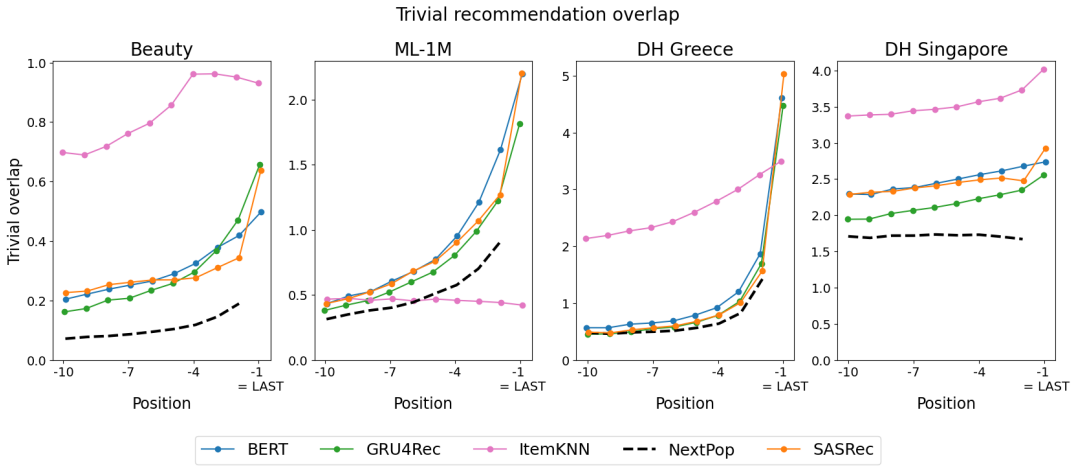

FIGURE 5.10: The models' trivial recommendation overlap against position. Position -1 denotes the position of the last item. The y-axis denotes the trivial overlap, which is the average number of items in the TOP-K recommendation slate that were also recommended by the trivial recommender on each position. Since NextPopular is the trivial recommender itself, we do not visualize its overlap on position -1, which would be $K = 10$.

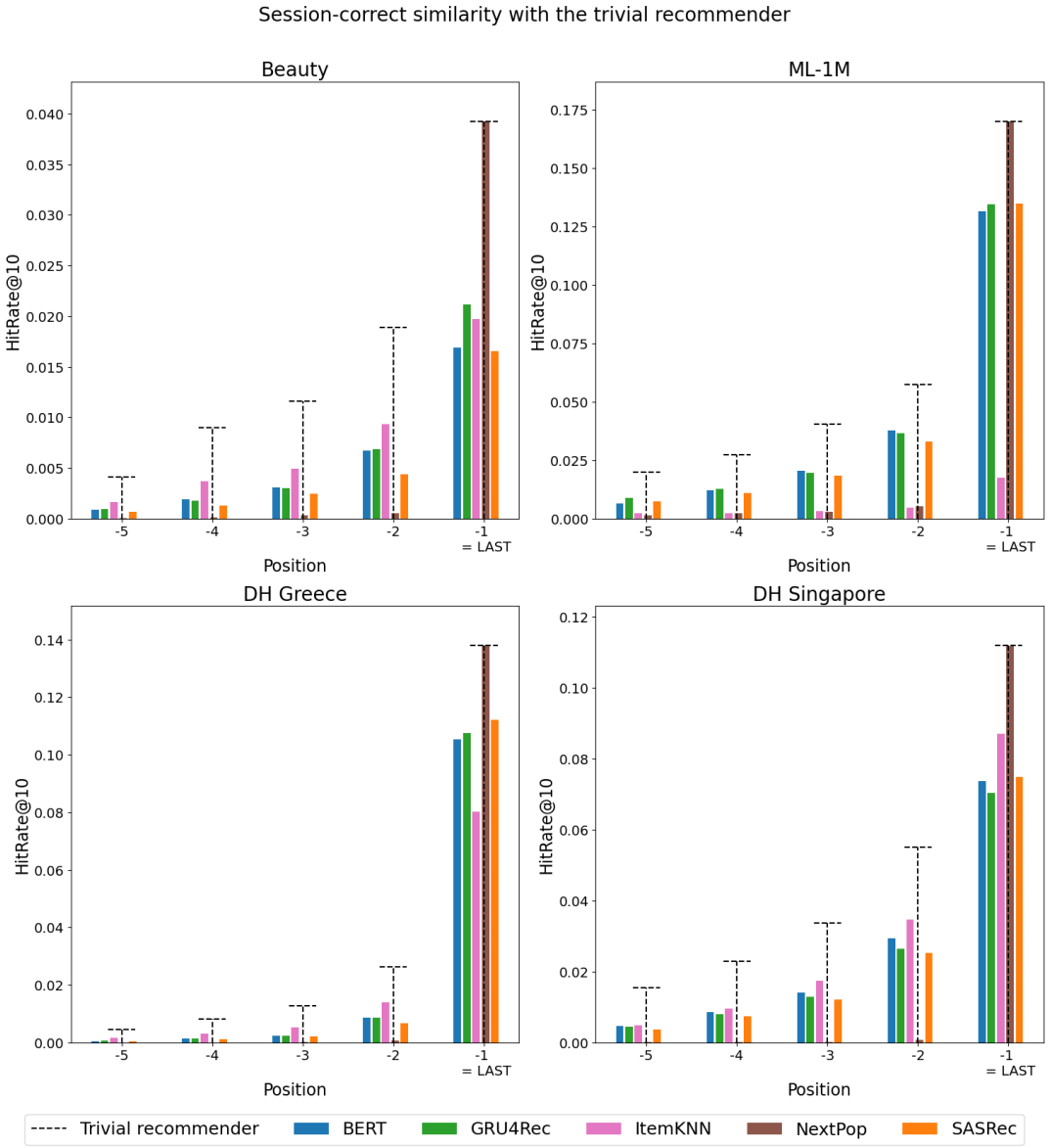

FIGURE 5.11: The models' session-correct similarity with the trivial recommender against position. Position -1 denotes the position of the last item. We compute the sessions that the trivial recommender got correct on each position, and compute the overlap in the sessions correct at each position. We account for duplicates by only counting a session correct for a position if later positions did not correctly predict this session. The y-axis denotes the HitRate@10, which is the number of correctly-predicted sessions normalized by the total amount of sessions.

General observations Evidently, the neural models are very similar in how much their recommendations overlap with the trivial recommendations, and all three seem to adapt to using trivial recommendations from previous positions when these trivial recommendation improve performance on a given dataset. Figure 5.10 also partially explains the HitRate against session length distribution for DH Greece in Figure 5.7 and the relatively high recommendation overlap plateau in Figure 5.9, because the models seem to have relatively low overlap with the trivial recommender on other positions than the last. This explains why the models seem invariant to

the session length in Figure 5.7. However, this is also to be expected, as Figure 5.11 shows that the trivial recommender has a low HitRate@10 on the other positions than the last anyway.

Interestingly, the trivial recommender has a steeper decline in HitRate@10 on ML-1M than on Beauty and DH Singapore. This is in sharp contrast with the results on session truncation on ML-1M in Figure 5.8 that shows that the models need relatively more items to approach their overall NDCG@10. So, these older items are apparently necessary to compute good recommendations, but their trivial recommendations are not necessarily good recommendations. This indicates that the models are capable of exploiting some form of *context* instead of simply choosing from the trivial recommendations. However, ML-1M is the only dataset on which the models exhibit this behaviour.

Explaining recommendations From the offset from the NextPopular line on the Beauty, ML-1M and DH Singapore datasets in Figure 5.10, we can apparently explain some additional recommendations because they originate from the trivial recommendations for the older items in the session. Table 5.3 and Table 5.4 provide a more precise view.

Dataset	BERT4Rec	SASRec	GRU4Rec
Beauty	15.9%	15.5%	16.9%
ML-1M	60.7%	58.4%	51.3%
DH GR	59.7%	60.1%	56.6%
DH SI	61.7%	60.0%	56.1%

TABLE 5.3: The total trivial recommendation overlap with the last 10 positions. We accounted for duplicates, which is why the overlap is not directly the area under the curve of Figure 5.10. In short, this is the recommendation similarity between the neural models and a model that recommended $10 * K = 100$ items, where every 10 items are trivial to a given position.

Dataset	BERT4Rec	SASRec	GRU4Rec
Beauty	66.1%	69.0%	67.9%
ML-1M	73.4%	73.1%	69.3%
DH GR	79.7%	80.6%	78.3%
DH SI	80.5%	81.8%	82.6%

TABLE 5.4: The average session-correct similarity with the trivial recommendations. In short, this is the share of sessions that each neural model got correct that were also correctly recommended by our trivial recommender. The trivial recommender recommended $10 * K = 100$ items, where every 10 items are trivial to a given position.

Interestingly, we find that around 60% of all recommendations by the neural models on the DH datasets can be explained by the trivial recommendations. In fact, we find that the numbers in Table 5.3 are similar or higher than the recommendation self-similarity from Figure 5.1. If we take the share of sessions that each model got correct that were also recommended correctly by the trivial recommender, we reach up to 80% similarity on the DH datasets. The increase between recommendation similarity and session correct similarity with the trivial recommender is most

significant on the Beauty dataset, where we go from 16% to almost 70%. Hence, the recommendations that were not accounted for by the trivial recommender have a much lower precision, because there is a significant difference between Table 5.3 and Table 5.4.

Exploiting trivial recommendation overlap Lastly, we note that the large overlap with this deterministic, trivial recommender has many uses. For example, when training with negative sampling, we can use the trivial recommendations as hard negatives during training [67], or we can speed up inference by only sampling the scores in R_t for the items recommended by the trivial recommender. Other publications like [40] resort to sampling popular items, but this naturally introduces a bias in the recommendations towards the popular items.

5.6 Discussion

In general, we find that the neural models are very similar in behaviour. We have found that two trained instances of the same model are almost as similar in recommendation behaviour as two trained instances from two different neural models. Moreover, we find that the non-neural models are good predictors of the recommendations by a neural model, especially on the top-ranked items. Lastly, we find that on other dimensions like session length, item popularity and trivial recommendation overlap there is very little difference in the behaviour of the models. All our results in this chapter essentially indicate the presence of a core driver of performance. Therefore, in chapter 6, we aim to constructively create a model that captures this core driver of performance while keeping the number of architectural components to a minimum. Moreover, we will explore how we can exploit the finding from Figure 5.11 that a significant amount of performance can be obtained by focusing on trivial recommendations.

Chapter 6

Simplifying models

Given the results in chapter 5 that the neural models behave very similar, we hypothesize that that there is a shared driver behind performance. Therefore, we also hypothesize it is possible to simplify the models while maintaining performance. We note that the publications on BERT4Rec [87] and SASRec [47] do perform an ablation study, but this was done by removing components individually. Instead, we believe that the existing model designs might be in a local minimum in terms of performance, so that each ablation of a component did mitigate the effects of overparameterization, but the architecture overall is still sub-optimal. Instead, we will construct the models by adding different components incrementally in the hopes of converging to a simpler but more performant model. Simultaneously, we aim to identify the effect of each component. In section 6.1, we optimize our simple model to only use the last item of a session during prediction. The motivation for this initial focus is the finding that the last item is most indicative of the next item in Figure 5.11. In section 6.2, we will subsequently compare increasingly complex ways of exploiting non-last items. Here, we will focus on exploiting the trivial recommendations from other positions.

6.1 Last item optimization

In chapter 5, we found that most of the models' performance can be attributed to the last item on some datasets. Moreover, the non-neural NextPopular model approaches the performance of the other models in Table 4.6, implying that the last item is indeed the most indicative of the next item. As such, we might not actually need the complex transformer or GRU layers that were designed to merge information from other parts of the session. Therefore, this section is intended to optimize a model operating only on the last item of a session. In subsection 6.1.1 we introduce LastEmbedding, a model that computes recommendations with the embedding of the last item only. We will provide our interpretation to this model in subsection 6.1.2. We then discuss the results of this model in subsection 6.1.3. Finally, we evaluate the various architectural components from the vanilla neural models that can be applied in the context of last-item optimization in subsection 6.1.5. Lastly, we demonstrate that our learned item associations are close to optimal with a variation on our standard training task in subsection 6.1.6.

6.1.1 Neural base model (LastEmbedding)

We start from a model that simply learns embeddings governed by Equation 6.1 called LastEmbedding. Recall that the scores for each item on each timestep, R is of shape $(N, |\mathcal{I}|)$, the item embedding matrix E is of shape $(|\mathcal{I}|, e)$, and the embeddings of the items in the session E_s is a submatrix of E and of shape (N, e).

$$R = E_s E^T \tag{6.1}$$

Note that $R_t = E_{s_t} E^T$, where E_{s_t} is the embedding of item s_t on timestep t in session s. The approach to model training is exactly the same as SASRec and GRU4Rec, where on position t the models are tasked with predicting the identity of the item on position $t + 1$ (See Figure 2.3).

Therefore, R_N, the prediction for the next-item, is only dependent on the last item s_N. Like the models in chapter 4, we use the Softmax loss to train LastEmbedding. A major benefit of LastEmbedding is that the recommendation slate can be precomputed per last item. As a result, it is highly scalable.

Reusing item embeddings Also note that LastEmbedding uses its embedding matrix for both input and output items. Re-using the item embedding matrix this way reduced model size and improved the performance of GRU4Rec, SASRec and BERT4Rec [39, 47, 87]. Our preliminary experiments also show that this greatly improves performance on three out of four datasets. The intuition behind using these *homogeneous* embeddings is that if item i is a good recommendation for item j, then item j would be a good recommendation for item i (commonly referred to as *symmetric* item transitions). In contrast, the motivation for *heterogeneous* embeddings, where the input and output embeddings are different matrices, would be that it supports *assymetric* item transitions. So, i might be a good recommendation for j, but j might not be a good recommendation for i. To give an example of such a scenario, the purchase of a screw-driver might be often succeeded by the purchase of a screw-driver set, in case that the user is content with the initial individual screw-driver. However, by looking at individual shopping carts we have concluded that such scenarios are rare at the least in the context of Q-Commerce. This explains why sacrificing asymmetry for a model size reduction results in better performance. Therefore, with *LastEmbedding* we refer to the homogeneous version of the embedding model. We include the results of the heterogeneous model in Table 6.1.

6.1.2 Interpretation

Intuitively, LastEmbedding now simply consists of items as points in an e-dimensional space, and draws items together if they appear successively in a training session. During prediction, the model simply returns the items closest (in terms of dot-product similarity) to the last item in the e-dimensional space. We visualize the workings of the base model in Figure 6.1.

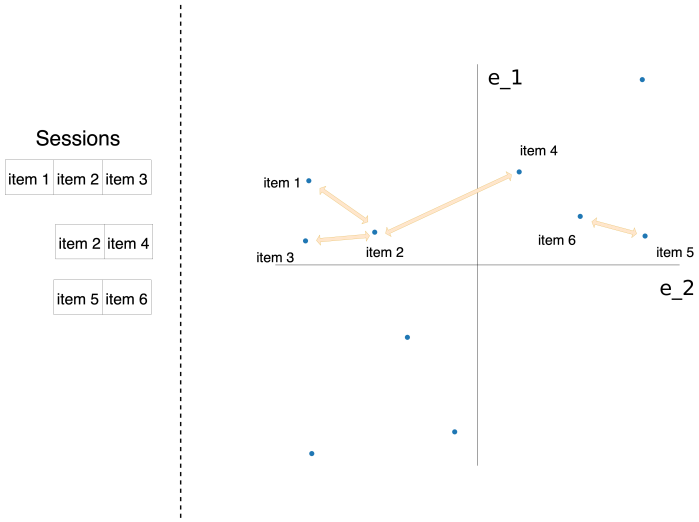

FIGURE 6.1: A visualization of the LastEmbedding model. Each item is represented by a point (embedding) in a multi-dimensional space, which has been simplified to 2 dimensions (e_1 and e_2) in this figure. Given the training sessions on the left, the model draws the embeddings of successive items closer. Note that the figure is slightly misleading, as the model aims to optimize the dot-product similarity between two embeddings. Therefore, the gradients are not actually in the same direction as the arrows. In this figure, the arrows merely indicate that the model learns the association.

6.1.3 Non-neural base model (NextPopular)

We note that we had already integrated the idea of homogeneity in our original version of NextPopular. By counting neighbours instead of directly succeeding items, we have that the count for item j on item i is the same as the count for item i on item j. As a result, NextPopular constitutes the non-neural equivalent of LastEmbedding. For reference, we also include the results of the heterogenous version of NextPopular in Table 6.1. To clarify, this version only counts directly succeeding items, and recommends the items that have most often succeeded the last item of the session.

6.1.4 Results

We summarize the results of the homogenous and heterogeneous version of the embedding and NextPopular models in Table 6.1. We note that we did optimize the hyperparameters of the embedding models, but found that the embedding models are very resilient against its hyperparameters. Therefore, we simply continue with the configuration $e = 256$, learning rate $= 1e - 3$, and batch size $= 256$ for the rest of this chapter unless stated otherwise.

Dataset	Model	HR	↓NDCG	CatCov	Ser	Nov
Beauty	LastEmb	.056 (87.5%)	.034 (91.9%)	.707	.053	14.0
	NextPop	.041 (64.1%)	.026 (70.3%)	.664	.039	13.5
	LastEmb (he)	.040 (62.5%)	.024 (64.9%)	.574	.036	13.0
	NextPop (he)	.033 (51.6%)	.022 (59.5%)	.595	.031	13.4
ML-1M	NextPop (he)	.227 (62.7%)	.136 (62.7%)	.577	.224	10.6
	NextPop	.210 (58.0%)	.119 (54.8%)	.529	.206	10.5
	LastEmb (he)	.156 (43.1%)	.101 (46.5%)	.330	.152	9.7
	LastEmb	.151 (41.7%)	.088 (40.6%)	.486	.146	10.8
DH GR	LastEmb	.143 (92.9%)	.084 (93.3%)	.797	.124	10.6
	NextPop	.140 (90.9%)	.083 (92.2%)	.796	.120	10.5
	LastEmb (he)	.140 (90.9%)	.083 (92.2%)	.708	.119	10.4
	NextPop (he)	.135 (87.7%)	.082 (91.1%)	.785	.117	10.6
DH SI	LastEmb	.129 (73.7%)	.073 (73.0%)	.380	.099	10.4
	NextPop	.114 (65.1%)	.067 (67.0%)	.335	.089	10.6
	LastEmb (he)	.113 (64.6%)	.063 (63.0%)	.283	.083	10.1
	NextPop (he)	.100 (57.1%)	.059 (59.0%)	.294	.077	10.6

TABLE 6.1: Evaluation results of the homogeneous and heterogeneous variants of LastEmbedding and NextPopular. The percentages in the HR and NDCG column denote how close the result is to the top value in Table 4.6.

General observations First of all, we find that the LastEmbedding model outperforms the the neural models on the Beauty dataset (but not yet SKNN), evidencing that reducing the parameterization of the neural architectures may be beneficial on certain datasets. The opposite is true for the ML-1M dataset, where the last item is apparently not indicative enough of the next-item to be competitive with the neural models. Interestingly, LastEmbedding approaches the top performance on DH Greece, but does not on DH Singapore. We find that the performance of LastEmbedding is very much correlated with the performance of its non-neural equivalent NextPopular. In chapter 5 we had already established that other items than the last are relatively more indicative of the next-item in DH Singapore than DH Greece, which explains the difference in performance of LastEmbedding.

Benefits of a neural model The results between NextPopular and LastEmbedding also demonstrate the benefit of a neural approach to session recommendation. While both are trained very similarly by associating direct neighbours with each other, we find that LastEmbedding outperforms NextPopular on Beauty, DH Greece and DH Singapore. The exception here is ML-1M, where NextPopular is shown to be much more performant. We can explain the performance difference between NextPopular and LastEmbedding by analyzing the models' accuracy against the popularity of the last item in the session. The last item is the only source of information for both LastEmbedding and NextPopular, so the last-item popularity is a dimension on which we can fairly compare the models' efficiency. We visualize the performance against last item popularity in Figure 6.2.

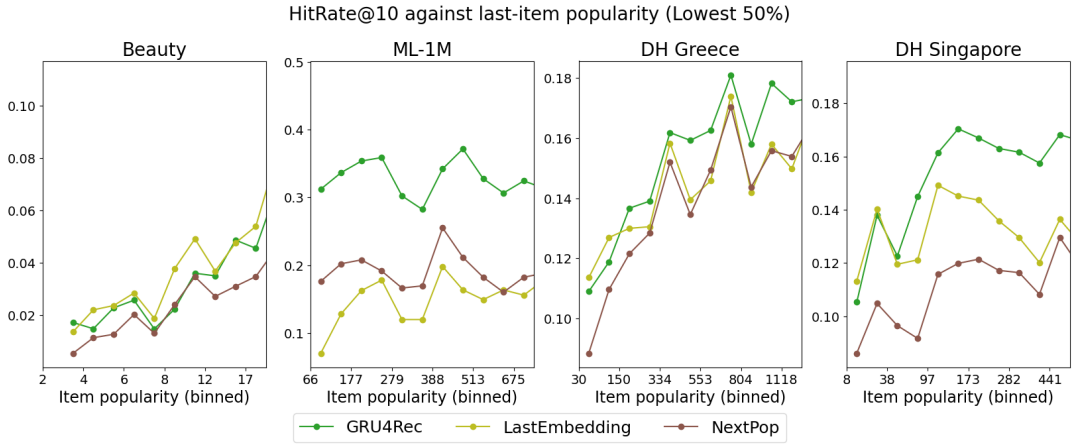

FIGURE 6.2: HitRate@10 against last-item popularity of GRU4Rec, LastEmbedding and NextPopular. This figure only includes the bottom 50% of items in terms of popularity.

We observe that the superiority of LastEmbedding over NextPopular can mainly be attributed to the performance on low-popularity last items. We find that LastEmbedding outperforms NextPopular on low-popularity items (< 500 interactions), after which their behaviour becomes very similar on popular items. Since Beauty only has unpopular items in absolute terms, we find that the performance difference between LastEmbedding and NextPopular is the greatest on this dataset. In contrast, DH Greece has a relatively small amount of unpopular items, causing only a small performance difference.

NextPopular needs relatively more interactions to learn a stable top-K recommendation slate per item because it only relies on co-occurence counts. On the other hand, LastEmbedding (and neural models in general) can leverage information from other items through similarity in the embedding space, leading to a stable top-K recommendation slate with much fewer interactions. This finding also motivates us to investigate whether we can use more than just the direct neighbours in the session to improve performance. We will explore this question in subsection 6.1.6.

Unfortunately, we also find that relying on similar items worsens performance on ML-1M. For ML-1M we have already found that the context in which movies were rated is extremely important[1] (see our session-truncation experiment in subsection 5.5.1 for example). As a result, interactions from similar items might act as noise to the LastEmbedding model, causing the inferior performance.

Beyond-accuracy metrics Also, we find that the homogeneous LastEmbedding version has more desirable beyond-accuracy metrics than the heterogeneous version and the vanilla neural models. Its catalog coverage reaches up to double the catalog coverage by the vanilla neural models, and its novelty is consistently higher than all other models, with the exception of GRU4Rec on DH Singapore. On DH Greece we find that the serendipity of LastEmbedding is 97% of the top Serendipity in Table 4.6, meaning that the majority of the gains by the neural models in comparison to LastEmbedding are gains made by correctly predicting the top-10 most popular items.

[1]To give a concrete example of different contexts in which movies could be rated, a rating for "Mamma Mia!" could be a user rating his/her favorite movies starring Meryl Streep, but it could also be a user rating his/her favorite comedies. Since Meryl Streep acts in many different genres, the interactions from either context could be very misplaced in the other context.

Comparison with session truncation Furthermore, from our experiment on session truncation in subsection 5.5.1 we can conclude that LastEmbedding significantly outperforms the neural models on all datasets if only the last-item is provided to the models. For convenience, we present the best NDCG@10 attained with just the last item in Table 6.2. The result implies that the vanilla models are less suitable for recommending items for sessions of unit length, in exchange for the ability to provide better recommendations on longer sessions.

Dataset	Vanilla models (last)	Vanilla models (all)	LastEmbedding
Beauty	.024	.030	0.034
ML-1M	.020	.203	0.088
DH GR	.078	.090	0.084
DH SI	.065	.097	0.073

TABLE 6.2: The best NDCG@10 attained by the vanilla neural models using the last item in our session truncation experiment in subsection 5.5.1. All values were attained by the BERT4Rec model. We also add the top NDCG@10 attained by the neural models when provided with the whole session, and add the NDCG@10 column of the LastEmbedding results from Table 6.1. This table illustrates that LastEmbedding is more data-efficient and learns better item embeddings, because LastEmbedding only operates on the last item of a session.

6.1.5 Architectural improvements

Having established a surprisingly performant model that only operates on the last item, we now continue with optimizing this model while maintaining its independence from other items in the session than the last. We will iteratively add components taken from or inspired by the original models. The goal of this subsection is to evaluate the performance contribution of various architectural components that were introduced in either BERT4Rec [87], SASRec [47] or GRU4Rec [40].

Embedding dropout The simplest way to improve performance is by adding embedding dropout [26]. This technique randomly sets entries of E_s to zero and rescales the rows of the matrix to retain its L2 norm. It is used by all three vanilla models. Similar to our implementations of these models, we add the hyperparameter drop_rate , and tune it with the rest of the parameters fixed. We will denote a matrix with the superscript ^{D} if we pass it through a dropout layer to avoid the cluttering of subsequent equations.

$$R = E_s^D E^T \tag{6.2}$$

Dataset	HR	NDCG	CatCov	Ser	Nov
Beauty	.056 (0.0%)	.034 (0.0%)	.676	.053	13.8
ML-1M	.189 (+25.1%)	.107 (+21.6%)	.595	.184	10.5
DH GR	.143 (0.0%)	.084 (0.0%)	.783	.124	10.5
DH SI	.129 (0.0%)	.073 (0.0%)	.375	.099	10.4

TABLE 6.3: Evaluation results of embedding dropout on the LastEmbedding model. The percentages indicate the performance difference with the LastEmbedding model in Table 6.1.

Interestingly, we find that the embedding dropout is only effective on ML-1M, and it barely affects performance of all other datasets. To be fair, this is likely because the default value for the embedding dimension that we are using in these experiments is 256, which is likely too large for a small dataset like ML-1M. However, this result does show the effectiveness of embedding dropout to mitigate this overfitting.

Bias Secondly, we can add a simple bias. A bias increases the expressiveness of a model by learning a predisposition towards items that might appear in various contexts that can not be sufficiently captured by embeddings alone. The current model is specified by Equation 6.3. Note that the bias b is shared across positions, so we define R_t instead of R as a whole. The results of this model can be found in Table 6.4.

$$R_t = E^D_{s_t} E^T + b \tag{6.3}$$

Dataset	HR	NDCG	CatCov	Ser	Nov
Beauty	.058 (3.6%)	.035 (2.9%)	.660	.054	13.7
ML-1M	.200 (5.8%)	.113 (5.6%)	.635	.195	10.5
DH GR	.145 (1.4%)	.084 (0.0%)	.673	.121	10.1
DH SI	.129 (0.0%)	.073 (0.0%)	.302	.097	10.0

TABLE 6.4: Evaluation results of a bias on the LastEmbedding model. The percentages indicate the performance difference with the LastEmbedding model in Table 6.3.

We find that the bias slightly improves performance on Beauty and DH Greece. Still, the effect is very limited. The main performance gains are made on ML-1M, where we find an increase in accuracy metrics of roughly 5%. This indicates that the predecessor of an item is not a sufficient source of information alone, and instead we need a popularity bias to start converging towards the performance of the models in Table 4.6. The beyond-accuracy metrics also highlight the effect of the bias on our proprietary datasets. While performance increases slightly in terms of HitRate, we sacrifice catalog coverage, serendipity and novelty. This is the expected behaviour of the bias of course, but it highlights how seemingly minor architectural additions can affect the behaviour of the model significantly.

Linear transformation Recalling our discussion in subsection 4.3.4, we originally decided to adopt the prediction network of BERT4Rec to allow all three models to have the same prediction network. Since the prediction network is position-independent, we can analyze its effect on our LastEmbedding without having to include non-last items. Since we feed embeddings directly into the prediction network, a non-linear activation would only reduce the representative capacity of the embedding. Therefore, we only apply the linear transformation W of the prediction network.

$$R = E^D_s W E^T + b \tag{6.4}$$

Dataset	HR	NDCG	CatCov	Ser	Nov
Beauty	.041 (-29.3%)	.024 (-31.4%)	.373	.037	12.8
ML-1M	.212 (6.0%)	.122 (8.0%)	.522	.206	10.4
DH GR	.142 (-2.1%)	.084 (0.0%)	.736	.120	10.3
DH SI	.117 (-9.3%)	.066 (-9.6%)	.351	.086	10.2

TABLE 6.5: Evaluation results of a linear transformation on the LastEmbedding model. The percentages indicate the performance difference with the LastEmbedding model in Table 6.4.

Interestingly, we find that using a linear transformation mostly harms performance. The performance on the sparse datasets is significantly worse, and the only dataset where performance improves is ML-1M. We can partly attribute this finding to the fact that the dense layer introduces asymmetry back into our model, as $e_i W e_j^T \neq e_j W e_i^T$ unless W is symmetric. As such, the performance of LastEmbedding with a prediction network should be correlated with the performance of LastEmbedding with heterogeneity (See Table 6.1). When we enforce W to be symmetric we find that we mitigate the performance degradation on Beauty and DH Singapore, but any linear transformation still reduces performance in general. All three of our vanilla models enforce that the embeddings are passed through some linear transformation, which implies we might be able to increase or maintain performance while removing such transformations. Moreover, our finding empirically supports the motivation for CORE [41] that the session embedding should fall into the space spanned by the item embeddings. We will further discuss the relatedness of our work to CORE in section 6.2.

At this point we observe a trend where the additional components introduced in the original publications only seem to enhance accuracy on ML-1M, whereas they do not affect or even harm performance on other datasets. Given that the linear transformation generally harms performance, we do not use this variant in upcoming experiments.

6.1.6 Training improvements

Having exhausted different architectural additions without much effect, we aim to improve the training task of the model. A shortcoming of our current model is that items only directly learn from their direct successors and predecessors in the sessions. However, we hypothesize that items could still learn from other items in the session, especially in the case of sparse datasets where there will only be a few sessions where the item occurs. To illustrate why this might improve performance of LastEmbedding, we found that NextPopular can be enhanced by not only counting the successors and predecessors of an item, but also the super-successor. This results in a model we denote with NextPop (3) whose results are summarized in Table 6.6.

Dataset	HR	NDCG	CatCov	Ser	Nov
Beauty	.045 (9.8%)	.029 (11.5%)	.671	.043	13.4
ML-1M	.207 (-1.4%)	.121 (1.7%)	.701	.202	10.5
DH GR	.142 (1.4%)	.084 (2.4%)	.784	.120	10.3
DH SI	.120 (5.3%)	.071 (6.0%)	.342	.094	10.5

TABLE 6.6: Evaluation results of NextPop (3). It indicates that counting other items than only the direct predecessor and successor can be beneficial to the performance of a first-order MDP. The percentages indicate the performance difference with the homogeneous NextPopular model in Table 6.1.

Hence, the results in Table 6.6 show that models might benefit from learning associations between non-neighbouring items. Currently, give a session (s_1, s_2, s_3), our LastEmbedding model draws the embeddings of item s_1 and s_2 closer and draws the embeddings of items s_2 and s_3 closer. Therefore, it may very well be the case that the model already sufficiently learns the connection between items s_1 and s_3 through s_2. Still, we aim to explore whether a more direct coupling between s_1 with s_3 will enhance performance.

Background Petrov and Macdonald [72] have already explored this idea to some degree by training BERT4Rec and SASRec with LambdaRank [6]. LambdaRank is a loss function that includes multiple items as targets in its loss computation. As a result, the model is stimulated to draw the item representations closer. Promisingly, Petrov and Macdonald [72] found that LambdaRank resulted in significant performance gains for the transformer models. Our motivation is also similar to the one presented for DropoutNet [94], which encourages the model to use side-information for unpopular items while disregarding side-information for popular items through a similar dropout mechanism.

Token dropout Similar to DropoutNet [94], we add token dropout before we feed the session to the model. In other words, we drop an item from a session with a fixed probability. For example, if we drop s_2 from (s_1, s_2, s_3), we feed (s_1, s_3) to the model. This would allow the model to learn the association between s_1 and s_3 with our current next-item training task and standard cross-entropy loss. Beside the fact that DropoutNet is applied to the matrix-completion recommendation task, the conceptual difference is that we want to use token dropout to encourage our model to use other items in the session to learn better representations of unpopular items. The results of LastEmbedding with token dropout (t) are summarized in Table 6.7 below.

Dataset	HR	NDCG	CatCov	Ser	Nov
Beauty	.059 (1.7%)	.036 (2.9%)	.556	.056	13.4
ML-1M	.147 (-26.5%)	.083 (-26.4%)	.699	.141	10.8
DH GR	.144 (-0.7%)	.084 (0.0%)	.779	.123	10.4
DH SI	.131 (1.6%)	.074 (1.4%)	.359	.102	10.5

TABLE 6.7: Evaluation results of token dropout on the LastEmbedding model. For these results we used a dropout probability of 0.3.

Interestingly, we only slightly improve performance on Beauty and DH Singapore, which are the datasets with the highest share of low-popularity items. The

fact that it harms accuracy on DH Greece or ML-1M indicates that associations with other items from the session are superfluous, or even noisy. We note that we did find a slight performance increase on the lowest-popularity last-items, but apparently this does not outweigh the performance decrease on the popular items. As a result, some ensemble would be possible where you use token dropout only on the lowest popularity items, but on higher popularities (after a 10 interactions already), it is better to use the models trained without token dropout.

In short, our datasets are too dense to significantly benefit from token dropout, but our results indicate that the technique would be effective on sparser datasets. Furthermore, this negative result indicates that we have optimized the quality of the item embeddings already, given that additional signals (between distant items in the session) did not improve performance.

6.1.7 Discussion

In this section we have explored to what degree we can exploit the last item in a session as the only source of information to predict the next-item. The degree to which we can approach the performance of the models in Table 4.6 with just embeddings depends on the dataset. We outperform the vanilla neural models on Beauty, come close to their performance on DH Greece, but have significantly worse performance on ML-1M and DH Singapore. To some degree this is to be expected, as Figure 5.10 indicates that other positions than the last are also indicative of the next-item. Architectural components used in the vanilla models other than the embeddings do not seem to add any performance, except on the ML-1M dataset. This illustrates that the embeddings themselves are already expressive enough to capture most of the patterns in the sessions.

Lastly, in subsection 6.1.6 we have seen that using other items than the direct successors to train an item's embedding is only effective on extremely low-popularity items, which are quite rare in our dataset suite. Therefore, we believe that through the current combination of training task and homogeneous embedding architecture, we have approached the maximum performance that can be reached with the last item alone.

6.2 Exploiting non-last items

We now continue with exploiting other items than the last. In chapter 5 we found that most of the performance of the models can be attributed to the last item, but every model still improves when we include older items. Furthermore, ItemKNN proves to be an extremely competitive baseline while disregarding the ordering of the items in the session. Hence, the next obvious step in constructing our model is by exploiting non-last items. The way that a model includes other items in the session is the unique differentiator, where GRU4Rec uses the GRU layer, SASRec uses a unidirectional transformer and BERT4Rec uses a bidirectional transformer. In this section, we will gradually converge to both types of layers to uncover what precisely enables their superior performance on the remaining datasets.

6.2.1 Positional weights

Our first step in moving towards the full architecture of the neural models will be the inclusion of other items in the session through weighted summation. Our motivation for this is our finding that the trivial recommendations from other positions

already appear to be good recommendations in Figure 5.11. This means that we do not have to take interaction effects between the items into account, and that we can instead rely on something simple like weighted summation to exploit the non-last items. More specifically, we want to define a matrix W of shape (N, N) so that the current model is defined by Equation 6.5. Clearly, the session embedding WE_s is a weighted summation of the embeddings. We constrain W so that the superdiagonal is zero ($\forall_{t \in \{1...N\}} W_{t,t+1} = 0$) because item s_{t+1} is the target for R_t in our training task.

$$R = (WE_s)^D E^T + b \tag{6.5}$$

When we take $W = \mathbf{1}$ with zeros on the superdiagonal, we disregard ordering and simply use the sum of the embeddings of the items in the session to predict the next-item. This is the architectural design of CORE-ave in [41], which was initially shown to be outperformed by the vanilla models. However, they train their model by only computing the loss on R_N instead of all timesteps [2]. Instead, we will train our variant where we train on all timesteps in order to maximize data usage.

Dataset	HR	↓NDCG	CatCov	Ser Nov	
Beauty	.055 (-5.2%)	.031 (-11.4%)	.392	.049	12.2
ML-1M	.067 (-66.5%)	.018 (-84.1%)	.277	.027	9.1
DH GR	.097 (-33.1%)	.052 (-38.0%)	.561	.063	9.1
DH SI	0.154 (19.4%)	.086 (17.8%)	.241	.115	9.5

TABLE 6.8: Evaluation results of the model defined by Equation 6.5 where $W = \mathbf{1}$. The percentages indicate the performance difference with the LastEmbedding model in Table 6.4.

Clearly, the results are worse than our results of the LastEmbedding model of Table 6.4 on Beauty, ML-1M and DH Greece. Of course, from Figure 5.2 we could have already concluded that simply taking the average embedding would be too simple as it completely disregards ordering like ItemKNN. We note that the minimum loss by this model is not lower than the loss of the vanilla neural models, indicating that overfitting is not the cause of the low performance. Instead, we believe that the summation of the item embeddings is too noisy for the model to differentiate the relevant (usually more recent) items. The exception to these observations is DH Singapore, where the model seems to improve performance over the LastEmbedding model in Table 6.4. This is to be expected, as ItemKNN is the most performant on this dataset in Table 4.6. Still, the model in Table 6.8 does not appear to surpass ItemKNN in terms of accuracy. This means that the effect of the noise in the tail of the session still limits the model to produce good recommendations.

Learning W through backpropagation Therefore, we propose to learn W through backpropagation. This means that for each timestep t, we have a learnable vector of weights W_t where $W_{t,t+1} = 0$, but the weights for timesteps $t' > t+1$ are not constrained. As a result, Equation 6.5 constitutes a *bidirectional* model. The results are summarized in Table 6.9. Similarly, we train a *unidirectional* model. Recall that the definition of a unidirectional model is that the predictions R_t are only dependent on t and previous timesteps. Hence, we constrain W so that the upper diagonal are all zeros. The results of the unidirectional model are summarized in Table 6.10.

[2] In the design of CORE-ave [41], the superdiagonal of W is not set to zeros, so that training on timesteps other than N is not possible due to information leakage.

Dataset	HR	↓NDCG	CatCov	Ser	Nov
Beauty	.067 (15.5%)	.040 (14.3%)	.649	.062	13.2
ML-1M	.239 (19.5%)	.130 (15.0%)	.684	.233	10.5
DH GR	.145 (0.0%)	.084 (0.0%)	.742	.116	9.9
DH SI	.167 (29.5%)	.093 (27.3%)	.288	.132	10.1

TABLE 6.9: Evaluation results of the model defined by Equation 6.5 where W is bidirectional and is learned through backpropagation. The percentages indicate the performance difference with the LastEmbedding model in Table 6.4.

Dataset	HR	↓NDCG	CatCov	Ser	Nov
Beauty	.067 (15.5%)	.040 (14.3%)	.745	.062	13.5
ML-1M	.257 (28.5%)	.142 (25.7%)	.696	.250	10.5
DH GR	.146 (0.7%)	.085 (1.2%)	.718	.119	9.9
DH SI	.170 (31.7%)	.095 (30.1%)	.304	.136	10.2

TABLE 6.10: Evaluation results of the model defined by Equation 6.5 where W is unidirectional and is learned through backpropagation. The percentages indicate the performance difference with the LastEmbedding model in Table 6.4.

General observations We find that the performance of the unidirectional model in Table 6.10 is either equal to or better than the bidirectional model in Table 6.9. This supports the finding in [73] that the bidirectionality of BERT4Rec is not the underlying cause of its superiority over unidirectional SASRec. At this point, we find that the unidirectional model outperforms all models on Beauty and outperforms SASRec and GRU4Rec on DH Singapore. Allthewhile, our model is significantly simpler to tune and implement. Moreover, it almost equals BERT4Rec in terms of performance on DH Singapore. We could already partially explain that BERT4Rec is better than the unidirectional models on Singapore because of its bidirectionality in subsubsection 5.2, but from these results we find that a unidirectional model is practically able to attain similar performance to BERT4Rec and ItemKNN. We can therefore conclude that both SASRec and GRU4Rec are overparameterized on DH Singapore, because this simple model outperforms them significantly while learning under the same unidirectionality constraint. For simplicity we will continue our experiments with the unidirectional model.

Masking process Also, note that we used a bidirectional model in Table 6.9 does not make use of the masking process from BERT4Rec. This is not necessary because the constraint of having W with a superdiagonal with all zeros is sufficient to prevent information leakage. In fact, this would also hold for BERT4Rec with a single transformer layer. As a result, we might be able to significantly reduce training time by removing the random train cases all together, and instead just train a single-layer BERT4Rec with the same training task of SASRec, namely predicting the item s_{t+1} at timestep t. We leave this experiment for future work.

On the other hand, the fact that BERT4Rec significantly outperforms SASRec on Beauty and DH Singapore can now only be explained by the masking process. We eliminated all differences between SASRec and BERT4Rec in chapter 4, and because the bidirectionality has been shown to not improve performance, it must be the masking process that allows BERT4Rec to surpass SASRec in Table 4.6.

Learned weights Naturally, we are interested in the final value of W. Therefore, we visualize the learned weights of the last 10 positions that are used for the next-item prediction in Figure 6.3. In other words, we visualize the values in the last 10 columns of the last row of W.

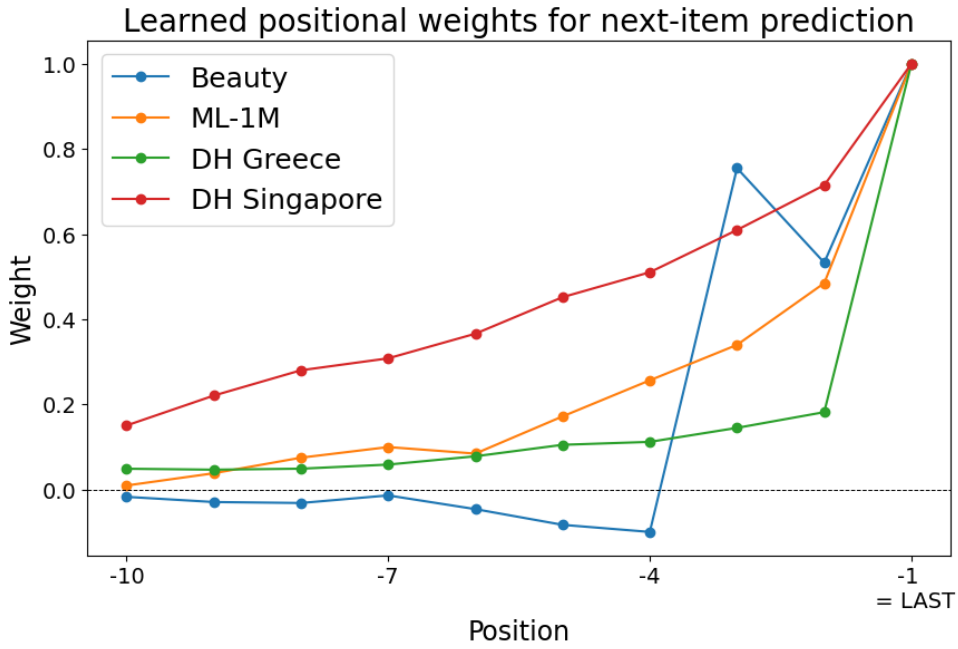

FIGURE 6.3: The learned positional weights for the next-item prediction by the model defined by Equation 6.5 where W is unidirectional and is learned through backpropagation. Essentially, we visualize the values in the last 10 columns of the last row of W.

In general, we find that W roughly learns the weights proportional to the importance of each position in Figure 5.11. Surprisingly, we find that the model on Beauty learns negative weights, meaning that the model actually subtracts scores for items related to the historical item interactions in the session.

6.2.2 Constrained positional weights

We do find some instability in the learned positional weights, even on our densest DH Greece dataset. More specifically, we have that the values of W between two trained model instances may differ. We believe this is due to the vulnerability to the initialization of the item embeddings matrix. If an item s_t and an item $s_{t'}$ coincidentally have a high dot-product similarity at initialization time, then the model will learn to increase $W_{t,t'-1}$ in order to exploit item s_t to predict item $s_{t'}$. Moreover, on Beauty we can even see that the third-to-last position is given more weight than the second-to-last, which is a clear sign of overfitting. Therefore, we explore whether we can address the instability and overfitting by incorporating several assumptions. Our first assumption is that the weight of a timestep t on the predictions for another timestep t' should only be based on the distance $t - t'$ and not the specific values of t and t'. Our second assumption is that this weight should decrease monotonically with the distance $t - t'$. Therefore, instead of learning the positional weights matrix W directly, we learn parameter α. We use this parameter to define $W^{(\alpha)}$ by Equation 6.6. We will refer to the model defined by Equation 6.5 and Equation 6.6 with $W^{(\alpha)}$.

$$W^{(\alpha)}_{t,t'} = \frac{1}{(t-t')^{\alpha}} \mathbf{1}(t < t') \tag{6.6}$$

Evidently, the weights are now only dependent on the distance between two items, and the weights monotonically decrease with the distance if $\alpha > 0$. The model can control the rate at which this weight decreases with α. Unfortunately, this formula does not allow for negative weights like we have seen on the Beauty dataset. The formula was chosen by consulting Figure 5.11. We found that it is important to initialize α with a high value (e.g. 10) so that W is close to an identity matrix at the start of training. As a result, the focus in the first epoch of training is on learning good embeddings before their weighted summations are taken in later epochs.

The results of the model are summarized in Table 6.11.

Dataset	HR	↓NDCG	CatCov	Ser	Nov
Beauty	.073 (9.0%)	.043 (7.5%)	.561	.068	12.9
ML-1M	.258 (7.9%)	.141 (8.4%)	.690	.251	10.5
DH GR	.147 (1.4%)	.086 (2.4%)	.708	.122	10.1
DH SI	.170 (1.8%)	.095 (2.2%)	.304	.136	10.2

TABLE 6.11: Evaluation results of the model defined by Equation 6.5 where W is defined by Equation 6.6. α is learned through backpropagation. We call this model $W^{(\alpha)}$. The percentages indicate the performance difference with the model in Table 6.10.

Surprisingly, the performance on Beauty rises considerably. It illustrates how constraining the expressiveness and parameterization of a model can sometimes lead to better results.

Dataset	Learned α
Beauty	0.64
ML-1M	1.21
DH GR	1.50
DH SI	0.40

TABLE 6.12: The learned values for α from the results in Table 6.11 by $W^{(\alpha)}$. Higher values indicate a higher importance for the last item(s).

In general the values for α in Table 6.12 coincide with the curve found in Figure 5.11. Table 6.12 also highlights the peculiarity of the ML-1M dataset, where the last item is apparently extremely important, but our vanilla models need multiple items to attain their overall NDCG@10. We leave the investigation of other definitions for the weight matrix like in Equation 6.6 for future work.

Inference time and approximation As a side note, we would like to recall our discussion in section 6.1 that the LastEmbedding model was particularly scalable because we could precompute the recommendations for each item. Fortunately, this is also possible for the models governed by Equation 6.5 because matrix multiplications are associative. More specifically, we can precompute EE^T, which is a $(|\mathcal{I}|, |\mathcal{I}|)$ matrix containing the score that each potential item in a session (row) would give to a potential item to be recommended (column). While this would cost a significant amount of memory, from preliminary experiments we found that for each item

we can simply precompute the top-50 items with the highest scores. This only demands linear memory in terms of the number of items. At inference time we can then take the top-50 items of each item in the session, and combine the confidence for each item according to W. This requires much less compute (no GPU) than the vanilla neural models. This approximation works because apparently, items that are not in the top-50 of any item in the session rarely end up in the top-K recommendation slate anyway. Of course, you can vary 50 to be any number to balance between scalability and the probability that you will miss an item that would end up in the recommendation slate.

6.2.3 Layer normalization

Having explored the potential of a simple weighted summation of item embeddings, we continue by exploring how we can further improve the performance of the $W^{(\alpha)}$ model with various architectural components from BERT4Rec [87], SASRec [47] and GRU4Rec [40]. For brevity, we include our most successful experiment which is the addition of layer normalization to the session embedding. Formally, we modify $W^{(\alpha)}$ with Equation 6.7, where superscript D denotes a dropout layer and superscript N denotes a normalization layer [4]. The layer normalization ensures that the the session embedding $(WE_s)^{DN}$ has a zero mean and unit variance. We summarize the results in Table 6.13. We will provide an overview of all model evaluation results later in section 6.3.

$$R = (W^{(\alpha)} E_s)^{DN} E^T + b \tag{6.7}$$

Dataset	HR	↓NDCG	CatCov	Ser	Nov
Beauty	.053 (-15.8%)	.032 (-25.6%)	.727	.049	13.8
ML-1M	.203 (-21.3%)	.116 (-17.8%)	.665	.198	10.5
DH GR	.149 (1.4%)	.088 (2.3%)	.794	.130	10.7
DH SI	.184 (8.2%)	.104 (9.5%)	.327	.153	10.7

TABLE 6.13: Evaluation results of $W^{(\alpha)}$ with layer normalization. The percentages indicate the performance difference with the $W^{(\alpha)}$ model in Table 6.11.

Interestingly, we find that we further improve the performance of the $W^{(\alpha)}$ model on the Delivery Hero datasets, whereas we significantly decrease accuracy on the Beauty and ML-1M datasets. We believe the layer normalization to be an effective mechanism to improve performance because it allows the model to capture interaction effects between items. To clarify, the computation of the variance on the weighted summation of the item embeddings involves cross-terms between the individual item embeddings, effectively allowing the model to take interaction effects into account. At the same time, the normalization layer does remove the opportunity to precompute results, since the normalization is not an associative operation. We believe interaction effects between items to be more relevant on sessions with a short timespan (shopping carts) in contrast sessions with a longer timespan (reviews/ratings), which could explain why performance increases on the Delivery Hero datasets, but decreases on the Beauty and ML-1M datasets.

6.3 Discussion

In this chapter we have explored the possibility of simplifying the neural models while maintaining or improving performance. To ease discussion we combine all results from chapter 4 and this chapter in Table 6.14. We include our LastEmbedding model, our $W^{(\alpha)}$ model, and $W^{(\alpha)}$ with layer normalization ($W^{(\alpha)}$ LN). For the LastEmbedding model, we pick the best result from the version with and the version without token dropout.

6.3.1 Hyperparameter resilience

We note that we optimized the hyperparameters for LastEmbedding, $W^{(\alpha)}$, and $W^{(\alpha)}$ LN, but found that the models are generally more resilient towards the hyperparameters than the vanilla neural models. The only hyperparameter that seemed to affect the results was the embedding dimension e. This is to be expected, as this parameter directly controls the model size and capacity for learning item associations. However, all other parameters seemed to have very little effect on the accuracy metrics. This is a very desirable model trait, as hyperparameter optimization is generally expensive. This is demonstrated by the fact that the original publications on BERT4Rec and SASRec only hypersearched 4 and 2 hyperparameters respectively, in contrast to the 8 common hyperparameters we identified in Table 4.1.

6.3.2 Accuracy results

Surprisingly, we found that LastEmbedding can already be considered a strong baseline on various datasets while only relying on the last item of each session. Moreover, the results show that $W^{(\alpha)}$ and its variant with layer normalization are extremely competitive models on 3 out of 4 datasets. On our sparsest datasets, Beauty and DH Singapore, we significantly outperform the vanilla neural models and the non-neural baselines with up to 16% in NDCG@10 on Beauty, and 4% on DH Singapore. On DH Greece we approach the top NDCG@10 with only a 2% difference. In addition, the recommendations of $W^{(\alpha)}$ without layer normalization can be precomputed, making it a highly scalable and fast model. To recall our discussion on trivial recommendation overlap in subsection 5.5.2, the $W^{(\alpha)}$ model shows that exploiting the trivial recommendations from various positions in the session can already provide significant performance, because the $W^{(\alpha)}$ model essentially combines trivial recommendations from each item weighted by its position. Furthermore, we observed that layer normalization increases performance on the Delivery Hero datasets, which we attribute to the fact that layer normalization allows for the modelling of interactions effects between items.

6.3.3 Beyond-acuracy results

Finally, from Table 6.14 we also observe that our embedding-based models generally have more desirable beyond-accuracy metrics than the vanilla neural models. For example, on Beauty we reach more than double the catalog coverage with $W^{(\alpha)}$ compared to GRU4Rec, BERT and SASRec. On the Delivery Hero datasets, $W^{(\alpha)}$ and $W^{(\alpha)}$ LN reach higher values for novelty, meaning that these models are less prone to the popularity bias discussed in section 5.3. While we do not surpass the neural models on DH Greece, we do find that that $W^{(\alpha)}$ LN has a higher value for serendipity, meaning it correctly predicted more ground-truth items that are not part of the top-10 of most popular items.

6.3.4 Limitations

One of the goals of this chapter was to evaluate whether the vanilla neural models were over-parameterized for the session recommendation task. While we concluded that it depends on the dataset and its characteristics, the main limitation of this chapter is that there are still a large number of architectural components between our $W^{(\alpha)}$ model and the fully-parameterized vanilla neural models. For the sake of brevity, we excluded a broad set of preliminary results on several of these missing architectural components, but we generally found that the accuracies attained by these intermediate models interpolate the accuracy of the $W^{(\alpha)}$ and fully-parameterized models. For example, a model where the positional weights in $W^{(\alpha)}$ are replaced with an attention mechanism seems to slightly surpass $W^{(\alpha)}$ model on DH Greece, but it also degrades in performance on Beauty.

Dataset	Model	HR	↓NDCG	CatCov	Ser	Nov
Beauty	$W^{(\alpha)}$	.073	.043	.561	.068	12.9
	SKNN	.064	.037	.540	.058	12.6
	ItemKNN	.052	.030	.741	.048	13.9
	LastEmbedding	.059	.036	.556	.056	13.4
	$W^{(\alpha)}$ LN	.053	.032	.727	.049	13.8
	GRU4Rec	.054	.030	.231	.048	12.9
	BERT	.047	.026	.226	.043	13.2
	NextPop	.041	.026	.664	.039	13.5
	SASRec	.039	.021	.106	.033	12.1
	Popular	.012	.005	.000	.000	9.7
ML-1M	GRU4Rec	.339	.203	.816	.331	10.7
	SASRec	.307	.176	.677	.300	10.7
	BERT	.309	.171	.633	.303	10.6
	$W^{(\alpha)}$	.258	.141	.690	.251	10.5
	$W^{(\alpha)}$ LN	.203	.116	.665	.198	10.5
	NextPop	.200	.114	.723	.194	10.5
	LastEmbedding	.200	.113	.635	.195	10.5
	SKNN	.154	.077	.502	.148	9.8
	ItemKNN	.053	.026	.060	.038	8.7
	Popular	.016	.008	.003	.000	8.5
DH GR	GRU4Rec	.155	.090	.443	.128	10.1
	SASRec	.153	.090	.486	.128	10.3
	$W^{(\alpha)}$ LN	.149	.088	.794	.130	10.7
	BERT	.150	.086	.375	.121	9.9
	$W^{(\alpha)}$	.147	.086	.708	.122	10.1
	LastEmbedding	.145	.084	.673	.121	10.1
	NextPop	.140	.083	.796	.120	10.5
	SKNN	.131	.077	.785	.112	10.5
	ItemKNN	.117	.065	.697	.083	9.4
	Popular	.052	.024	.001	.000	7.5
DH SI	$W^{(\alpha)}$ LN	.184	.104	.327	.153	10.7
	ItemKNN	.175	.100	.362	.136	10.0
	BERT	.171	.097	.125	.139	10.4
	$W^{(\alpha)}$	.170	.095	.304	.136	10.2
	SKNN	.159	.094	.361	.131	10.4
	GRU4Rec	.156	.089	.375	.129	11.0
	SASRec	.158	.088	.292	.128	10.7
	LastEmbedding	.120	.071	.342	.094	10.5
	NextPop	.114	.067	.335	.089	10.6
	Popular	.049	.024	.000	.000	7.5

TABLE 6.14: The final evaluation results of our vanilla neural models, the non-neural baselines and our simplified embedding models. The table is sorted by NDCG@10.

Chapter 7

Conclusion

In this thesis we have explored the performance and behaviour of the popular neural session recommendation models on several open and several real world session datasets. We will explicitly summarize our answer to each research question introduced in chapter 1.

Research question 1: What are the methodological errors made in the design and evaluation of the models, and how can we create an evaluation setup that does not suffer from these issues? We identified and addressed several problems related to the original publications in the literature. Most importantly, we replaced the sampled metrics evaluation task with a full ranking evaluation task. Furthermore, we do not use the test set during training, and instead design an early-stopping mechanism in order to determine when to stop training. Lastly, we do not fix any hyperparameters during training, and instead rely on the TPE sampler to find the optimal configuration in a freely-defined hyperparameter search space.

Research question 2: What models are most effective for sequential recommendation on our datasets? Having addressed these issues, we aimed to uncover the potential of each architecture by standardizing auxiliary design decisions like the batch design and loss function. We found that this standardization of auxiliary design decisions significantly improves the performance of the models, and subsequently causes the models to be more similar in performance than previously reported. Moreover, the precise model ranking depends on the dataset, and no clearly superior model can be determined in general. Instead, we found that a model's relative performance depends on the dataset's predisposition towards the unique properties of that model. For example, we showed that BERT4Rec's bidirectionality enables it to mimic ItemKNN better, which in turn can explain some of the performance gains of BERT4Rec in comparison to SASRec and GRU4Rec on DH Singapore. In addition, we also found that the non-neural baselines can be extremely competitive, and outperform our vanilla neural models on 2 of our 4 datasets. Interestingly, we also identified a significant difference in model performance and behaviour between the Delivery Hero Greece and Delivery Hero Singapore dataset, indicating that minor details in the data collection phase might already significantly affect a model's ability to find suitable recommendations.

Research question 3: How do these models' recommendations compare on various dimensions? We then analyzed the recommendation behaviour of the models on various dimensions, including the recommendation slate size, the session length, item popularity and overlap with trivial models. We find that the models are surprisingly similar and sometimes even predictable. For example, we find that the non-determinism in the random batching and random initialization causes a pair of

two instances of the same model to be almost indiscriminable from any other pair of models in terms of recommendation overlap. Moreover, we also find that different models agree most on the top-ranked items, and start to differ more when the size of the recommendation slate, K, grows. While on the open datasets the models seem to exhibit desired behaviour in terms of recommending items from different popularities, we find that on the proprietary datasets all three models overconfidently recommend popular items. We have a diverse set of results on session lengths, and it appears that on some datasets the session length plays little role in the performance of the model. Instead, it seems to be correlated with the last item's relatedness to the next-item. Still, the models remain very similar in their performance on different session lengths, which opposes the general belief that GRU4Rec is more suitable for short-term preference modelling, while the transformer models are more suitable for long-term preference modelling. In general, the neural models exhibit fluidity in their behaviour, and converge towards either a first-order Markov Decision Process or ItemKNN depending on what works better for the dataset. All in all, the models' recommendations are extremely similar and almost indistinguishable on the various dimensions explored.

Research question 4: Given these findings, can we devise simpler models while maintaining or improving performance? Given that the models have very different architectures, we hypothesized that there must be a core architectural component causing the inherent similarity, which in turn might imply that the full architectures are over-parameterized. Indeed, we find that most of the performance on all four datasets can be attributed to the item embeddings, which in turn explains the similarity between the neural models. In fact, our LastEmbedding model, which recommends the items that are most similar to the last item, already significantly outperforms the neural models on the Beauty dataset, confirming that the models are indeed over-parameterized on that dataset. Other architectural components like a bias, dropout, or transformations had a limited positive or severe negative effect on the performance of the models.

We then exploited our observation that the trivial recommendations from non-last items are already a fruitful source of recommendations, which means we do not need to model any interaction effects between the items. Therefore, we continue by exploring the potential of a weighted summation between the item embeddings, at which point our model attains better or similar performance on 3 out of 4 datasets. Allthewhile, our model remains pre-computable, which means that our model is fast and scalable with no need for a GPU in production. Lastly, we further increased performance on the Delivery Hero datasets by modelling the interaction effects between items through a normalization layer.

Chapter 8

Discussion

Having answered our research questions, this chapter is intended to address several limitations of this thesis and comment on the work in general.

Focus of this thesis First of all, this thesis was mainly concerned with the neural architectures. The main result on this is that there is very little difference between the performance and behaviour of the models once all auxiliary components like the training task and loss function are standardized. It seems that the models converge to the same behaviour as long as their architecture is sufficiently expressive to process all items in the session. As such, we personally believe that the future discoveries in neural session recommendation domain will not be new and more complex architectures, and instead will be better loss functions and training tasks. Admittedly, we have neglected these components in favour of a more in-depth analysis of the architecture.

While we advocate for a more holistic approach to recommendation in general, this thesis also just focuses on optimizing offline metrics like the bulk of the research on recommendation. However, there is very little research on whether the offline evaluation metrics actually translate to higher business metrics like user satisfaction [65]. Frankly, we have optimized a model's ability to predict the last item of a shopping cart, but this is not straightforwardly correlated with a model's ability to recommend good items that will increase profit. Due to internal delays we were unfortunately not able to properly analyze the online performance of our algorithms as well, and instead decided to focus on optimizing the offline evaluation metrics as a proxy. We did move away from the typical conduct of simply introducing a model and providing an evaluation by also conducting an analysis in chapter 5. This lead to multiple promising avenues for optimizing the models explored in chapter 6, highlighting that a holistic approach can be fruitful.

Early decision designs Besides the training task and loss function, there are much more design decisions that may have affected our results. We navigated the space of possible implementations and hyperparameters through preliminary experiments, which is by no means a rigorous process. Due to the sheer number of decisions related to the design and implementation of a deep learning model, it was difficult to balance between maintaining rigor and maintaining a decent scope for this thesis. For example, all BERT4Rec experiments in this thesis use 10 as the number of random train cases. Clearly, this might not be optimal, but it is also extremely hard to re-evaluate (re-optimize) this decision at every single design step. Ironically, the initial motivation for our research was to address the exact same issues in the original publications, but we found that this would simply demand too much compute and meticulosity.

Dataset suite Lastly, we would like to note that our dataset suite is limited as well. We chose Beauty [69] and ML-1M [31] as our open datasets because of their widespread use and diverse dataset statistics. Still, these datasets originally consist of reviews and ratings instead of interactions, and so they only partially represent the true session recommendation scenario. Fortunately, the Delivery Hero datasets provide us with real session datasets. Still, shopping carts are very particular types of sessions, so that our results on these datasets might not be directy transferrable to use-cases with other types of sessions.

Bibliography

[1] Charu C. Aggarwal. *Recommender Systems: The Textbook*. 1st. Springer Publishing Company, Incorporated, 2016. ISBN: 3319296574.

[2] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, 214–223.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[5] James Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Granada, Spain: Curran Associates Inc., 2011, 2546–2554. ISBN: 9781618395993.

[6] Christopher JC Burges. "From ranknet to lambdarank to lambdamart: An overview". In: *Microsoft Research Technical Report* 11.23-581 (2010), p. 81.

[7] Rocío Cañamares and Pablo Castells. "On Target Item Sampling In Offline Recommender System Evaluation". In: *Proceedings of the 14th ACM Conference on Recommender Systems*. RecSys '20. Virtual Event, Brazil: Association for Computing Machinery, 2020, 259–268. ISBN: 9781450375832. DOI: 10.1145/3383313.3412259. URL: https://doi.org/10.1145/3383313.3412259.

[8] Huiyuan Chen et al. "Denoising Self-Attentive Sequential Recommendation". In: *Proceedings of the 16th ACM Conference on Recommender Systems*. RecSys '22. Seattle, WA, USA: Association for Computing Machinery, 2022, 92–101. ISBN: 9781450392785. DOI: 10.1145/3523227.3546788. URL: https://doi.org/10.1145/3523227.3546788.

[9] Qiwei Chen et al. "Behavior Sequence Transformer for E-Commerce Recommendation in Alibaba". In: *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*. DLP-KDD '19. Anchorage, Alaska: Association for Computing Machinery, 2019. ISBN: 9781450367837. DOI: 10.1145/3326937.3341261. URL: https://doi.org/10.1145/3326937.3341261.

[10] Yongjun Chen, Jia Li, and Caiming Xiong. "ELECRec: Training Sequential Recommenders as Discriminators". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '22. Madrid, Spain: Association for Computing Machinery, 2022, 2550–2554. ISBN: 9781450387323. DOI: 10.1145/3477495.3531894. URL: https://doi.org/10.1145/3477495.3531894.

[11] Heng-Tze Cheng et al. "Wide & Deep Learning for Recommender Systems". In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. DLRS 2016. Boston, MA, USA: Association for Computing Machinery, 2016, 7–10. ISBN: 9781450347952. DOI: 10.1145/2988450.2988454. URL: https://doi.org/10.1145/2988450.2988454.

[12] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: https://aclanthology.org/D14-1179.

[13] Philippe Clement and Wolfgang Desch. "An elementary proof of the triangle inequality for the Wasserstein metric". In: *Proceedings of The American Mathematical Society - PROC AMER MATH SOC* 136 (Jan. 2008), pp. 333–340. DOI: 10.1090/S0002-9939-07-09020-X.

[14] Paul Covington, Jay Adams, and Emre Sargin. "Deep Neural Networks for YouTube Recommendations". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, 191–198. ISBN: 9781450340359. DOI: 10.1145/2959100.2959190. URL: https://doi.org/10.1145/2959100.2959190.

[15] Alexander Dallmann, Daniel Zoller, and Andreas Hotho. "A Case Study on Sampling Strategies for Evaluating Neural Sequential Item Recommendation Models". In: *Proceedings of the 15th ACM Conference on Recommender Systems*. RecSys '21. Amsterdam, Netherlands: Association for Computing Machinery, 2021, 505–514. ISBN: 9781450384582. DOI: 10.1145/3460231.3475943. URL: https://doi.org/10.1145/3460231.3475943.

[16] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[17] Yimin Ding. "The Impact of Learning Rate Decay and Periodical Learning Rate Restart on Artificial Neural Network". In: *Proceedings of the 2021 2nd International Conference on Artificial Intelligence in Electronics Engineering*. AIEE '21. Phuket, Thailand: Association for Computing Machinery, 2021, 6–14. ISBN: 9781450389273. DOI: 10.1145/3460268.3460270. URL: https://doi.org/10.1145/3460268.3460270.

[18] Hanwen Du et al. "Contrastive Learning with Bidirectional Transformers for Sequential Recommendation". In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. CIKM '22. Atlanta, GA, USA: Association for Computing Machinery, 2022, 396–405. ISBN: 9781450392365. DOI: 10.1145/3511808.3557266. URL: https://doi.org/10.1145/3511808.3557266.

[19] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12.null (2011), 2121–2159. ISSN: 1532-4435.

[20] Travis Ebesu, Bin Shen, and Yi Fang. "Collaborative Memory Network for Recommendation Systems". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR '18. Ann Arbor, MI, USA: Association for Computing Machinery, 2018, 515–524. ISBN: 9781450356572.

DOI: 10.1145/3209978.3209991. URL: https://doi.org/10.1145/3209978.3209991.

[21] Xinyan Fan et al. "Lighter and Better: Low-Rank Decomposed Self-Attention Networks for Next-Item Recommendation". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. Virtual Event, Canada: Association for Computing Machinery, 2021, 1733–1737. ISBN: 9781450380379. DOI: 10.1145/3404835.3462978. URL: https://doi.org/10.1145/3404835.3462978.

[22] Ziwei Fan et al. "Modeling sequences as distributions with uncertainty for sequential recommendation". In: *Proceedings of the 30th ACM international conference on information & knowledge management*. 2021, pp. 3019–3023.

[23] Ziwei Fan et al. "Sequential Recommendation via Stochastic Self-Attention". In: *Proceedings of the ACM Web Conference 2022*. WWW '22. Virtual Event, Lyon, France: Association for Computing Machinery, 2022, 2036–2047. ISBN: 9781450390965. DOI: 10.1145/3485447.3512077. URL: https://doi.org/10.1145/3485447.3512077.

[24] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. "Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches". In: *Proceedings of the 13th ACM Conference on Recommender Systems*. RecSys '19. Copenhagen, Denmark: Association for Computing Machinery, 2019, 101–109. ISBN: 9781450362436. DOI: 10.1145/3298689.3347058. URL: https://doi.org/10.1145/3298689.3347058.

[25] Andres Ferraro, Dietmar Jannach, and Xavier Serra. "Exploring Longitudinal Effects of Session-Based Recommendations". In: *Proceedings of the 14th ACM Conference on Recommender Systems*. RecSys '20. Virtual Event, Brazil: Association for Computing Machinery, 2020, 474–479. ISBN: 9781450375832. DOI: 10.1145/3383313.3412213. URL: https://doi.org/10.1145/3383313.3412213.

[26] Yarin Gal and Zoubin Ghahramani. "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, 1027–1035. ISBN: 9781510838819.

[27] Aleksandra Gałka, Jan Grubba, and Krzysztof Walentukiewicz. "Performance and reproducibility of bert4rec". In: *European Conference on Advances in Databases and Information Systems*. Springer. 2023, pp. 620–628.

[28] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. "Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. Barcelona, Spain: Association for Computing Machinery, 2010, 257–260. ISBN: 9781605589060. DOI: 10.1145/1864708.1864761. URL: https://doi.org/10.1145/1864708.1864761.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[30] Michael U. Gutmann and Aapo Hyvärinen. "Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics". In: *J. Mach. Learn. Res.* 13.null (2012), 307–361. ISSN: 1532-4435.

[31] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context". In: *ACM Trans. Interact. Intell. Syst.* 5.4 (2015). ISSN: 2160-6455. DOI: 10.1145/2827872. URL: https://doi.org/10.1145/2827872.

[32] Jesse Harte et al. "Leveraging Large Language Models for Sequential Recommendation". In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, 1096–1102. ISBN: 9798400702419. DOI: 10.1145/3604915.3610639. URL: https://doi.org/10.1145/3604915.3610639.

[33] Xiangnan He et al. "Neural Collaborative Filtering". In: *Proceedings of the 26th International Conference on World Wide Web*. WWW '17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, 173–182. ISBN: 9781450349130. DOI: 10.1145/3038912.3052569. URL: https://doi.org/10.1145/3038912.3052569.

[34] Xiangnan He et al. "Neural Collaborative Filtering". In: *Proceedings of the 26th International Conference on World Wide Web*. WWW '17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, 173–182. ISBN: 9781450349130. DOI: 10.1145/3038912.3052569. URL: https://doi.org/10.1145/3038912.3052569.

[35] Zhankui He et al. "Locker: Locally Constrained Self-Attentive Sequential Recommendation". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021, 3088–3092. ISBN: 9781450384469. DOI: 10.1145/3459637.3482136. URL: https://doi.org/10.1145/3459637.3482136.

[36] Dan Hendrycks and Kevin Gimpel. "Gaussian error linear units (gelus)". In: *arXiv preprint arXiv:1606.08415* (2016).

[37] Balázs Hidasi and Ádám Tibor Czapp. "The Effect of Third Party Implementations on Reproducibility". In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, 272–282. ISBN: 9798400702419. DOI: 10.1145/3604915.3609487. URL: https://doi.org/10.1145/3604915.3609487.

[38] Balázs Hidasi and Ádám Tibor Czapp. "Widespread Flaws in Offline Evaluation of Recommender Systems". In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, 848–855. ISBN: 9798400702419. DOI: 10.1145/3604915.3608839. URL: https://doi.org/10.1145/3604915.3608839.

[39] Balázs Hidasi and Alexandros Karatzoglou. "Recurrent neural networks with top-k gains for session-based recommendations". In: *Proceedings of the 27th ACM international conference on information and knowledge management*. 2018, pp. 843–852.

[40] Balázs Hidasi et al. "Session-based Recommendations with Recurrent Neural Networks". In: (Nov. 2015).

[41] Yupeng Hou et al. "CORE: Simple and Effective Session-Based Recommendation within Consistent Representation Space". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '22. Madrid, Spain: Association for Computing Machinery, 2022, 1796–1801. ISBN: 9781450387323. DOI: 10.1145/3477495.3531955. URL: https://doi.org/10.1145/3477495.3531955.

[42] Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative Filtering for Implicit Feedback Datasets". In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 263–272. DOI: 10.1109/ICDM.2008.22.

[43] Dietmar Jannach, Lukas Lerche, and Markus Zanker. "Recommending Based on Implicit Feedback". In: *Social Information Access: Systems and Technologies*. Ed. by Peter Brusilovsky and Daqing He. Cham: Springer International Publishing, 2018, pp. 510–569. ISBN: 978-3-319-90092-6. DOI: 10.1007/978-3-319-90092-6_14. URL: https://doi.org/10.1007/978-3-319-90092-6_14.

[44] Dietmar Jannach and Malte Ludewig. "When Recurrent Neural Networks Meet the Neighborhood for Session-Based Recommendation". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. RecSys '17. Como, Italy: Association for Computing Machinery, 2017, 306–310. ISBN: 9781450346528. DOI: 10.1145/3109859.3109872. URL: https://doi.org/10.1145/3109859.3109872.

[45] Dietmar Jannach et al. "What Recommenders Recommend: An Analysis of Recommendation Biases and Possible Countermeasures". In: *User Modeling and User-Adapted Interaction* 25.5 (2015), 427–491. ISSN: 0924-1868. DOI: 10.1007/s11257-015-9165-3. URL: https://doi.org/10.1007/s11257-015-9165-3.

[46] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated Gain-Based Evaluation of IR Techniques". In: *ACM Trans. Inf. Syst.* 20.4 (2002), 422–446. ISSN: 1046-8188. DOI: 10.1145/582415.582418. URL: https://doi.org/10.1145/582415.582418.

[47] Wang-Cheng Kang and Julian J. McAuley. "Self-Attentive Sequential Recommendation". In: *Proceedings of IEEE International Conference on Data Mining*. ICDM '19. IEEE Computer Society, 2018, pp. 197–206. ISBN: 978-1-5386-9159-5. DOI: 10.1109%2fICDM.2018.00035.

[48] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[49] Anton Klenitskiy and Alexey Vasilev. "Turning Dross Into Gold Loss: Is BERT4Rec Really Better than SASRec?" In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, 1120–1125. ISBN: 9798400702419. DOI: 10.1145/3604915.3610644. URL: https://doi.org/10.1145/3604915.3610644.

[50] Yehuda Koren. "Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model". In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '08. Las Vegas, Nevada, USA: Association for Computing Machinery, 2008, 426–434. ISBN: 9781605581934. DOI: 10.1145/1401890.1401944. URL: https://doi.org/10.1145/1401890.1401944.

[51] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8 (2009), pp. 30–37. DOI: 10.1109/MC.2009.263.

[52] Walid Krichene and Steffen Rendle. "On Sampled Metrics for Item Recommendation". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, 1748–1757. ISBN: 9781450379984. DOI: 10.1145/3394486.3403226. URL: https://doi.org/10.1145/3394486.3403226.

[53] Sara Latifi, Dietmar Jannach, and Andrés Ferraro. "Sequential Recommendation: A Study on Transformers, Nearest Neighbors and Sampled Metrics". In: *Inf. Sci.* 609.C (2022), 660–678. ISSN: 0020-0255. DOI: 10.1016/j.ins.2022.07.079. URL: https://doi.org/10.1016/j.ins.2022.07.079.

[54] Chengxi Li et al. "STRec: Sparse Transformer for Sequential Recommendations". In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, 101–111. ISBN: 9798400702419. DOI: 10.1145/3604915.3608779. URL: https://doi.org/10.1145/3604915.3608779.

[55] Jiacheng Li, Yujie Wang, and Julian McAuley. "Time Interval Aware Self-Attention for Sequential Recommendation". In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. WSDM '20. Houston, TX, USA: Association for Computing Machinery, 2020, 322–330. ISBN: 9781450368223. DOI: 10.1145/3336191.3371786. URL: https://doi.org/10.1145/3336191.3371786.

[56] Jing Li et al. "Neural Attentive Session-Based Recommendation". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM '17. Singapore, Singapore: Association for Computing Machinery, 2017, 1419–1428. ISBN: 9781450349185. DOI: 10.1145/3132847.3132926. URL: https://doi.org/10.1145/3132847.3132926.

[57] Jing Li et al. "Neural Attentive Session-Based Recommendation". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM '17. Singapore, Singapore: Association for Computing Machinery, 2017, 1419–1428. ISBN: 9781450349185. DOI: 10.1145/3132847.3132926. URL: https://doi.org/10.1145/3132847.3132926.

[58] Li Li, Yuxi Fan, and Kuo-Yi Lin. "A Survey on federated learning". In: *2020 IEEE 16th International Conference on Control Automation (ICCA)*. 2020, pp. 791–796. DOI: 10.1109/ICCA51439.2020.9264412.

[59] Yang Li et al. "Lightweight Self-Attentive Sequential Recommendation". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021, 967–977. ISBN: 9781450384469. DOI: 10.1145/3459637.3482448. URL: https://doi.org/10.1145/3459637.3482448.

[60] Chang Liu et al. "Noninvasive self-attention for side information fusion in sequential recommendation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 5. 2021, pp. 4249–4256.

[61] Yang Liu, Alan Medlar, and Dorota Glowacka. "On the Consistency, Discriminative Power and Robustness of Sampled Metrics in Offline Top-N Recommender System Evaluation". In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, 1152–1157. ISBN: 9798400702419. DOI: 10.1145/3604915.3610651. URL: https://doi.org/10.1145/3604915.3610651.

[62] Zhiwei Liu et al. "Augmenting Sequential Recommendation with Pseudo-Prior Items via Reversely Pre-Training Transformer". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. Virtual Event, Canada: Association for Computing Machinery, 2021, 1608–1612. ISBN: 9781450380379. DOI: 10.1145/3404835.3463036. URL: https://doi.org/10.1145/3404835.3463036.

[63] Zhiwei Liu et al. "Contrastive self-supervised sequential recommendation with robust augmentation". In: *arXiv preprint arXiv:2108.06479* (2021).

[64] Malte Ludewig and Dietmar Jannach. "Evaluation of session-based recommendation algorithms". In: *User Modeling and User-Adapted Interaction* 28 (2018), pp. 331–390.

[65] Malte Ludewig and Dietmar Jannach. "User-Centric Evaluation of Session-Based Recommendations for an Automated Radio Station". In: *Proceedings of the 13th ACM Conference on Recommender Systems*. RecSys '19. Copenhagen, Denmark: Association for Computing Machinery, 2019, 516–520. ISBN: 9781450362436. DOI: 10.1145/3298689.3347046. URL: https://doi.org/10.1145/3298689.3347046.

[66] Malte Ludewig et al. "Empirical analysis of session-based recommendation algorithms: A comparison of neural and non-neural approaches". In: *User Modeling and User-Adapted Interaction* 31 (2021), pp. 149–181.

[67] Haokai Ma et al. "Exploring False Hard Negative Sample in Cross-Domain Recommendation". In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, 502–514. ISBN: 9798400702419. DOI: 10.1145/3604915.3608791. URL: https://doi.org/10.1145/3604915.3608791.

[68] Jianxin Ma et al. "Disentangled Self-Supervision in Sequential Recommenders". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, 483–491. ISBN: 9781450379984. DOI: 10.1145/3394486.3403091. URL: https://doi.org/10.1145/3394486.3403091.

[69] Julian McAuley et al. "Image-Based Recommendations on Styles and Substitutes". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '15. Santiago, Chile: Association for Computing Machinery, 2015, 43–52. ISBN: 9781450336215. DOI: 10.1145/2766462.2767755. URL: https://doi.org/10.1145/2766462.2767755.

[70] Bibek Paudel et al. "Updatable, Accurate, Diverse, and Scalable Recommendations for Interactive Applications". In: *ACM Trans. Interact. Intell. Syst.* 7.1 (2016). ISSN: 2160-6455. DOI: 10.1145/2955101. URL: https://doi.org/10.1145/2955101.

[71] Aleksandr Petrov and Craig Macdonald. "A Systematic Review and Replicability Study of BERT4Rec for Sequential Recommendation". In: *Proceedings of the 16th ACM Conference on Recommender Systems*. RecSys '22. Seattle, WA, USA: Association for Computing Machinery, 2022, 436–447. ISBN: 9781450392785. DOI: 10.1145/3523227.3548487. URL: https://doi.org/10.1145/3523227.3548487.

[72] Aleksandr Petrov and Craig Macdonald. "Effective and Efficient Training for Sequential Recommendation Using Recency Sampling". In: *Proceedings of the 16th ACM Conference on Recommender Systems*. RecSys '22. Seattle, WA, USA: Association for Computing Machinery, 2022, 81–91. ISBN: 9781450392785. DOI: 10.1145/3523227.3546785. URL: https://doi.org/10.1145/3523227.3546785.

[73] Aleksandr Vladimirovich Petrov and Craig Macdonald. "GSASRec: Reducing Overconfidence in Sequential Recommendation Trained with Negative Sampling". In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, 116–128. ISBN: 9798400702419. DOI: 10.1145/3604915.3608783. URL: https://doi.org/10.1145/3604915.3608783.

[74] Ruihong Qiu et al. "Contrastive Learning for Representation Degeneration Problem in Sequential Recommendation". In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. WSDM '22. Virtual Event, AZ, USA: Association for Computing Machinery, 2022, 813–823. ISBN: 9781450391320. DOI: 10.1145/3488560.3498433. URL: https://doi.org/10.1145/3488560.3498433.

[75] Massimo Quadrana et al. "Personalizing Session-Based Recommendations with Hierarchical Recurrent Neural Networks". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. RecSys '17. Como, Italy: Association for Computing Machinery, 2017, 130–137. ISBN: 9781450346528. DOI: 10.1145/3109859.3109896. URL: https://doi.org/10.1145/3109859.3109896.

[76] Pengjie Ren et al. "RepeatNet: A Repeat Aware Neural Recommendation Machine for Session-Based Recommendation". In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. ISBN: 978-1-57735-809-1. DOI: 10.1609/aaai.v33i01.33014806. URL: https://doi.org/10.1609/aaai.v33i01.33014806.

[77] Steffen Rendle et al. "BPR: Bayesian Personalized Ranking from Implicit Feedback". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. UAI '09. Montreal, Quebec, Canada: AUAI Press, 2009, 452–461. ISBN: 9780974903958.

[78] Francesco Ricci, Lior Rokach, and Bracha Shapira. "Recommender systems: introduction and challenges". In: *Recommender systems handbook* (2015), pp. 1–34.

[79] Stephen Robertson. "Understanding inverse document frequency: on theoretical arguments for IDF". In: *Journal of documentation* 60.5 (2004), pp. 503–520.

[80] Peter Gordon Roetzel. "Information overload in the information age: a review of the literature from business administration, business psychology, and related disciplines with a bibliometric approach and framework development". In: *Business Research* (2018), pp. 1–44. URL: https://api.semanticscholar.org/CorpusID:149923282.

[81] Badrul Sarwar et al. "Item-Based Collaborative Filtering Recommendation Algorithms". In: *Proceedings of the 10th International Conference on World Wide Web*. WWW '01. Hong Kong, Hong Kong: Association for Computing Machinery, 2001, 285–295. ISBN: 1581133480. DOI: 10.1145/371920.372071. URL: https://doi.org/10.1145/371920.372071.

[82] Barry Schwartz. *The paradox of choice: Why more is less*. Harper Perennial, 2004.

[83] Guy Shani, Ronen I. Brafman, and David Heckerman. "An MDP-Based Recommender System". In: *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*. UAI'02. Alberta, Canada: Morgan Kaufmann Publishers Inc., 2002, 453–460. ISBN: 1558608974.

[84] Faisal Shehzad and Dietmar Jannach. "Everyone's a Winner! On Hyperparameter Tuning of Recommendation Models". In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, 652–657. ISBN: 9798400702419. DOI: 10.1145/3604915.3609488. URL: https://doi.org/10.1145/3604915.3609488.

[85] Derya Soydaner. "A Comparison of Optimization Algorithms for Deep Learning". In: *International Journal of Pattern Recognition and Artificial Intelligence* 34.13 (2020), p. 2052013. DOI: 10.1142/S0218001420520138. eprint: https://doi.org/10.1142/S0218001420520138. URL: https://doi.org/10.1142/S0218001420520138.

[86] Cheri Speier, Joseph S Valacich, and Iris Vessey. "The influence of task interruption on individual decision making: An information overload perspective". In: *Decision sciences* 30.2 (1999), pp. 337–360.

[87] Fei Sun et al. "BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. Beijing, China: Association for Computing Machinery, 2019, 1441–1450. ISBN: 9781450369763. DOI: 10.1145/3357384.3357895. URL: https://doi.org/10.1145/3357384.3357895.

[88] Gábor Takács et al. "Matrix Factorization and Neighbor Based Algorithms for the Netflix Prize Problem". In: *Proceedings of the 2008 ACM Conference on Recommender Systems*. RecSys '08. Lausanne, Switzerland: Association for Computing Machinery, 2008, 267–274. ISBN: 9781605580937. DOI: 10.1145/1454008.1454049. URL: https://doi.org/10.1145/1454008.1454049.

[89] Yong Kiam Tan, Xinxing Xu, and Yong Liu. "Improved Recurrent Neural Networks for Session-Based Recommendations". In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. DLRS 2016. Boston, MA, USA: Association for Computing Machinery, 2016, 17–22. ISBN: 9781450347952. DOI: 10.1145/2988450.2988452. URL: https://doi.org/10.1145/2988450.2988452.

[90] Gongbo Tang et al. "Why self-attention? a targeted evaluation of neural machine translation architectures". In: *arXiv preprint arXiv:1808.08946* (2018).

[91] Jiaxi Tang and Ke Wang. "Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. WSDM '18. Marina Del Rey, CA, USA: Association for Computing Machinery, 2018, 565–573. ISBN: 9781450355810. DOI: 10.1145/3159652.3159656. URL: https://doi.org/10.1145/3159652.3159656.

[92] Wilson L Taylor. ""Cloze procedure": A new tool for measuring readability". In: *Journalism quarterly* 30.4 (1953), pp. 415–433.

[93] Ashish Vaswani et al. "Attention is All You Need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, 6000–6010. ISBN: 9781510860964.

[94] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. "DropoutNet: Addressing Cold Start in Recommender Systems". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, 4964–4973. ISBN: 9781510860964.

[95] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. "Unifying User-Based and Item-Based Collaborative Filtering Approaches by Similarity Fusion". In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '06. Seattle, Washington, USA: Association for Computing Machinery, 2006, 501–508. ISBN: 1595933697. DOI: 10.1145/1148170.1148257. URL: https://doi.org/10.1145/1148170.1148257.

[96] Shoujin Wang et al. "A Survey on Session-Based Recommender Systems". In: *ACM Comput. Surv.* 54.7 (2021). ISSN: 0360-0300. DOI: 10.1145/3465401. URL: https://doi.org/10.1145/3465401.

[97] Tianxin Wei et al. "Model-Agnostic Counterfactual Reasoning for Eliminating Popularity Bias in Recommender System". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. KDD '21. Virtual Event, Singapore: Association for Computing Machinery, 2021, 1791–1800. ISBN: 9781450383325. DOI: 10.1145/3447548.3467289. URL: https://doi.org/10.1145/3447548.3467289.

[98] Liwei Wu et al. "SSE-PT: Sequential Recommendation Via Personalized Transformer". In: *Proceedings of the 14th ACM Conference on Recommender Systems*. RecSys '20. Virtual Event, Brazil: Association for Computing Machinery, 2020, 328–337. ISBN: 9781450375832. DOI: 10.1145/3383313.3412258. URL: https://doi.org/10.1145/3383313.3412258.

[99] Xu Xie et al. "Contrastive learning for sequential recommendation". In: *2022 IEEE 38th international conference on data engineering (ICDE)*. IEEE. 2022, pp. 1259–1273.

[100] Chengfeng Xu et al. "Long-and short-term self-attention network for sequential recommendation". In: *Neurocomputing* 423 (2021), pp. 580–589.

[101] Shahpar Yakhchi et al. "Towards a Deep Attention-Based Sequential Recommender System". In: *IEEE Access* 8 (2020), pp. 178073–178084. DOI: 10.1109/ACCESS.2020.3004656.

[102] Junliang Yu et al. "Self-supervised learning for recommender systems: A survey". In: *IEEE Transactions on Knowledge and Data Engineering* (2023).

[103] Shuai Zhang et al. "Deep Learning Based Recommender System: A Survey and New Perspectives". In: *ACM Comput. Surv.* 52.1 (2019). ISSN: 0360-0300. DOI: 10.1145/3285029. URL: https://doi.org/10.1145/3285029.

[104] Shuai Zhang et al. "Deep Learning Based Recommender System: A Survey and New Perspectives". In: *ACM Comput. Surv.* 52.1 (2019). ISSN: 0360-0300. DOI: 10.1145/3285029. URL: https://doi.org/10.1145/3285029.

[105] Tingting Zhang et al. "Feature-Level Deeper Self-Attention Network for Sequential Recommendation". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI'19. Macao, China: AAAI Press, 2019, 4320–4326. ISBN: 9780999241141.

[106] Qihang Zhao. "RESETBERT4Rec: A Pre-Training Model Integrating Time And User Historical Behavior for Sequential Recommendation". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '22. Madrid, Spain: Association for Computing Machinery, 2022, 1812–1816. ISBN: 9781450387323. DOI: 10.1145/3477495.3532054. URL: https://doi.org/10.1145/3477495.3532054.

[107] Yuhang Zhao. "Analysis of TikTok's Success Based on Its Algorithm Mechanism". In: *2020 International Conference on Big Data and Social Sciences (ICBDSS)*. 2020, pp. 19–23. DOI: 10.1109/ICBDSS51270.2020.00012.

[108] Kun Zhou et al. "S3-Rec: Self-Supervised Learning for Sequential Recommendation with Mutual Information Maximization". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. CIKM '20. Virtual Event, Ireland: Association for Computing Machinery, 2020, 1893–1902. ISBN: 9781450368599. DOI: 10.1145/3340531.3411954. URL: https://doi.org/10.1145/3340531.3411954.

[109] Tao Zhou et al. "Solving the apparent diversity-accuracy dilemma of recommender systems". In: *Proceedings of the National Academy of Sciences* 107.10 (2010), pp. 4511–4515.