TÉCNICO LISBOA

Context-aware Session-based Recommendation

An Attentional Deep Learning approach to Re-ranking

Tomás Alexandre de Menezes Pereira

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisor: Prof. Susana Margarida da Silva Vieira

Examination Committee

Chairperson: Prof. Carlos Baptista Cardeira
Supervisor: Prof. Susana Margarida da Silva Vieira
Member of the Committee: Prof. Luís Manuel Fernandes Mendonça

October 2021

Acknowledgments

I want to thank my dissertation supervisor, Prof. Susana Vieira, for the relevant feedback and many drifting conversations.

To all of my driven friends, colleagues, and self-appointed thesis reviewers. Thank you for sharing this intricate academic journey of learning and growth with me. It would have paled in comparison to what it has been, without all the memories and continuous motivation you provided (especially throughout all the typical engineering sleepless nights), year after year.

To my family. My uncles, António and Nuno, and my grandparents, João, Ilda, and Graça. Thank you for your knowledge, for the incessant support, for showing me how to turn challenges into opportunities, and for pushing me to dream big while challenging the status quo. To my sister Sara, and my parents Sandra and Paulo, who inspire me every day to strive to become a better version of myself. Thank you for instilling in me the need to work hard, to put everything that I have into everything that I do, and for providing me with experiences and tools to reach my goals without ever asking for anything in return. I'm proud to have you as role models.

Resumo

Os sistemas de recomendação moderam uma quantidade substancial da interface de interação online, fornecendo uma gama cada vez mais relevante de benefícios para utilizadores e fornecedores de negócios, numa era de crescimento exponencial de conteúdos, produtos e serviços digitais disponíveis.

Alguns domínios, como aplicações de viagens e de e-commerce, estão dependentes de perfis de atividade esporádica e maioritariamente anónima, permitindo apenas acesso a dinâmicas derivadas de sessões de curto prazo que exigem soluções robustas independentes de perfis de utilizadores estruturados. Este trabalho abrange o desenvolvimento de um sistema de recomendação para re-ranking com deep learning e mecanismos de atenção, sujeito a estas condições desafiadoras, com base nos dados fornecidos para o 2019 ACL Recommender Systems Challenge.

A abordagem desenvolvida demonstra um resultado no 85º percentil (em termos de MRR online previsto, considerando um intervalo de resultados limitado a valores superiores aos da baseline fornecida) com geração mínima de features num ambiente computacional bastante restritivo. As várias inconsistências inerentes ao campo de investigação, incluindo a divergência proeminente entre objetivos de produção offline e online, motivaram uma mudança no foco principal do trabalho de desempenho ao nível da tabela classificativa, que normalmente recompensa o excesso de engenharia de features e complexidade de modelos. Em alternativa, foi enfatizada a análise do potencial de aprendizagem de representações e do impacto dos componentes específicos da arquitetura obtida, derivados de um procedimento de Otimização Bayesiana automática com Processos Gaussianos sobre um espaço de hiperparâmetros altamente condicional.

Palavras-chave: Sistemas de recomendação, recomendação baseada em sessões, modelação de interações sequenciais, deep learning, self-attention, otimização bayesiana

Abstract

Recommender systems mediate a substantial share of the online interaction interface, providing an ever-more relevant array of benefits for both users and business providers amidst the exponential growth of available digital content, products and services.

Some domains, such as travel and e-commerce applications experience sporadic and mostly anonymous activity, thereby only observing short-term session-based dynamics, requiring robust solutions independent of structured user profiles. This work encompasses the development of an attentional deep learning re-ranking recommender under these challenging conditions, on the basis of the dataset provided for the 2019 ACL Recommender Systems Challenge.

The developed approach demonstrates an 85th percentile result (in predicted online MRR, considering a limited score range exceeding that of the provided baseline) with minimal feature generation effort in a very restricted computational environment. Nonetheless, various inconsistencies in the research field including the prominent divergence between offline and online production objectives, motivated a shift in the main focus away from leaderboard performance, which typically rewards feature over-engineering and model complexity. An analysis of representation learning's potential and specific component impact, derived from an automatic Bayesian Optimization with Gaussian Processes procedure over a highly conditional hyperparameter space, was emphasized instead.

Keywords: Recommender systems, session-based recommendation, sequential interaction modeling, deep learning, self-attention, bayesian optimization

Contents

List of Tables

List of Figures

Nomenclature

$a^{(\tau)}$	Event interaction at time step τ
b_i	Item i's rating deviation from the global rating average
b_u	User u's rating deviation from the global rating average
$\boldsymbol{c}^{(\tau)}$	Context vector at time step τ
$\mathcal{D}$	Full dataset
d_{emb}	Arbitrary embedding dimensionality
d_{filt}	Filters' embedding dimension
d_{meta}	Metadata attributes' embedding dimension
d_{voc}	Arbitrary vocabulary size
$\boldsymbol{e}^{(\tau)}$	τ^{th} session event
$\boldsymbol{e}_h^{(\tau_h)}$	h^{th} click event at time step τ_h
ϵ	Small time interval
h_{imp}	Number of impression branch GRU hidden units
h_{seq}	Number of interaction sequence branch GRU hidden units
i or $i^{(\tau)}$	Arbitrary item or reference item at time step τ
K	Number of classes/relevant ranking items
$\boldsymbol{k}^{(h)}$	Impression list associated with the h^{th} click event
$k_p^{(h)}$	Impression list h's item at position p
μ_r	Overall mean item rating
m_{imp}	Impression feature space dimensionality
m_{seq}	Sequential feature space dimensionality

m_{ses}	Session feature space dimensionality
N	Total number of samples
n	Total number of events
n_{imp}	Maximum number of impression list items
n_{items}	Number of items
n_{seq}	Number of interacted sequence time steps
n_{users}	Number of users
$\Omega_q^{(\tau)}$	Previous action of type q at time step τ
$\boldsymbol{P}$	Latent user matrix
$\boldsymbol{p}_u$	User u's embedding vector
$\boldsymbol{Q}$	Latent item matrix
$\boldsymbol{q}_i$	Item i's embedding vector
$\boldsymbol{R}$	Utility matrix
$\hat{r}_{ui}$	Rating prediction for user u and item i
rank_j	Predicted target rank for sample j
r_{meta}	Clicked to general availability metadata ratio
r_{og}	Original ratio of most to least frequent action count
r_{pre}	Preprocessed ratio of most to least frequent action count
$\boldsymbol{S}^{(j)}$	Arbitrary user session j
$\boldsymbol{s}^{(j)}$	Arbitrary subsession/sample j
$\Delta t_j^{(\tau)}$	Time delta of type j for time step τ
τ_{Ω_q}	Time step of previous action of type q
$t^{(\tau)}$	Event timestamp at time step τ; takes the form t_{ini} and t_{end} for the temporal endpoints of interaction groups
u	Arbitrary user
w	Number of session clicks
$\bar{\boldsymbol{X}}$	Raw feature model input
$\boldsymbol{X}$	General transformed model input

$\boldsymbol{X}_{\text{filt}}$	Filter features
$\boldsymbol{X}_{\text{imp}}$	Impression features
$\boldsymbol{X}_{\text{meta}}$	Metadata features
$\boldsymbol{X}_{\text{seq}}$	Interaction sequence features
$\boldsymbol{X}_{\text{ses}}$	Session features
$\boldsymbol{y}_{\text{rank}}$	Ranked predicted output probabilities (with sorted indices)
$\hat{\boldsymbol{y}}^{(h)}$	Predicted impression click probabilities for click event h

Chapter 3. Deep Learning

α_n	Gaussian process acquisition function at iterative step n
$\boldsymbol{\alpha}$	Attention weights
β_n	Upper confidence bound tradeoff factor at iterative step n
$\boldsymbol{b}^{(k)}$	Trainable bias associated with network layer k
B	Minibatch
$\boldsymbol{c}$	Attention context vector
C	Confusion matrix
$\mathcal{D}_n$	Dataset with n points or at iteration n, if mutable
$\mathcal{D}_{\text{test}}$	Test set
$\mathcal{D}_{\text{train}}$	Training set
d	Euclidean distance
p_{drop}	Dropout probability
ϵ	Gaussian noise
η	Learning rate
e_{ij}	Attention alignment score between source and target annotations, $\boldsymbol{h}^{(j)}$ and $\bar{\boldsymbol{h}}^{(i)}$
$f(\boldsymbol{x};\boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$
Γ	Gamma function
$\hat{\boldsymbol{g}}$	Minibatch-based unbiased estimator of the generalization error's gradient
$g^{(k)}$	Arbitrary activation function associated with network layer k; takes the Heaviside H, sigmoid σ, $\tanh$ and ReLU variation forms in Appendix

$\bar{\boldsymbol{h}}^{(i)}$	Target annotation for item $\boldsymbol{x}^{(i)}$
$\tilde{\boldsymbol{h}}^{(t)}$	GRU candidate state vector
$\boldsymbol{h}^{(k)}$ or $\boldsymbol{h}^{(t)}$	Hidden layer output from layer k or at time step t
$\overrightarrow{\boldsymbol{h}}$ and $\overleftarrow{\boldsymbol{h}}$	Forward and backward hidden state annotation vectors
$\mathbf{I}$	Identity matrix
J	Cost function
k or $\mathbf{K}$	Gaussian process kernel (covariance) function or matrix; assumes the posterior σ_n^2 form at iterative step n
K	Number of multiclass classification classes
K_ν	Modified Bessel function
k_ν	*Matérn* kernel function with smoothness parameter ν
L	Per-example loss function
l	Gaussian process kernel length scale parameter
m or $\mathbf{m}$	Gaussian process mean function or vector; assumes the prior μ_0 and posterior μ_n forms at iterative step n
$\boldsymbol{r}^{(t)}$	GRU reset gate vector
σ^2_{noise}	Noise variance
$\boldsymbol{\theta}$	Model parameters
$\boldsymbol{\theta}_{ML}$	Maximum likelihood model parameter estimate
$\boldsymbol{w}^{(k)}$ or $\boldsymbol{W}^{(k)}$	Trainable weight vector or matrix associated with network layer k; specific attention layer weights can assume the form $\boldsymbol{u}$ or $\boldsymbol{v}$
$\boldsymbol{X}$	Matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$; subsets may be denoted with subscripts such as train, test, val and subtrain
$\mathbf{x}$	Collection or sequence of n input points
$\mathcal{X}$	Hyperparameter configuration design space
$\boldsymbol{x}^{(i)}$ or $\boldsymbol{x}^{(t)}$	The i-th example (input) from a dataset or the input at time step t; the subscript form $\boldsymbol{x}_i$ is used for the optimization section
$\boldsymbol{x}^*$	Objective function maximizer (or minimizer), best hyperparameter configuration
$\hat{\boldsymbol{y}}^{(i)}$ or $\hat{\boldsymbol{y}}^{(t)}$	Output prediction vector associated with $\boldsymbol{x}^{(i)}$ or $\boldsymbol{x}^{(t)}$

$\mathbf{y}$	Collection of n observations or target sequence points; subsets may be denoted with subscripts such as train, test, val and subtrain
$y^{(i)}$, $\boldsymbol{y}^{(i)}$ or $\boldsymbol{y}^{(t)}$	General output or the label/target associated with $\boldsymbol{x}^{(i)}$ or $\boldsymbol{x}^{(t)}$, for supervised learning; the subscript form $\boldsymbol{y}_i$ is used for the optimization section
$\boldsymbol{z}^{(t)}$	GRU update gate vector

Chapter 1

Introduction

1.1 Motivation

As the amount of available digital information continues to increase exponentially, allowing for unprecedented access to large-scale services and products, so does the need for systems that can effectively reduce the impact of overchoice by filtering and retrieving relevant content for users. Recommender systems, as they are known, constitute key item discovery and exploration platforms, resulting in business growth and substantial added value to the service providers [1]. YouTube, for instance, which has recently reported a striking upload rate of 500 hours of content per minute [2], credits their recommendations for driving more than 70% of web traffic on the platform [3], a figure that closely follows Netflix's estimated 80%, with search accounting for the rest [4]. Another example is that of Amazon's purchases derived from recommendations, which McKinsey has predicted to be of around 35% [5]. Consequently, keyphrases such as "recommended for you", "others have liked" or "you may also know" keep reshaping the digital landscape, transcending the online shopping, media and advertisement domains, and expanding into an ever-growing diversity of fields.

Recommendation methods have been refined over time to further enhance the personalized user experience, usually only made possible with detailed knowledge and deep understanding of the item space, the users, their preferences and behavior, and additional contextualization. Item similarity and neighborhood-based methods have given way to latent factor models that take different users' interaction information into account and hybrid strategies that can better tackle cold start[1] situations and provide multi-scale data modeling. The generalization of these concepts with neural network-based architectures, for instance, has allowed for increased control over the incorporation of crucial multi-modal contextual signals, enabling circumstantial factors to adapt the recommendation space, among an array of other advantages [6]. This is especially useful in the sequential recommendation setting, where temporal patterns and trends from past implicit and explicit feedback, usually derived from activity logs, can be complemented with features modeling the continually changing needs and interests of users to infer their intent and predict future interactions.

[1] Cold-start refers to the lack of interaction information usually associated with new item or user profiles, non-logged in searches or infrequent activity producing mostly independent session behavior.

Nonetheless, some domains such as travel and e-commerce applications experience sporadic and mostly anonymous activity, thereby only observing short-term dynamics, requiring alternative solutions independent of structured user profiles. This thesis encompasses the development of a recommender under these challenging conditions on the basis of the dataset provided for the 2019 ACL Recommender Systems Challenge (Section 1.2), using state-of-the-art methods such as attention mechanisms, ubiquitous in modern approaches. The misalignment between most offline objectives and those relevant in production environments (see Section 2.3.3) motivated a shift in focus away from maximum leaderboard performance achievement with traditional competition strategies such as the usage of overly complex models or hundreds of input features. Instead, the model's architectural optimization process, usually disregarded in the literature, was emphasized in an exploration of representation learning's potential. Additionally, the developed model was tested against simpler rule-based baselines which have recently produced competitive results even against the state of the art in sequential recommendation, and its specific component impact was also assessed.

Although not the focus of this work, the study of these algorithms at the core of news, social media, and educational or entertainment content feeds is relevant not only for their benefits but also for the exact opposite. The aforementioned YouTube content figures can be used to provide some perspective on how insignificantly small the realistically accessible part of the corpus is when compared to the total amount of available information on the platform, a property reflected across various domains. The amount of control over when, how, or what information gets disseminated provides the ideal setup for a highly manipulative setting, strongly tied to the design considerations and objectives of these underlying systems.[2] The number of popular services[3] exploiting user engagement and psychological cues derived therefrom (trust, cognitive dissonance, etc.) [8, 9] to build highly addictive algorithms capable of changing beliefs and behaviors [10] is concerning, promoting current discussions about ethics, fairness, diversity, explainability and interpretability in recommendations [6, 11].

1.2 The RecSys 2019 challenge

In its 2019 edition, the annual ACL Recommender Systems (RecSys) Challenge was organized by *trivago*, TU Wien, Politecnico di Bari and Karlsruhe Institute of Technology with the goal of exploring context-aware recommendation on a highly sparse session-based setting, unsuitable for most traditional approaches [12]. The selected digital travel domain is subject to a variety of obstacles, including the constantly changing prices, offers, and availability of a vast collection of accommodation, the extreme cold-start induced by the infrequent listing browsing by mostly anonymous users, and the highly dynamic search criteria motivated by their variable long and short-term preferences and trip-specific needs [13].

trivago's global search platform aggregates localized accommodation information (including price,

[2] Covington et al. [7], for instance, noted that the simple change in ranking objective from click-through rate to watch time allowed for better retention of user engagement while significantly reducing the promotion of deceptive and clickbait videos.

[3] Especially free social platforms with business models revolving around user data.

Figure 1.1: trivago's website interaction distribution with the user action types available in the RecSys19 dataset (with shortened identifiers) presented in Table 1.2, adapted from [13].

rating, visual and textual descriptions, among others) based on user queries. Users can interact with the listings in multiple ways, some of which are presented in Figure 1.1, and click on relevant ones to be redirected to external affiliate booking sites where transactions can be completed.

Naturally influenced by the platform's primary revenue stream centered on cost-per-click [14], the technical objective of the challenge consisted in the prediction of which items were more likely to be clicked at the end of each of their user's largest sessions, from lists of items (impressions) presented at these events to improve the ranking process. This required the re-ordering of the impression lists by click likelihood (representing contextual relevance), which was evaluated with the Mean Reciprocal Rank (MRR),

$$\mathrm{MRR} = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{\mathrm{rank}_j}, \tag{1.1}$$

where N is the number of evaluated lists (samples) and rank_j is the predicted position rank for the clicked item (target label) in sample j. The Reciprocal Rank's value, and consequent new ranking performance for a sample, increases with the proximity of the target to the top of each list, reaching its maximum of one for ranked lists where the clicked item appears in the top position ($\mathrm{rank}_j = 1$).

Data description The data provided by *trivago* was comprised of anonymized user session interaction logs recorded in their platform over eight days, 01-08 November 2018, and an additional accommodation metadata database with item-specific attribute lists. Each log entry is characterized by the features displayed in Table 1.1, and consists of a single user interaction over a given item, from the fixed set of ten available types presented in Table 1.2.

Table 1.1: RSC19 original features.

Original feature	Description
User ID	User identifier
Session ID	Session identifier
Timestamp	UNIX timestamp in seconds for the time of interaction
Step	Time step in the sequence of events within the session
Action	Action type for each event (see Table 1.2)
Reference	Reference identifier for each action (see Table 1.2)
Platform	Country of the web platform used for the search
City	City name for the session search context
Device	Device used for the search
Filters	List of active filters at a given timestamp
Impressions	List of item identifiers displayed to the user in a click event
Prices	Corresponding list of nightly prices for the impression items
Metadata	Specific item attributes (features and amenities)

Table 1.2: Types of user actions and respective action references.

Action type	Description	Reference
Clickout	Item click that redirects the user to an affiliate website	Item ID
Interaction item rating	Interaction with an item's rating/review elements	Item ID
Interaction item info	Interaction with an item's information elements	Item ID
Interaction item image	Interaction with an item's images	Item ID
Interaction item deals	Interaction to extend a given item's affiliate price deals element	Item ID
Search for item	Specific accommodation search	Item ID
Change of sort order	User determined sort of the presented impressions, by price, distance, rating and popularity	Sort type
Filter selection	User determined impression filtering by feature (e.g., amenities or minimum number of stars)	Filter type
Search for destination	Specific location-based filtering	Location
Search for POI	Specific point of interest-based filtering	Point of interest

The logs were provided in separate training and test sets, with the latter consisting of events occurring in the last two days (07 and 08 November), for which the ground truth labels of the clicked items in the largest user sessions were omitted. Up until the challenge's deadline of July 2019, the user-submitted entries, consisting of predicted rank lists for the relevant click event items, were evaluated on an online platform which has, since then, ceased to operate. At the time of writing, the omitted test labels have not yet been made publicly available for offline use. As such, the data regarded in this work is limited to the six-day training set, with its almost 16 million actions, 730 803 users, and 910 683 sessions, from here on defined as RSC19.

The local evaluation procedure and further considerations surrounding the data setup are discussed in Section 4.2.3.

1.3 Objectives

This work's research objectives (ROs) were divided into three main points, already lightly introduced in the first Section 1.1, subject to the limitations described in the later Section 2.3.3:

RO.1 Development of a modular deep learning re-ranking recommender system for sparse session-based domains, focused on representation learning in a limited computational environment;

RO.2 Application of state-of-the-art processing techniques and methods, such as self-attention mechanisms to help capture user preferences and intent from behavioral interaction sequences;

RO.3 Implementation of an automatic bayesian optimization process for the model's architecture, subject to a highly conditional hyperparameter space.

To more explicitly evaluate the proposed objectives, each one is associated with complementary research questions (RQs), which guide Chapter 5's result discussion:

RQ.1 How does the model perform both ranking and classification-wise?

- RQ.1.1 What is the influence of the various model components in the predictive task?
- RQ.1.2 How does its performance compare to that of simpler non-tunable baselines and challenge's leaderboard submissions?

RQ.2 Is the attention mechanism able to:

- RQ.2.1 Provide any performance benefit?
- RQ.2.2 Learn any meaningful sequence-based patterns?

RQ.3 How does the bayesian optimization process:

- RQ.3.1 Compare to a random space search?
- RQ.3.2 Impact the optimization results?

1.4 Thesis outline

The remaining document is organized into five different chapters, as follows. Chapter 2 provides a brief overview of the core recommendation concepts, while delineating the research field's evolution and enumerating various modern approaches, including their focuses and limitations, supported by relevant literature. Chapter 3 delves into deep learning, describing the theoretical background behind the structural units and mechanisms used, their learning processes, optimization strategies and evaluation procedures. The following Chapter 4 consists of the methodology approach taken to tackle the recommendation problem, applying knowledge introduced in previous chapters to the dataset processing and model development. The recommender's performance is then modularly evaluated and compared against baselines in Chapter 5. Finally, Chapter 6 includes a few concluding points and enumerates interesting topics to take into consideration in future work.[4]

[4] **Future work display note** Future work items from Section 6.2's list are referenced throughout the text using (FW) tags.

Chapter 2

Recommender Systems

This chapter provides a brief overview of the core recommendation concepts, expanding on some of the previously introduced topics and contextualizing modern approaches with literature review based on related work and traditional foundations.

2.1 Traditional recommendation

The traditional recommendation problem is typically formulated on the basis of a utility matrix $\boldsymbol{R}$, with each entry r_{ui} corresponding to the interaction of a user u with an item i [15]. In this setting, the interaction variables often correspond to explicit preferences, such as ratings, or discretized representations of implicit feedback, usually in binary form (see Section 2.2). The objective is to extrapolate unknown interaction information from the known entries (i.e., complete the matrix), with the predicted interaction scores serving as proxies for user interest over the relevant items. Constrained on the inherent matrix sparsity attributed to the fact that most users on a platform have not interacted with most available items, and cold start situations regarding new users and items without interaction histories, recommendation strategies are generally divided into three main categories: content-based, collaborative filtering and hybrid methods [6, 15].

2.1.1 Content-based

Content-based approaches recommend items with features similar to those of other items a user has previously interacted with. To this end, item profiles consisting of either feature vectors manually extracted from auxiliary attributes (metadata, for instance), or more automated distributed representations (embeddings)[1], are created. The profiles of a given user's interacted item set can be combined using an averaging method to obtain a vector representation for that same user. A nonparametric approach such as nearest-neighbors with an arbitrary similarity measure is then often used between this representation and unevaluated candidate item profiles to retrieve a recommendation list.

This process is able to recommend new and unpopular items, cater to specific niches, and provide explanations based on the relevant item features. Nevertheless, basing the user profiles solely on

[1] For documents, the classic information retrieval term frequency-inverse document frequency (TF-IDF) encoding over a set of relevant words is a popular choice [11].

previous interactions leads to overspecialization, greatly limiting the item discovery potential of the system. Additionally, the non-trivial selection of appropriate comparison features and metrics also constitute important issues.

2.1.2 Collaborative filtering

Collaborative filtering (CF) methods were introduced with the ability to recommend items to users based on other users' preferences and item interactions, addressing some of the content-based shortcomings, namely the overspecialization and feature dependence aspects [15].

Memory-based CF still requires the designation of a similarity measure. Cosine and Jaccard similarities, the Pearson correlation coefficient, or values obtained with co-occurrence methods, in which item similarity is proportional to sequential interaction proximity, are often used. Item-to-item approaches use these measures to compare item profiles, which are given directly by their corresponding utility matrix interaction score vectors (columns, following the previous definition for $\boldsymbol{R}$) instead of the previously considered feature-based embeddings. The prediction of an interaction value for an unknown user-item pair consists of first retrieving a given number of other items most similar to the relevant item from the user's previous interaction set, usually accomplished with clustering or nearest-neighbor techniques. The user's interaction scores on those items are then combined with an averaging method to obtain the final value. Item-based strategies have been shown to outperform the alternative dual user-to-user processes, where predictions are derived from the preferences of most similar users instead [16].

The need to predict a large number of interaction scores to produce recommendation lists, using expensive pairwise similarity computations in sparse high dimensional spaces results in scalability issues. This led to the wide adoption of the more efficient and flexible latent factor models, which also make use of previously neglected interdependencies among users and items.

Latent factor models Popularized during the Netflix Prize competition [17], a pivotal event for recommender systems research that took place between 2006 and 2009, latent factor models such as matrix factorization (MF) achieved considerably more efficient and better performing CF solutions.

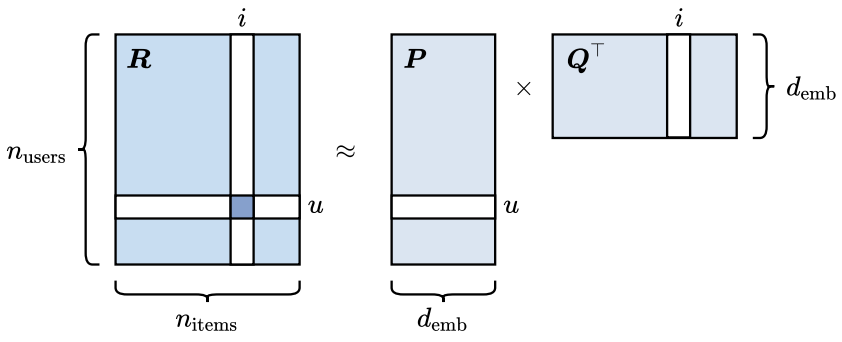


Figure 2.1: Matrix factorization representation, adapted from [18]. Reconstruction of the utility matrix $\boldsymbol{R}$ with the product of latent user and item matrices, $\boldsymbol{P}$ and $\boldsymbol{Q}$ respectively.

The challenge, which consisted of accurately predicting the last few movie ratings for a set of users, promoted the concept of utility matrix reconstruction, usually framed as a supervised learning problem. A test set of known ratings is omitted, and the predictions over their values are compared and optimized against the true withheld labels during training (in this case, minimizing the root-mean-squared error (RMSE)). MF tackles this problem with a singular value decomposition (SVD) variation suitable for the sparse setting [15]. The process provides a low-rank utility matrix approximation given by the product of two dense, lower-dimensional latent matrices representing the user and item space, such that each item and each user is described by an embedding vector, as represented in Figure 2.1. The latent matrices are generally optimized using a gradient descent method and new predictions can be made by taking the dot product between the corresponding user and item embeddings.

Although the necessity for restrictive manual feature selection is removed in these approaches, their inability to incorporate additional query and item information also results in the impossibility of recommending new items, a clear drawback compared to content-based alternatives. Likewise, the cold start situation regarding new user profiles is also not solved.

2.1.3 Hybrids

Still within the scope of the Netflix Prize, base MF solutions were rapidly modified to address some of the drawbacks mentioned above. The inclusion of global and local effect modeling, with the addition of specific biases, pointed towards the benefits of personalization. Methods such as SVD++ [19] generate new predictions with

$$\hat{r}_{ui} = \mu_r + b_u + b_i + \boldsymbol{p}_u \cdot \boldsymbol{q}_i, \tag{2.1}$$

where μ_r is the overall mean item rating, b_u is the rating deviation of user u from the global rating average, b_i is the rating deviation of item i, and $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ are the respective user and item embeddings. This form can be extended with supplementary terms providing content and neighborhood-based factors, combining the advantages of both traditional approaches [20]. Other information can further be used to solve recurring issues such as profile initialization with demographic or geographic data. The incorporation of temporal signals in the updated timeSVD++ [21], which accounted for different rating behaviors over time (including, for instance, the tendency of older movies to get progressively higher ratings), became a crucial component of the model ensemble that ended up obtaining the 10% RMSE decrease required to win the competition [22], further promoting the idea of item utility being highly variable and context-dependent.

Since then, MF extensions with pairwise ranking optimization and bayesian probabilistic concepts [23, 24] have gone on to obtain even better results in the matrix completion setting. Subsequently, the need for more flexibility in the modeling of additional, structurally variable features led to the development of various architecture solutions including Factorization Machines [25], Gradient Boosting Machines [26–28], Markov Decision Process models [29], and other hybrid strategies not explored in this work, which focuses exclusively on deep learning applications (expanded in Section 2.3).

2.2 Biased explicit and implicit feedback

The perception of users' interests on a platform is a defining aspect of the recommendation process. Systems based uniquely on explicit feedback, requiring users to express their preferences through likes, ratings or reviews, for example, promote sparser, hardly scalable environments since most of the user activity is passive.[2] Furthermore, users are also more likely to engage explicitly with items at extremes of their satisfaction spectrum [11].

These constraints have shifted the research focus to the modeling and incorporation of the significantly more abundant and easily collectible implicit signals in modern systems [10, 11]. This information enables the creation of algorithms capable of leveraging noisy user engagement and behavioral activity patterns such as clicks, views, purchases, and other interactions that indirectly but more consistently reflect their preferences. Additionally, as mentioned in the last section, the usage of these features is key to more accurately portray the circumstances defining the variable item utility, providing good foundations for engagement eliciting strategies [10]. The notion of indirect preference is important and stems from the fact that true item utility can only be guessed when dealing with implicit activity. Clicks, for example, do not necessarily represent positive interactions (although they can be accounted as such depending on the recommendation objective) as users might click items aimlessly or out of curiosity only to find them to be uninteresting after the fact. It is known that the complexity of the decision-making process, influenced by past behavior, present context, and the user's objectives might lead them to interact without awareness as to why exactly [8]. The necessity for robust models is extended by the possible intent changes during sessions, attention drifts and lack of negative feedback supporting the prediction of future interactions which all add to the challenge surrounding recommendations.

User interaction data is also inherently biased towards the system in which the interactions took place, causing feedback loops when used to train new models [30–33]. For instance, users are more likely to click higher-ranked items sometimes independently (to a certain degree) of their actual interest in them or of the query context. This type of bias is only one of many, most of which are unknown, variable, unpredictable, and exacerbated by data collection limitations and unobservable factors. As expressed in [34], the best way to deal with such biases is still an open question. In regard to the mitigation of presentation bias, a possible approach consists in the input of the presented items' positions to the model as a feature, with this information being removed at testing (either using an out-of-vocabulary token or, more typically, by selecting every item to be at the top position at serving) [34, 35]. Decreasing the weight of highly reachable item interactions in training and considering only sporadic clickstream information as test data to reduce the impact of bias towards popular items is also proposed in [33]. The impacts of these changes are, however, hard to evaluate offline and their study is more suitable for online settings where A/B testing is possible.

[2] The explicit interaction distribution is usually long-tailed, with a small portion of the users being responsible for most of the contributing likes, ratings, reviews or survey answers.

2.3 Related work

As companies began gravitating towards engagement and retention maximization objectives over rating RMSE-based solutions, the matrix completion formulation[3] started to be put aside in favor of learning to rank-oriented approaches [36]. Offering more flexibility, these methods generate ranked predictions of K items most likely to be interacted with in the near future or after a given action (top-K), with the rank being determined by probabilities, absolute or relative values, taking into account the factors discussed in the previous sections. Most modern recommenders tend to adapt the framework represented in Figure 2.2, sometimes introducing a prior candidate generation phase to filter larger item corpora and posterior re-ranking stages to further refine the recommendation process (typically using more focused signals conditioned on previously ranked items that allow for control over factors such as diversity, fairness, novelty and freshness[4] [37]).

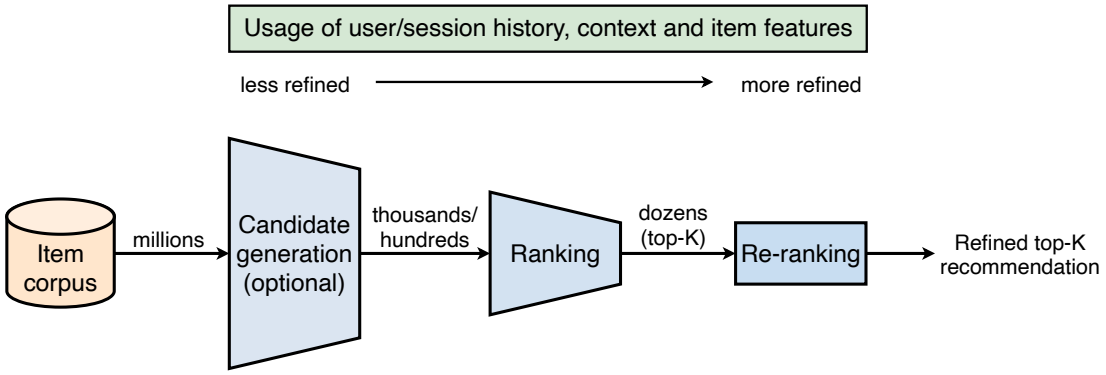


Figure 2.2: Modern recommendation framework, modified from [7].

The success deep learning has had in a wide variety of domains over the last decades, including computer vision and natural language processing, has greatly influenced the architectures of recommender systems in recent years [6]. Its ability to model complex non-linear user-item interaction patterns combined with the capacity to learn joint representations of multi-modal structured and unstructured data [38], allowing for the incorporation of various contextual signals, have been major contributing factors to the continuous increase of neural network-based state-of-the-art models.

2.3.1 Neural generalization of traditional methods

One of the most prevalent neural recommendation approaches consists in the generalization of more traditional methods, such as those reviewed above. [39], for instance, replaced the inner product, used to approximate interaction data entries in MF, with a neural architecture able to learn an arbitrary function from the data. He et al. [40] expanded on this idea with a hybrid model that combined a linear kernel generalized matrix factorization layer with a deeper non-linear architecture to predict CF implicit feedback scores based on user and item input vectors. This joint-training of

[3] Important disadvantages include, for instance, sequential signal and variable context modeling difficulties due to single user-item pair considerations.

[4] As with biases, the impact of these factors is more easily deduced in online settings, as some require trade-offs from typical offline metrics (e.g., accuracy and diversity) [33].

linear and non-linear structures had been previously introduced in the Wide & Deep model [41], which input several raw and transformed continuous and categorical contextual features into a single layer neural network - Wide - and a Multilayer Perceptron (MLP) - Deep - simultaneously, to learn both memorization and generalization from low and high-order feature interactions. Following the same basis, [42] complemented linear and deep neural structures with a parallel Factorization Machine-inspired Compressed Interaction Network, allowing it to learn high-order feature interactions both explicitly and implicitly at the vector-wise level.

Covington et al. [7] developed a two-stage system for large-scale recommendation which dramatically improved A/B results at YouTube and strongly impacted some of this work's processing and modeling decisions. Its dual MLP-based structure consists of an extreme multiclass classification with nearest-neighbors phase (candidate generation) followed by a more refined and contextualized regression to produce a top-K video ranking based on watch time prediction scores. Nevertheless, while much more informative than simply considering a single user-item interaction pair as in most CF methods, the input of past interaction information provided limited sequential awareness with the loss of temporal ordering, consequence of the interacted item embeddings' averaging considered.

2.3.2 Sequential recommendation

The sequential recommendation problem [29, 43] encompasses methods that thoroughly explore the temporal dependencies and modeling of the evolving sequentially ordered implicit and explicit signals contained in user activity logs, leveraging information neglected by most traditional approaches to aid in the prediction of future behavioral trajectories. Due to its unmatched sequence processing ability in a variety of tasks, the use of Recurrent Neural Network (RNN)-based architectures in this domain is naturally ubiquitous.

Sequence-based recommendation Settings where user profile information is available can exploit both dynamic long-term user-specific attributes across sessions and short-term information to build robust predictive representations.

Wu et al. [44] complemented low-rank factorization, which traditionally assumes static user and item profiles, with individual RNN autoregressive architectures for both users and movies in a rating prediction problem. This allowed the model to dynamically adapt to exogenous (award effect) and endogenous (age effect) temporal changes, instead of in a purely reactive way, greatly outperforming fixed bias-induced shifts. [45] used an alternative convolutional sequence embedding approach to learn union-level sequential patterns with skip behaviors, something earlier Markov Chain (MC)-based approaches, such as [46, 47], failed to accomplish. Unlike RNNs, for instance, MC models are unable to capture intricate dynamics of more complex, longer dependency scenarios [29] and their performance is also known to decline in sparse environments [43].

Focusing on context-aware recommendation, [48] developed a latent feature crossing technique to counter the inefficiency of feedforward neural networks in modeling low-rank multiplicative relations. This technique was then used to introduce various contextual signals, a considerable amount

of which temporal-based, into an improved RNN-based sequential YouTube recommender.

The tendency towards attention mechanism incorporation in the architecture of modern systems is clear, and the benefits which span from increased recommendation interpretability to better sequential modeling are detailed in Section 3.4. [49], for example, modeled a hashtag recommendation task as a multiclass classification problem, based on tweet topic representations learned by a Long-Short Term Memory (LSTM) with pre-trained word vectors. The last LSTM hidden state was used to attend over the input sequence and attribute different contribution weights to each word, achieving better results than pooling methods.

Although they fall outside the scope of this thesis, recent works such as [34, 50] have begun tackling multi-objective recommendation with attentional mixture-of-expert models, capable of leveraging different temporal ranges and dynamics depending on the request context to deal with short-term and long-term dependencies and predict independent probabilities for different possible user actions (FW.1). Newly proposed solutions that drop the need for recurrent units in favor of attention-only (transformer-based) models are also extremely promising and constitute the most recent shift in recommender research [51–54] (FW.2).

Session-based recommendation Sparser session-based settings, such as the one studied in this work, where individual user information is limited or completely unavailable and cold-start scenarios are frequent (greatly limiting the performance of user-dependent neighborhood and matrix factorization methods), have been shown to benefit significantly from short-term intent representations derived from implicit sequential interaction patterns.

Hidasi et al. [55] created the foundation for top-K next-item session-based neural recommenders by introducing a second order RNN architecture with negative output sampling to manage the large output space in the RSC15 clickstream dataset [56], GRU4Rec. Gated Recurrent Units (GRUs) were found to outperform the more complex LSTM units with numerous one-hot encoded item ID input sequence variations. The specialized pairwise ranking loss functions introduced in their work were later found to promote vanishing gradients and were modified in [57], which also showed a comparatively good performance from an altered categorical cross-entropy log loss. Tan et al. [58] further improved the GRU-based model with embedding dropout, an adaptation to temporal changes and data augmentation and processing techniques that were also implemented in this work. A direct embedding output similar to that of [7]'s in candidate generation was proposed with cosine similarity instead but a high-dimensional softmax cross-entropy approach was still found to perform better. [59] developed a hierarchical RNN model, in which a top-level predicts initializations for a lower-level GRU modeling the sequential session information. If user identifiers are available, the top-level can easily relay the evolution of each user's interests over time and across sessions.

Simultaneously, the incorporation of additional context features besides past item ID interaction sequences has also shown to be crucial in session-based recommendation, leading to performance gains over the last mentioned solutions. [60] developed an architecture with three parallel GRU-based structures to process item IDs with descriptive image and text feature vectors simultaneously.

Smirnova and Vasile [61] proposed a conditional RNN model and considered supplemental temporal and event type features, with different stage context injection at the input, unit dynamic and output structural levels.

As in sequence-based settings, works including [62, 63] have consolidated the importance of attention in the more challenging session-based environments. Because interest drifts that lead to redundant clicks are relevant in shorter timespans, [62]'s hybrid attentional encoders (with global and local focus) emphasize the user's main session purpose in item click sequences, producing a unified session representation used to compute click probabilities for available items, trained with a categorical cross-entropy loss. Liu et al. [63] built upon this solution, introducing higher attention dependency with a memory priority model, more capable of deriving user intent from smaller sessions. The use of transformers has started to become equally central in these domains, with [64], for instance, outperforming previous recurrent non-attentional approaches in next-item recommendation (FW.2).

2.3.3 Difficulties and limitations

Most modern recommendation approaches, including the ones mentioned earlier in this section, are concerned with the production of ranked item lists from the entirety or majority of the item corpus, corresponding to the first ranking stage of Figure 2.2. This objective differs from that of the challenge studied in this work (Section 1.2), resulting in the inability to directly compare the presented methods to the developed solution. The item list conditioning experienced by the re-ranking process can be easily lifted, and adaptations to the high-dimensional output space, using hierarchical softmax [65], negative sampling [66], or output embedding generation with nearest neighbors [7, 58], for instance, can be quickly implemented. However, unlike in the prevalently studied e-commerce domain, where inter-category product exploration can occur in an almost frictionless manner, the travel domain's extreme location dependence combined with the lack of item-specific location metadata in the RSC19 dataset prevented full corpus-based tests.[5]

Nevertheless, problems in recommender systems research extend far beyond these unaligned objectives. The large variety of public and private datasets with endless variations, evaluation metrics, validation procedures, and baselines used in the field has left researchers struggling to measure progress, questioning state-of-the-art advances. Recent works, including [67], have found that the lack of standardization and solid benchmarking has led to most datasets only being equally considered in a very small amount of relevant papers.[6] Metrics defining the performance of developed algorithms can range from the traditional information retrieval Precision and Recall to the Normalized Discounted Cumulative Gain (NDCG), Mean Average Precision (MAP), Click Through Rate (CTR), and the previously introduced RMSE and MRR, among others [33, 67]. These, matched with another assortment of validation processes, are often chosen based on past usage without additional

[5] In most cases, it would not make sense for hotels in Dubai to show up in a London-based search session. Furthermore, in RSC19, the same items can appear associated to dozens of different search cities, complicating location-based grouping.

[6] Some of the most popular benchmarking datasets, such as MovieLens, consist of explicit interactions but are often used to evaluate implicit-based algorithms without clearly defined objectives apart from increased accuracy.

justification, usually with also arbitrary list cutoff sizes (for example MRR@K, for which only the K top-ranked items are considered). Most importantly, maximizing offline user-centric objectives such as CTR might not even reflect the intended business-centric purposes of the system (i.e., additional revenue) [33]. After the Netflix Prize, for instance, Netflix's priorities rapidly changed as they realized that there are much better ways to recommend videos than "focusing only on those with a high predicted star rating" [11].[7] The challenge's contribution to their recommender had also mostly been limited to the resulting matrix factorization advances, as the accuracy gains of the winner solution were deemed not being justifiable of the engineering effort needed to apply them in a production environment [68]. Likewise, YouTube preferred a general unified deep learning pointwise system over more complicated ensemble or better offline performing systems for efficiency and scalability reasons [7]. Furthermore, crucial hyperparameter optimization process information is usually omitted, with its code and that of data preprocessing, validation, evaluation, and baseline implementations being rarely shared, resulting in serious reproducibility issues that hinder the comparative process [67, 69].

Deep learning applications are also affected by these drawbacks and the subfield's lack of direction is concerning. [67, 69, 70] found that "computationally and conceptually" simpler alternatives including association rule, nearest-neighbor, and even popularity aggregator methods could outperform algorithms such as NCF [40] or base GRU4Rec [55].[8]

[7] Netflix's focus shift towards recommendation diversity, user awareness, explainability, social integration and freshness was in great part driven by their business model change from DVD mailing to streaming [68].

[8] Although in offline conditions more comparable to data science competitions where popularity bias, small corpus coverage and scalability issues inherent to most can be disregarded.

Chapter 3

Deep Learning

The various attempts to emulate the human brain's plasticity, cognitive learning and pattern recognition ability, among others over the years, resulted in the development of numerous promising algorithms and computational structures. Although still inspired by the biological neuron's architecture, interconnectivity, and segmented specialization, the resulting artificial neuron-based models diverged to pursue empirical evaluation performance improvement [71].

This chapter provides an overview of the machine learning subfield, specifically concentrating on the different units and mechanisms used to develop the probabilistic classification recommendation model explored in this work, in addition to the theory behind its learning and evaluation processes.

3.1 Multilayer perceptrons

The archetypal neuronal model developed by McCulloch and Pitts [72] was improved by the perceptron [73], represented below in Figure 3.1, which first enabled the training of a single artificial neuron, a unit whose output is a function of the learned biased weighted sum of a given input vector, defined by[1]

$$y = g(\boldsymbol{w} \cdot \boldsymbol{x} + b), \tag{3.1}$$

where $\boldsymbol{x}$ is the input vector, $\boldsymbol{w}$ is the trainable weight vector, b is the bias term and g is the, often non-linear, activation function.

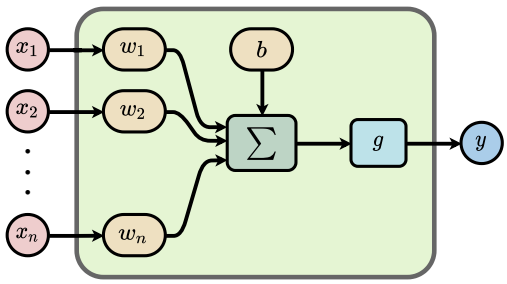


Figure 3.1: An artificial neuronal unit with n-dimensional input $\boldsymbol{x}$ and single output y.

In its strictest sense [74], the perceptron's input is a binary vector and its activation function is a

[1]The dot product $\boldsymbol{w} \cdot \boldsymbol{x} \equiv \sum_{i=1}^{n} w_i x_i$ can also be written as $\sum_{i=0}^{n} w_i x_i$, with $x_0 = 1$ to incorporate the bias term.

modified Heaviside step function[2], $g = H(\cdot)$, enabling it to learn a binary linear classifier,

$$y = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

The term perceptron is, however, commonly used to describe similar architectures with real-valued input vectors and different activation functions, some of which are displayed in Appendix Figure A.1, such as the sigmoid neuron, $g = \sigma(\cdot)$, and the now more prevalent rectified linear unit, $g = \mathrm{ReLU}(\cdot)$.

Artificial neurons can be combined to form directed, noncyclic, weighted networks called Multi-layer Perceptrons (MLPs), with each inter-nodal connection representing a coarse analogue to the biological (neuronal) axon and dendrite-bridging synapses, and each corresponding weight symbolizing the influence between end nodes. The most common way to arrange neurons is in fully connected layer structures [75], where the output $\boldsymbol{h}^{(k)}$ of a given layer k is a function of the preceding one, $\boldsymbol{h}^{(k-1)}$. Equation 3.1 can be generalized for an $L-$layer network with:

$$\boldsymbol{h}^{(k)} = g^{(k)}(\boldsymbol{W}^{(k)\top}\boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)}), \tag{3.3}$$

where $\boldsymbol{W}^{(k)}$ is the interconnection weight matrix and $\boldsymbol{b}^{(k)}$ the bias vector, associated with layer $k \in \{1, ..., L\}$ [38]. With this notation, $\boldsymbol{h}^{(0)}$ corresponds to the input layer $\boldsymbol{x}$, and $\boldsymbol{h}^{(L)}$ to the output layer $\boldsymbol{y}$. A simple single hidden layer network is represented in Figure 3.2.

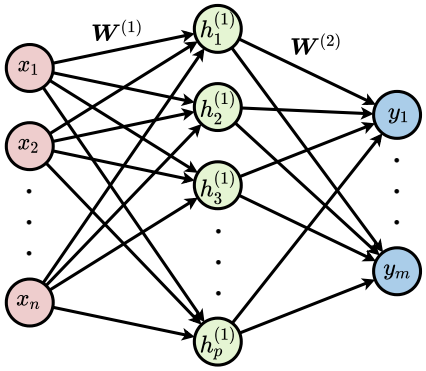


Figure 3.2: A two-layer MLP (input layer not considered) with n-dimensional input $\boldsymbol{x}$, single hidden layer $\boldsymbol{h}^{(1)}$ with p units, and m-dimensional output $\boldsymbol{y}$. Neurons in adjacent fully connected layers are fully pairwise connected but no interconnections exist within the same layer. In general, although only one is represented in this figure, a network might have an arbitrary number of stacked hidden layers with varying number of units.

An MLP defines a mapping $\boldsymbol{y} = f'(\boldsymbol{x}; \boldsymbol{\theta})$ and learns the value of parameters $\boldsymbol{\theta}$, which comprise the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$, that better approximate a given function f with a gradient-based optimization process [38]. It is known that a neural network with at least one hidden layer and a reasonable choice of non-linearity can approximate any continuous function on a compact input domain to arbitrary accuracy, given sufficiently large number of units [38, 71, 76]. This universal approximation

[2] with $H(0) = 0$.

property is, however, not enough to guarantee that the desired function is learned, as the challenge relies on obtaining the best parameter values to model a given task (assuming properly representative data), as is explored in the following sections of this chapter. Another powerful attribute of neural networks is their ability to combine simple concepts and learn complex abstract representations from raw, structured and unstructured data, alleviating the need for hand-designed features [38].

3.2 Recurrent neural networks

Neural networks are not limited to feedforward architectures characterized by the unidirectional flow of information, such as that observed in MLPs. Recurrent Neural Networks (RNNs) introduced cyclic connections (feedback loops) allowing for parameters to be shared across the model, constituting an optimal specialized architecture to process sequential data [38, 76]. While traditional fully connected MLPs attribute different parameters to each input feature, in RNNs the same weights are shared across time steps[3], allowing information to persist in its internal state. Specifically, most RNNs calculate a hidden state $\boldsymbol{h}^{(t)}$, at time step t, as

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}), \tag{3.4}$$

a function of the previous hidden state $\boldsymbol{h}^{(t-1)}$ and the current input vector $\boldsymbol{x}^{(t)}$, parametrized by $\boldsymbol{\theta}$. Furthermore, the processing can be done on variable length sequences in a variety of different structural design patterns, including those presented in Figure 3.3. This results in unprecedented flexibility in the use of context information as these networks are able to learn to store and ignore selective past information and recognize sequential patterns in the presence of sequential distortions [77].

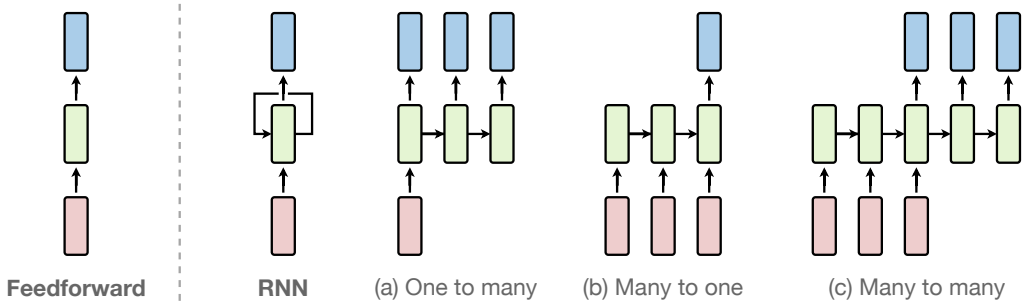


Figure 3.3: Unlike the fixed-sized one-to-one mapping provided by MLPs, RNNs are able to model tasks with sequential outputs (for example image captioning) (a), sequential inputs (like sentiment analysis) (b), or both (machine translation, for instance) (c). This last many-to-many setting can be expanded to output a vector at every input time step. Adapted from [78, 79].

Standard RNNs, however, suffer from a serious inherent long-term dependency flaw. The range of past relevant context that can be accessed with these architectures is quite limited, due to the tendency of gradients to either vanish or explode exponentially as they are propagated over many

[3]As expanded in [38], time steps need not denote the passage of time in the real world and might refer, for instance, to the index positions of sequence's input vectors. When specifically used with time-based sequences, the network may have connections that go backward in time, maintaining causality given the entire sequence is observed beforehand [77].

cyclic stages, providing little directional information to the cost function optimization (Section 3.3.3) and leading to unstable learning, respectively [38, 77, 80]. To overcome this problem, second-order[4] unit (cell) architectures such as the Long Short-term Memory (LSTM) [81] were introduced. In LSTMs, the information can flow along a central cell state, regulated by three gating mechanisms composed of self-connected memory cells (sigmoid neural network layers) and multiplicative units [77, 78]. These gates provide in-cell continuous analogues of the write, read and reset operations, allowing for the storage and access of information over long periods of time.

This work will focus on the application of Gated Recurrent Units, a simpler LSTM variant, found to be better performing in most of the sequential recommendation literature [6, 43].

Gated Recurrent Units Gated Recurrent Units (GRUs) [82], represented in Figure 3.4, essentially combine the LSTM's forget and input gates into an update gate, and merge the cell and hidden states [78]. The hidden state update, which results in the hidden state $\boldsymbol{h}^{(t)}$ at time step t, given input $\boldsymbol{x}^{(t)}$ and previous hidden state $\boldsymbol{h}^{(t-1)}$, is calculated with the following equations:

$$\boldsymbol{z}^{(t)} = \sigma(\boldsymbol{W}_z \cdot [\boldsymbol{h}^{(t-1)}; \boldsymbol{x}^{(t)}] + \boldsymbol{b}_z), \tag{3.5a}$$

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{W}_r \cdot [\boldsymbol{h}^{(t-1)}; \boldsymbol{x}^{(t)}] + \boldsymbol{b}_r), \tag{3.5b}$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh(\boldsymbol{W}_h \cdot [\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}; \boldsymbol{x}^{(t)}] + \boldsymbol{b}_h), \tag{3.5c}$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{z}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{z}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}, \tag{3.5d}$$

where the symbol $\circ$ denotes the Hadamard (element-wise) product and $[\cdot\,;\,\cdot]$ the vector concatenation operation. In Equation 3.5b, $\boldsymbol{r}^{(t)}$ is the reset gate vector, which allows the hidden state to drop any irrelevant past information and is used to produce a candidate state $\tilde{\boldsymbol{h}}^{(t)}$ in Equation 3.5c. The update gate, given by $\boldsymbol{z}^{(t)}$ in Equation 3.5a, has control over how much the past and candidate hidden states contribute to the current hidden state update given by Equation 3.5d.

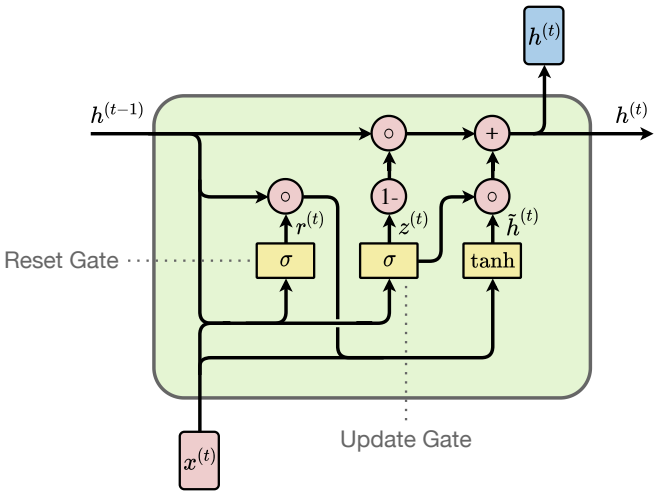


Figure 3.4: GRU unit's hidden state update at time step t, adapted from [78, 83].

[4]Second-order units include multiplicative interactions between internal states.

3.3 Learning

In most machine learning algorithms, the learning process for a specific task is directly tied to the interaction between a model and an optimization procedure to minimize a given performance-based cost function over a dataset.

In a supervised setting, learning is dependent on several provided pairs of evaluated input points $\boldsymbol{x}$ and respective label or target values y that constitute a training set $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}^{(i)}_{\text{train}}, y^{(i)}_{\text{train}})\}_{i=1}^{n} = \{(\boldsymbol{X}_{\text{train}}, \mathbf{y}_{\text{train}})\}$. In the particular case of a deep learning supervised classification task, the goal is to drive a function $f'(\boldsymbol{x}; \boldsymbol{\theta})$ created by a neural network to approximate the classifier $y = f(\boldsymbol{x})$, that maps the known noisy data sample input points to the respective ground truth category labels, such that the model is then able to predict the labels of new, previously unobserved data points [38]. This is accomplished by defining a cost function, measuring the performance or compatibility between the predictions and true labels, and optimizing it over the training set to find an optimal parameter vector $\boldsymbol{\theta}$ which jointly maximizes the mapping performance on a separate test set with the unobserved data $\mathcal{D}_{\text{test}} = \{(\boldsymbol{X}_{\text{test}}, \mathbf{y}_{\text{test}})\}$, representative of the model's generalization, after the learning process is over [76, 77].

3.3.1 Probabilistic multiclass classification

A possible approach to tackle a classification task is to define the mapping created by the parametric network as a conditional probability distribution $p(y|\boldsymbol{x}; \boldsymbol{\theta})$. To represent this probability distribution over K possible target classes, for the specific case of multiclass classification[5], the network can be constructed to output a prediction vector $\hat{\boldsymbol{y}}$, with $\hat{y}_i = P(y = i|\boldsymbol{x}; \boldsymbol{\theta})$, $\hat{y}_i \in [0, 1]$ and $\sum_i \hat{y}_i = 1$, using a final ($K-$unit) linear layer $\boldsymbol{z}$ followed by the softmax activation function:

$$\boldsymbol{z} = \boldsymbol{W}^{\top} \boldsymbol{h} + \boldsymbol{b}, \tag{3.6a}$$

$$\hat{y}_i = \text{softmax}(\boldsymbol{z})_i = \frac{\exp z_i}{\sum_j \exp z_j}, \tag{3.6b}$$

where $z_i = \log \tilde{P}(y = i|\boldsymbol{x}; \boldsymbol{\theta})$ consists of unnormalized log[6] probabilities and $y, i \in \{1, ..., K\}$ [38].

3.3.2 Cost function

Most neural networks are trained using the principle of maximum likelihood [38]. That is, an estimation for the parameters $\boldsymbol{\theta}$ is obtained by maximizing the likelihood $\mathcal{L}(\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{X}) = p(\mathbf{y}|\boldsymbol{X}; \boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} P(\mathbf{y}|\boldsymbol{X}; \boldsymbol{\theta}), \tag{3.7}$$

where the data points correspond to the training set, $(\boldsymbol{X}, \mathbf{y}) = (\boldsymbol{X}_{\text{train}}, \mathbf{y}_{\text{train}})$. Assuming the training examples to be independent and identically distributed (i.i.d.), this estimation can instead take the

[5] A one-vs-all multiclass setting is considered, where each sample is matched with a single possible output class.
[6] $\log$ denotes the natural logarithm.

form of the minimization of a cost function $J(\boldsymbol{\theta})$, consisting of the negative conditional log-likelihood expectation,

$$J(\boldsymbol{\theta}) = -\mathrm{E}_{\boldsymbol{x},y\sim\hat{p}_{\mathrm{data}}}[\log p_{\mathrm{model}}(y|\boldsymbol{x};\boldsymbol{\theta})]. \tag{3.8}$$

Minimizing Equation 3.8 corresponds to minimizing the cross-entropy (Equation A.7) between the empirical distribution defined by the training set $\hat{p}_{\mathrm{data}}$ and the probability distribution defined by the model p_{model} [38].

With maximum likelihood estimation, the cost function for the multiclass softmax output defined above, in Section 3.3.1, can be obtained by substituting Equation 3.8's probability distribution with Equation 3.6b, evaluated at the label index y,

$$J(\boldsymbol{\theta}) = -\mathrm{E}[\log \mathrm{softmax}(\boldsymbol{z})_y]. \tag{3.9}$$

While dedicated ranking cost functions exist to optimize directly for the ranking task via predicted output scores, classified as pointwise, pairwise and listwise in the learning to rank framework [36], they were not directly explored in this work (FW.3). Pairwise and listwise functions can be used to potentially introduce more diversity into the recommendations but pointwise approaches are often preferred for online serving due to greater efficiency and scalability [34]. The negative log likelihood is, in essence, a pointwise cost function, but its definition with the softmax function efficiently introduces useful listwise properties [57].

3.3.3 Gradient-based training

One of the challenges of training neural networks is that their non-linearity causes most cost functions to become non-convex. Consequently, the cost function optimization process is usually done iteratively, using gradient-based methods which do not guarantee optimal convergence [38]. Most often, a variation of Stochastic Gradient Descent (SGD) is used, which requires computing the cost function's gradient with respect to the parameters,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}), \tag{3.10}$$

where the expectation from Equation 3.8 is written as an average of the per-example loss L, given, in the multiclass case define above, by the negative log-likelihood from Equation 3.9,

$$L(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\log \mathrm{softmax}(\boldsymbol{z})_y. \tag{3.11}$$

The loss' gradient with respect to the parameters $\nabla_{\boldsymbol{\theta}} L$ is calculated using backpropagation (backprop) [84]. A network's forward pass for an input data point $\boldsymbol{x}$ returns the prediction $\hat{\boldsymbol{y}} = f(\boldsymbol{x};\theta)$. Backprop then obtains the gradients with respect to the weights and biases, $\nabla_{\boldsymbol{W}^{(k)}} L$ and $\nabla_{\boldsymbol{b}^{(k)}} L$ respectively, by iteratively applying the derivative chain rule backwards through the network for all the hidden layers, $k = (l, l-1, ..., 1)$, starting with the output loss gradient (error signal) $\nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, y)$.

In RNNs, the gradients are calculated with backprop after first unfolding their computational graph, with a slightly modified algorithm called backpropagation through time (BPTT) [38, 77].

SGD further assumes that the gradient is an expectation that may be approximately estimated using a small set of i.i.d. examples at each step of the algorithm[7], called a minibatch:

$$B = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(b)}, y^{(b)})\}, \tag{3.12}$$

where the minibatch size b is usually selected to be a power of two, with smaller values possibly offering a regularizing effect [38, 85]. More specifically, starting with small, random initialization weights [86], the gradient at each step is computed as

$$\hat{\boldsymbol{g}} \leftarrow \frac{1}{b}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{b} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}). \tag{3.13}$$

The parameter vector estimate is then updated by following the negative gradient,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\hat{\boldsymbol{g}}, \tag{3.14}$$

where η is the learning rate.

This algorithmic basis is used in various other optimization algorithms, such as the popular Adam [87], which uses an adaptive learning rate and momentum strategy, improving convergence by more efficiently avoiding local minima with an inertia-based motion through the loss space [77].

3.3.4 Bayesian hyperparameter optimization

Similarly to other machine learning algorithms, neural networks have hyperparameters that influence the algorithm's behavior and performance, such as the previously introduced learning rate or the number of layers and units. Unlike the network parameters $\boldsymbol{\theta}$, however, these hyperparmeters are not learned during the training process and must be chosen *a priori* to maximize a given objective function.

If tuned on the training set, some of the hyperparameters, especially those controlling the model's capacity, would tend to values that would consequently result in overfitting [38] (see Section 3.3.5). To prevent the tuning or any decision about the model's structure from impacting the overall generalization evaluation, a set of unobserved data points, separate from the test set examples, must be used. Therefore, the original training set is sub-divided into a new training subset used to learn the parameters, and another subset used to update the hyperparameters based on the estimated generalization performance of the model over its data points, called the validation set: $\{(\boldsymbol{X}_{\text{train}}, \mathbf{y}_{\text{train}})\} = \{(\boldsymbol{X}_{\text{subtrain}}, \mathbf{y}_{\text{subtrain}}), (\boldsymbol{X}_{\text{val}}, \mathbf{y}_{\text{val}})\}$.[8]

As the size of the available hyperparameter set grows, the need for automated tuning algorithms

[7] In traditional SGD (online learning) only a single example is used at a time.

[8] Generally, when a validation set is required, 'training set' refers to the resulting training subset and not the original unsplit data.

becomes more relevant over manual methods. A direct generalization of manual tuning is grid search, which samples and evaluates all combinations of predefined sets of discrete values for each hyperparameter, with a single hyperparameter varying at a time. The combination that yields the best objective performance result is chosen after several iterations or when a convergence criteria is satisfied, given the search space is not exhausted beforehand. Nevertheless, due to its exponential computational cost growth with the increase in the number of hyperparameters, this method is only suitable for simple hyperparameter spaces (preferably with available prior insight into what range of values might achieve good generalization performance). Random search, in which the hyperparameter values are randomly sampled from predefined continuous distributions that allow for a broader exploratory range, has been shown to converge much faster to better results [88]. The gain in efficiency can become exponential in the common cases where the usually expensive to evaluate objective function is unaffected by some hyperparameters. An additional alternative is to use model-based optimization. Bayesian optimization [89], in particular, allows for information about previously evaluated hyperparameter combinations to influence future sampling decisions and has been shown to outperform random search in various settings, including highly conditional spaces[9] [90].

Bayesian optimization Bayesian Optimization (BO) considers the problem of finding a global maximizer of an unknown objective function f:

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x}), \tag{3.15}$$

where $\mathcal{X}$ is some possibly conditional design space of interest over which the input hyperparameter configuration $\boldsymbol{x}$ is optimized. The continuous black-box function f is typically expensive to evaluate and is assumed to only be observable through noisy point-wise observations

$$y = f(\boldsymbol{x}) + \epsilon. \tag{3.16}$$

In this setting, the hyperparameter optimization involves designing a sequential iterative strategy which successively maps previously collected data to following query points.

The *a priori* beliefs about probable values of f, before any data is observed, may be captured in a prior distribution $p(f)$. Given set $\mathcal{D}_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, a collection[10] of input points $\mathbf{x} = \boldsymbol{x}_{1:n}$ with corresponding noisy observations $\mathbf{y} = y_{1:n}$, and a likelihood model $p(\mathbf{y}|\mathbf{x}, f)$, the prior may be written $p(f|\mathbf{x})$ and a posterior distribution $p(f|\mathbf{y}, \mathbf{x})$ representing updated beliefs about f after observing the data can be inferred using Bayes' rule (A.3):

$$p(f|\mathbf{y}, \mathbf{x}) = \frac{p(\mathbf{y}|\mathbf{x}, f)p(f|\mathbf{x})}{p(\mathbf{y}|\mathbf{x})}. \tag{3.17}$$

The denominator normalization term $p(\mathbf{y}|\mathbf{x})$, called the evidence or marginal likelihood distribution,

[9] Spaces with hyperparameters dependent on particular conditions or on the values of other (parent) hyperparameters.
[10] The set notation $z_{i:j} = \{z_i, ..., z_j\}$ from [89] is used.

can be obtained by marginalizing (A.1) f out of the numerator, with the usually intractable integral

$$p(\mathbf{y}|\mathbf{x}) = \int_f p(\mathbf{y}|\mathbf{x}, f)p(f|\mathbf{x})\, df. \tag{3.18}$$

Gaussian processes Gaussian processes (GPs) [91] are non-parametric models $\mathrm{GP}(\mu_0, k)$, fully characterized by their prior mean function μ_0 and positive-definite kernel, or covariance function, k, commonly used to obtain the posterior distribution, inferring confidence intervals on f [89]. GP inference assumes that the objective function values $f(\mathbf{x}) = \{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_n)\}$, associated with the previously defined n inputs $\mathbf{x}$, are random variables, such that any finite collection of which is jointly Gaussian [91, 92]. That is, with a GP on f,

$$f(\boldsymbol{x}) \sim \mathrm{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')), \tag{3.19}$$

defined by mean and kernel functions

$$m(\boldsymbol{x}) = \mathrm{E}[f(\boldsymbol{x})], \tag{3.20}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{Cov}(f(\boldsymbol{x}), f(\boldsymbol{x}')), \tag{3.21}$$

the prior distribution conditioned on the inputs is given by

$$f(\mathbf{x})|\mathbf{x} \sim \mathcal{N}(\mathbf{m}(\mathbf{x}), \mathbf{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)), \tag{3.22}$$

or expanded,

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_n) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_0(\boldsymbol{x}_1) \\ \vdots \\ \mu_0(\boldsymbol{x}_n) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix} \right). \tag{3.23}$$

The prior mean function is assumed to be constant and equal to zero from this point forward, $m(\boldsymbol{x}) = 0$, as is often done in the literature.

The prior described above is chosen because it is also assumed that the noise associated with an observation (from Equation 3.16) is given by an i.i.d. zero-mean Gaussian, $\epsilon \sim \mathcal{N}(0, \sigma^2_{\mathrm{noise}})$, yielding the likelihood model

$$\mathbf{y}|\mathbf{x}, f \sim \mathcal{N}(f(\mathbf{x}), \sigma^2_{\mathrm{noise}}\mathbf{I}), \tag{3.24}$$

where $\mathbf{I} \in \mathbb{R}^{n \times n}$ is the identity matrix. It follows that the prior is conjugate[11] [76, 91] and the numerator's joint distribution fully defines the posterior, which is given by another Gaussian process

$$p(f|\mathbf{y}, \mathbf{x}) \propto p(\mathbf{y}|\mathbf{x}, f)p(f|\mathbf{x}), \tag{3.25}$$

$$f(\boldsymbol{x})|\mathbf{x}, \mathbf{y} \sim \mathrm{GP}(m_{\mathrm{post}}(\boldsymbol{x}), k_{\mathrm{post}}(\boldsymbol{x}, \boldsymbol{x}')). \tag{3.26}$$

[11] The prior is said to be conjugate for the likelihood function if the posterior distribution is in the same probability distribution family as the prior, which is true in this case as the product of two Gaussians gives an unnormalized Gaussian (Equation A.12).

Given an arbitrary test point $\boldsymbol{x}_*$, the joint distribution of the marginalized observed target values and the function value at the test location under the prior can be written as Equation A.9, where the prior on the noisy observations is defined by $\mathrm{Cov}(\mathbf{y}) = \mathbf{K}(\mathbf{x}, \mathbf{x}) + \sigma^2_{\text{noise}}\mathbf{I}$ due to noise independence,

$$\begin{bmatrix} \mathbf{y} \\ f(\boldsymbol{x}_*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}(\mathbf{x}, \mathbf{x}) + \sigma^2_{\text{noise}}\mathbf{I} & \mathbf{K}(\mathbf{x}, \boldsymbol{x}_*) \\ \mathbf{K}(\boldsymbol{x}_*, \mathbf{x}) & \mathbf{K}(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix}\right). \tag{3.27}$$

The posterior mean and kernel functions[12] are then given by Equation A.11 and are subsequently used to select the next query point $\boldsymbol{x}_{n+1}$:

$$\mu_n(\boldsymbol{x}_*) = \mathbf{K}(\boldsymbol{x}_*, \mathbf{x})[\mathbf{K}(\mathbf{x}, \mathbf{x}) + \sigma^2_{\text{noise}}\mathbf{I}]^{-1}\mathbf{y}, \tag{3.28}$$

$$\sigma^2_n(\boldsymbol{x}_*) = \mathbf{K}(\boldsymbol{x}_*, \boldsymbol{x}_*) - \mathbf{K}(\boldsymbol{x}_*, \mathbf{x})[\mathbf{K}(\mathbf{x}, \mathbf{x}) + \sigma^2_{\text{noise}}\mathbf{I}]^{-1}\mathbf{K}(\mathbf{x}, \boldsymbol{x}_*). \tag{3.29}$$

Kernel function The kernel function k defines the similarity between data points, under the assumption that points with close input values are likely to have similar target values [91]. Consequently, it also defines the structure of the prior functions considered for inference [89].

One of the most commonly chosen classes of kernel functions is the stationary (dependent on $\boldsymbol{x}_i - \boldsymbol{x}_j$) *Matérn* class [91], whose functions are given by

$$k_\nu(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{2^{1-\nu}}{\Gamma(\nu)}\Big(\frac{\sqrt{2\nu}}{l}d(\boldsymbol{x}_i, \boldsymbol{x}_j)\Big)^\nu K_\nu\Big(\frac{\sqrt{2\nu}}{l}d(\boldsymbol{x}_i, \boldsymbol{x}_j)\Big), \tag{3.30}$$

with positive smoothness ν and length scale l parameters, where $d(\cdot, \cdot)$ is the Euclidean distance, $\Gamma(\cdot)$ is the Gamma function and $K_\nu(\cdot)$ is a modified Bessel function. A typical choice for ν is $5/2$ as used in [90], for which the functions are neither too rough (lower ν values) nor too smooth (higher ν values) and take the simplified form

$$k_{5/2}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Big(1 + \frac{\sqrt{5}}{l}d(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{5}{3l^2}d(\boldsymbol{x}_i, \boldsymbol{x}_j)^2\Big)\exp\Big(-\frac{\sqrt{5}}{l}d(\boldsymbol{x}_i, \boldsymbol{x}_j)\Big). \tag{3.31}$$

The length scale l is generally tuned by maximizing the log marginal likelihood (Equation 3.18), for which the Gaussian process model provides an analytical expression (type-II maximum likelihood) [89], but the process will not be explored further in this work. Also not explored was the possible usage of conditional kernels, which Lévesque et al. [90] introduced to obtain better optimization performance in conditional hyperparameter spaces (FW.4).

Acquisition function Instead of optimizing f directly, an easier to evaluate acquisition function α_n is chosen to define a strategy that maps the GP model to the next query point $\boldsymbol{x}_{n+1}$,

$$\boldsymbol{x}_{n+1} = \underset{\boldsymbol{x} \in \mathcal{X}}{\arg\max}\, \alpha_n(\boldsymbol{x}; \mathcal{D}_n), \tag{3.32}$$

[12] The functions generalize to arbitrary inputs with $m_{\text{post}}(\boldsymbol{x}) = \mu_n(\boldsymbol{x})$ and $k_{\text{post}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2_n(\boldsymbol{x}, \boldsymbol{x}')$.

based on the obtained posterior mean $\mu_n(\boldsymbol{x})$ and kernel $\sigma_n^2(\boldsymbol{x})$ functions, given by Equations 3.28 and 3.29, respectively. This strategy tries to greedily obtain information about f's maximum by attributing an exploration bonus to under-explored (high variance) regions [93].

The Gaussian Process Upper Confidence Bound (GP-UCB) algorithm [94], in which the acquisition function is given by,

$$\alpha_n(\boldsymbol{x}) = \mu_n(\boldsymbol{x}) + \beta_n^{1/2}\sigma_n(\boldsymbol{x}), \tag{3.33}$$

essentially an upper bound over the posterior controlled by the hyperparameter β_n [93], is commonly used. β_n functions as a tradeoff factor between exploitative (focus on $\mu_n(\boldsymbol{x})$) and exploratory (focus on $\sigma_n(\boldsymbol{x})$) optimization behavior [95]. As in the kernel function's case, this specific hyperparameter's tuning process is not explored.

The iterative process The overall iterative hyperparameter set optimization process detailed throughout this section is summarized in Algorithm 1.

Algorithm 1: Bayesian Optimization with Gaussian Processes, adapted from [89, 90].

$\mathcal{D}_1 \leftarrow \varnothing$ ▷ *An initial collection of points can be used instead of an empty set*
for $n \in 1, ..., N$ **do**
  fit GP on observations to obtain $\mu_n(\boldsymbol{x})$, $\sigma_n(\boldsymbol{x})$
  select new hyperparameters $\boldsymbol{x}_{n+1}$ by optimizing acquisition function α, $\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x}} \alpha_n(\boldsymbol{x}; \mathcal{D}_n)$
  query objective function to obtain y_{n+1}
  augment data $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\boldsymbol{x}_{n+1}, y_{n+1})\}$
  update statistical model
return model with best performing hyperparameter input

3.3.5 Regularization

The process of obtaining a good generalization performance on the test data is directly tied to the representational power of the model and to its training optimization process. Representational power or capacity encapsulates the amount of functions a given model can fit, which is proportional to its structural complexity or number of free parameters [38, 71]. In general, if its capacity is too large, a model can begin to fit noise instead of the intended feature relations and targets during training, and the function it learns cannot generalize well to data it has not seen before, in a process called overfitting. On the contrary, if the capacity is not large enough, the model underfits, meaning it becomes overly biased and can not obtain a suitable fit for the training data, which results in a similarly high test error, given the data was split properly.

The balance in obtaining both a small train error (given by the cost function) and a small gap between the train and test errors can be managed with regularization. In fact, to avoid overfitting, the application of regularization techniques in combination with higher capacity models is preferred over less complex models, as their associated loss function has, generally, better valued local minima (although their number can be much more substantial) [71].

Various regularization methods exist, from parameter norm penalties that constraint and encode priors into the weight values and penalize structural complexity (L_2 and L_1), to ensemble and noise injection methods [38], and more recently, batch [96] and layer [97] normalization. In this work, only early stopping and dropout were explored (FW.5). Proper feature representation, discussed in Section 4.2.4, can have additional regularizing effects though less explicitly.

Early stopping One of the simplest regularization methods can be applied by removing a part of the training set for use as a validation set to roughly estimate the model's performance on the test set during the learning process, as in the previous hyperparameter optimization section. When a model's representational capacity is enough to overfit the task, it is typically observed that while the training error decreases continuously, the error on the validation set starts to increase after a given point, signaling a loss in generalization. Early stopping consists in monitoring the validation error and returning the parameter values that minimize it during training, after a predefined number of full dataset pass iterations (epochs) without improvement over the lowest error value obtained.

The whole training dataset (without the validation set split) can then be used to train a re-initialized model until this estimated best-valued iteration [38].

Dropout Introduced by Srivastava et al. [98], dropout consists in randomly removing a subset of neurons (multiplying their outputs by zero) during each training forward/backward pass with a hyperparameter probability p_{drop} [75]. This is equivalent to training an exponentially-sized ensemble of all the parameter sharing sub-networks that can be formed by removing non-output units from the underlying base network [38] and can also be a first-order equivalent to an L_2 regularizer under certain conditions [99]. Dropout is not applied during testing and the network's output may be regarded as the average ensemble prediction. To keep the network's behavior well-defined the neurons' outputs must be scaled either during training or testing (inverted dropout) such that their expected output is the same in both instances [71].

3.4 Attention mechanisms

Originally motivated by human visual selective focus, attention mechanisms are joint-trained with a model and learn context vectors that weigh (attend to) different input elements based on their relevancy to the respective prediction task [100]. As such, attention can be leveraged to improve the interpretability of black-box models, enhance RNN memorization potential [101], and filter uninformative features from raw inputs, consequentially reducing the impact of noisier data [6]. These factors have strongly contributed to the mechanism's popularity among recent state-of-the-art recommendation approaches [29], whose implementations have been mostly derived from the following works.

Bahdanau et al. [102] first introduced attention to sequence-to-sequence (seq2seq) [103] Neural Machine Translation (NMT), allowing the encoder-decoder model to automatically search for parts of a source sentence relevant to the prediction of a target word, without having to encode the

source sentence into a fixed-length vector, a known bottleneck for longer sentences. Luong et al. [104] further expanded this approach with global (which accounts for all source positions) and local (only attends to a subset of source positions at a time) attention structures with dot product-based alignment scores.

Instead of using different source and target sequences, attention can be computed on a single sequence, inducing relations among its items. Self-attention was introduced in [105], which adapted the LSTM's cell architecture with an attention-based memory network to attend over input sequences, improving the processing of structured inputs in language modeling, sentiment analysis and natural language inference. It is common for these approaches to reduce the attention weights to a single vector. In the classification setting, Yang et al. [106], for instance, applied two hierarchical attention levels to attend over sentences and words in a document classification task. Both levels attend according to a randomly initialized general joint-learned vector, with an adapted version of [102]'s proposal. Their work demonstrated performance improvement over other complementary RNN mechanisms (output max-pooling and averaging) even in shorter sequences, with the additional insight into what sequence parts contribute the most to the classification decision.

More recently, Vaswani et al. [107] modified [104]'s dot attention to include a scaling factor, creating the basis for multi-head attention, the core block of the transformer architecture. Transformers are purely self-attention based, meaning they do not require RNNs for sequence processing, and their state-of-the-art results in NLP have inspired new applications in the recommendation space with promising results [54, 64], though out of this work's scope as had already been pointed in the state-of-the-art Section 2.3 (FW.2).

Definition Let $\mathbf{x} = (\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)})$ denote a source sequence of length n, and $\mathbf{y} = (\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(m)})$ a target sequence of length m. While the seq2seq terminology for source and target sequences is used, inspired by [102]'s original implementation, any two sequences may be used. The source sequence can be encoded into a forward source hidden state (annotation) vector $\overrightarrow{\boldsymbol{h}} = (\overrightarrow{\boldsymbol{h}}^{(1)}, ..., \overrightarrow{\boldsymbol{h}}^{(n)})$ by an RNN. A Bidirectional RNN (Bi-RNN) can instead be used to concatenate a backward hidden state vector to the forward one, by also processing the sequence in reverse, generating the annotation for item $\boldsymbol{x}^{(j)}$, $\boldsymbol{h}^{(j)} = [\overrightarrow{\boldsymbol{h}}^{(j)\top}; \overleftarrow{\boldsymbol{h}}^{(j)\top}]^\top$, which contains summarized information of both preceding and following items.[13] The target annotations $\bar{\boldsymbol{h}}$ can be similarly generated.[14]

The attention mechanism outputs a weighted sum of the source annotations, the context vector:

$$\boldsymbol{c}_i = \sum_{j=1}^{n} \alpha_{ij} \boldsymbol{h}^{(j)}. \tag{3.34}$$

The attention weights α_{ij} represent the alignments of output-input item pairs $(\boldsymbol{y}^{(i)}, \boldsymbol{x}^{(j)})$ [100]. They define the influence each source hidden state has on each output with the softmax function (Equation

[13] Every annotation $h^{(j)}$ contains information about the whole sequence with a strong focus on the parts surrounding the j-th item [102].

[14] The general structure of attention integration varies by task. In the case of [102]'s seq2seq model, for instance, the hidden state $\bar{\boldsymbol{h}}^{(i)}$ for a word at target position $i \in \{1, ..., m\}$ is conditioned on the attention output $\boldsymbol{c}_i$ and given by a decoder network such that $\bar{\boldsymbol{h}}^{(i)} = f_{\text{dec}}(\bar{\boldsymbol{h}}^{(i-1)}, \bar{\boldsymbol{y}}^{(i-1)}, \boldsymbol{c}_i)$.

3.6b),

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}, \tag{3.35}$$

$$e_{ij} = \text{score}(\bar{\boldsymbol{h}}^{(i)}, \boldsymbol{h}^{(j)}), \tag{3.36}$$

where e_{ij} is given by a predefined alignment score method.[15] The alignment score functions explored in this work are present in Table 3.1.

Table 3.1: Explored attention alignment score functions. Weight vectors $\boldsymbol{v}_a$, $\boldsymbol{u}_a$ and matrices $\boldsymbol{W}_a$, $\boldsymbol{W}_b$ are jointly learned during the training process (biases were integrated into the weight matrices). According to [106], $\boldsymbol{u}_a$ can be regarded as a high level representation of a fixed relevancy query to find the "informative sequence items" over the sequences, inspired by memory networks.

Name	Alignment score function	Eq.	Citation
Additive	$e_{ij} = \boldsymbol{v}_a^\top \tanh(\boldsymbol{W}_a \bar{\boldsymbol{h}}^{(i)} + \boldsymbol{W}_b \boldsymbol{h}^{(j)})$ The score function is parametrized by a single layer neural network.	(3.37)	Bahdanau et al. [102]
Dot (General)	$e_{ij} = \bar{\boldsymbol{h}}^{(i)\top} \boldsymbol{W}_a \boldsymbol{h}^{(j)}$	(3.38)	Luong et al. [104]
Self (General)	Variants of the presented above, where the target sequence is either the input sequence, a learnable weight or non-existent.	-	Cheng et al. [105]
Hierarchical	$e_j = \boldsymbol{u}_a^\top \tanh(\boldsymbol{W}_a \boldsymbol{h}^{(j)})$ Modified additive, where the attention weights are reduced to a single vector $\alpha_j = \text{softmax}(e_j)$.	(3.39)	Yang et al. [106]
Dot (Scaled)	$e_{ij} = (\boldsymbol{W}_a \bar{\boldsymbol{h}}^{(i)\top} \boldsymbol{W}_b \boldsymbol{h}^{(j)})/\sqrt{d_k}$ The expression above, where d_k is the input sequence's length, follows from the modified scaled dot attention usually represented with Query (Q), Key (K) and Value (V) sequence notation, $\boldsymbol{\alpha} = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V$. In a self-attention setting, they are either all the same, $Q = K = V = \boldsymbol{h}$, or different weighted projections of the input sequence. $d_k = \lvert K \rvert$.	(3.40)	Vaswani et al. [107]

3.5 Evaluation

The performance of a machine learning algorithm on a given task can be evaluated from various perspectives with different metrics that allow for comprehensive result analysis and easier model comparison, such as the already defined ranking-based MRR (Equation 1.1), the regression-based RMSE or classification accuracy, for instance. The following definitions are adapted from [38, 108].

In a classification setting, the classifier's performance over K classes can be partly visualized using a confusion matrix $\boldsymbol{C}_{K \times K}$, where the rows represent true instance classes and the columns represent predicted instance classes (or otherwise), i.e., c_{ij} corresponds to the number of observations of class i predicted to be in class j.

[15] In [102], $e_{ij} = \text{score}(\bar{\boldsymbol{h}}^{(i-1)}, \boldsymbol{h}^{(j)})$.

For a binary case with positive and negative samples, as represented below, the main diagonal entries correspond to the correct predictions, the true positives (TP) and true negatives (TN), while the remaining antidiagonal entries correspond to the false negatives (FN) and false positives (FP) or type II and I statistical errors, respectively.

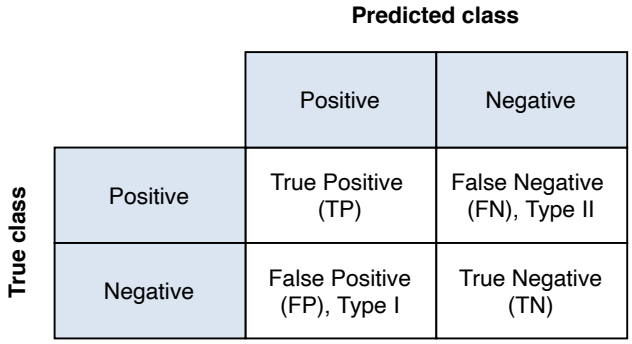


Figure 3.5: Confusion matrix for a binary setting with positive and negative sample classes.

Accuracy From this matrix, the value for classification accuracy, corresponding to the proportion of correct predictions from the total amount of samples, can be written as

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.41}$$

When in imbalanced settings with highly disparate proportions between classes, however, accuracy values might be misleading (in certain problems, for instance, exclusively selecting the most frequent class might return high accuracy values) and other metrics are required to better describe performance [38].

Recall The model's ability to find the positive instances can be evaluated using the recall, or True Positive Rate (TPR), given by the fraction of actual positive class instances identified correctly:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.42}$$

Precision Precision, the ability of the classifier not to label a negative sample as positive, is given by the fraction of correct positive predictions:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.43}$$

F1-score Recall and precision can be combined to form the F-score metric. Specifically, with equal contributions from both, given by their harmonic mean, the F1-score is obtained:

$$\text{F}_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.44}$$

Multiclass metrics While accuracy can be given more generally by

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} 1(\hat{y}_i = y_i), \tag{3.45}$$

the remaining metrics presented above can be extended to the multiclass setting by considering individual binary cases for each class and averaging the results to obtain an overall performance value. Different averaging methods[16] can be used depending on the objective:

- Macro averaging - unweighted mean of the individual binary metrics, tends to over-emphasize infrequent class performance. Macro recall is known as a type of balanced accuracy in the literature.
- Weighted averaging - accounts for class imbalance by weighting each binary metric in the average with the corresponding class support (number of class samples), thereby favoring the most frequent classes. Weighted recall corresponds to the classification accuracy.

[16] Micro averaging is sometimes also mentioned, weighing each sample-class pair equally. Its value for recall, precision and F1-score is the same and corresponds to the classification accuracy.

Chapter 4

Methodology

This chapter defines the framework used to tackle the proposed recommendation objectives, encompassing the data and feature processing implementation methodology, followed by the model generation process based on the previously defined methods and architectural units from Chapters 2 and 3.

General processing implementation was done in `Python 3.7.7`, on a 6-core 2.6GHz i7, 16GB RAM machine. Model implementation was done in machine learning library `TensorFlow 2.3.0` [109] and training was executed on Google Colab's free GPU mode with approximately 12GB RAM, subject to dynamic usage limits.[1]

4.1 Problem formulation

The re-ranking of the accommodation impression lists required by the challenge can be framed as a supervised learning, multiclass classification task solving the implicit click prediction surrogate problem introduced in Section 1.2.

To predict the clicked item's position at any given click event in a user session y, a deep learning model was designed to generate an array of click probabilities $\hat{\boldsymbol{y}}$ for the displayed impression list items, leveraging previous session-based sequential behavior signals and numerous types of additional context as input, further detailed in the following sections. These probabilities are used to calculate the negative log likelihood cost (NLL), with the maximum value's index contributing to the classification evaluation with the metrics introduced in Section 3.5. The ranking performance is given by the MRR of the sorted probability indices, $\boldsymbol{y}_{\text{rank}}$, as shown in the high-level modeling process for a session sample input displayed in Figure 4.1.

Formally, let $\boldsymbol{S} = (\boldsymbol{e}^{(1)}, ..., \boldsymbol{e}^{(n)}) \equiv (\boldsymbol{e}^{(\tau)})_{\tau=1}^{n}$ denote an arbitrary user session with n sequential events. Each of these events consists of an interaction a, from the ten possible types previously introduced in Table 1.2, with a reference item i at time t contextualized[2] by $\boldsymbol{c}$, such that the τ^{th} event can be represented as the tuple $\boldsymbol{e}^{(\tau)} = (a^{(\tau)}, i^{(\tau)}, t^{(\tau)}, \boldsymbol{c}^{(\tau)})$. In particular, a single session contains

[1] Including alternating allocation of Nvidia K80s, T4s, P4s and P100s, limited idle and virtual machine session time with restricted GPU and disk memory. In `https://research.google.com/colaboratory/faq.html`, last acc. 2020-02-03.

[2] For simplicity, $\boldsymbol{c}$ corresponds to a vectorial proxy for different time-dependent and independent contextual signals.

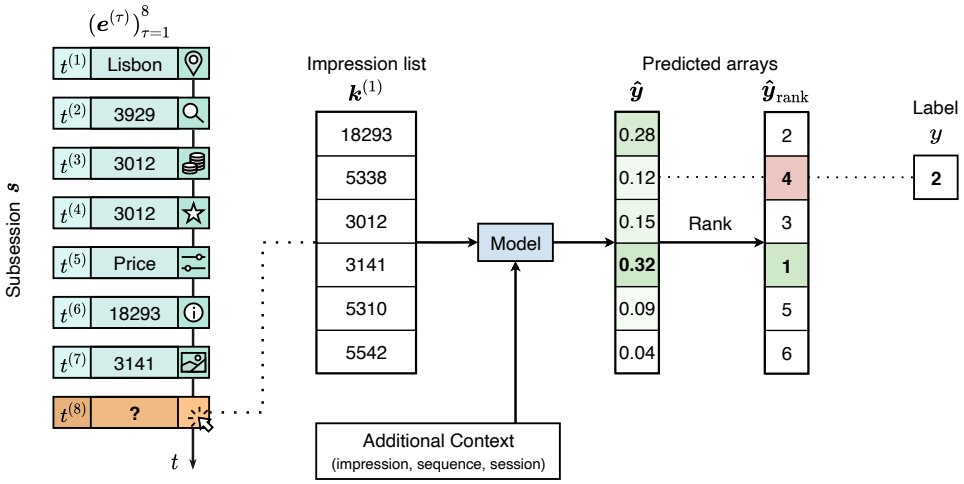


Figure 4.1: High-level simplified modeling process for the eight event session sample (subsession) s. Event action icon legend available in Figure 1.1. The model outputs a vector $\hat{y}$ with the click likelihood for each of the classes, which, in this case, consist of impression list $k^{(1)}$'s six item positions. A generalized transformed context input is represented, consisting of dynamic and static signals captured up to the click event's timestamp $t^{(8)}$ (unknown in the prediction), a combination of features explored in Section 4.2. The click probability vector is sorted to produce a position rank vector $\hat{y}_{\text{rank}}$. Since the predicted rank for the ground truth label y is 4, the Reciprocal Rank for this example is 1/4.

a subsequence of $w \in \{1, ..., n\}$ click events, $(e_h^{(\tau_h)} | a^{(\tau_h)} = \text{clickout})_{h=1}^{w}$, with $e^{(n)} = e_w^{(\tau_w)}$, i.e., the last session event corresponds to a click. Click events are crucial to the problem as they contain the impression lists presented to the users, $k^{(h)}$ (for every arbitrary $e_h^{(\tau_h)}$), which in turn hold the targets. Based on these, the model was trained to produce a sequential prediction for what item in the impression list is clicked at $t^{(\tau_h)}+\epsilon$, where ϵ is a small time interval, by calculating click probabilities for each one, $\hat{y}^{(h)} = P(y^{(h)} = k_p^{(h)} | x^{(\tau_h)}, \theta)_{p=1,...,|k^{(h)}|}$, as noted above.

To leverage every click's information, the training set was augmented with a vectorized implementation of the method used in [58, 62, 63], such that every session produced $w - 1$ additional subsessions (training samples) containing the events preceding each of the w session click events, $s^{(l)} = (e^{(\tau)})_{\tau=1}^{\tau_l}$, represented below in Figure 4.2.

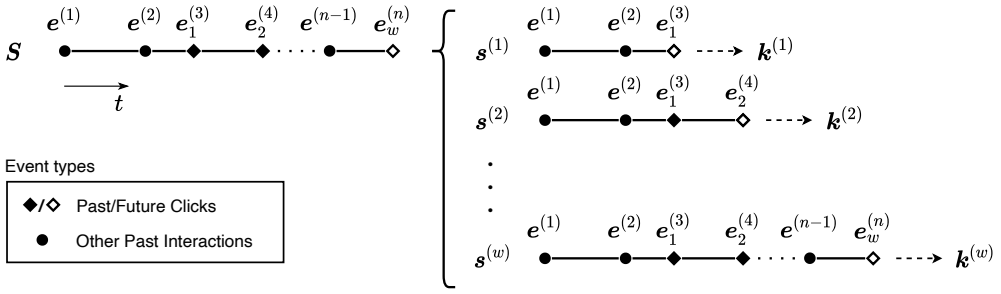


Figure 4.2: Augmentation process for session S, which generates $w - 1$ new subsessions $\{s^{(1)}, ..., s^{(w-1)}\}$, ($s^{(w)} = S$), based on the w click events, each associated with the corresponding impression list $k^{(h)}$.

The process described above was repeated for every session $S_{\forall j \in N}^{(j)}$ in the dataset $\mathcal{D}$, for which the session axis had been omitted in simplification.

4.2 Data processing

In most data-driven tasks, the processing phase ensures the creation of proper complete input representations, on which the performance of predictive machine learning algorithms is highly dependent [38, 77]. This section encompasses the rationale behind the development of the recommender's input $\boldsymbol{X}$, consisting of relevant features generated and extracted from the raw activity log dataset, such that the contextual signals and patterns depicting user intent are captured efficiently.

The `Pandas 1.1.0` data analysis library was used for data manipulation mainly due to its inclusion of DataFrames: structured multidimensional multi-labeled arrays that allow for heterogeneous data storage and database operations [110].

Display notes The mean and outlier[3] values for the distribution box plots presented in Figures 4.3, 4.4, 4.6, 4.7, 4.13 and 4.14, are represented by triangular and circular markers, respectively. Throughout the following sections, items presented in colored text boxes correspond to generated or extracted `features`.

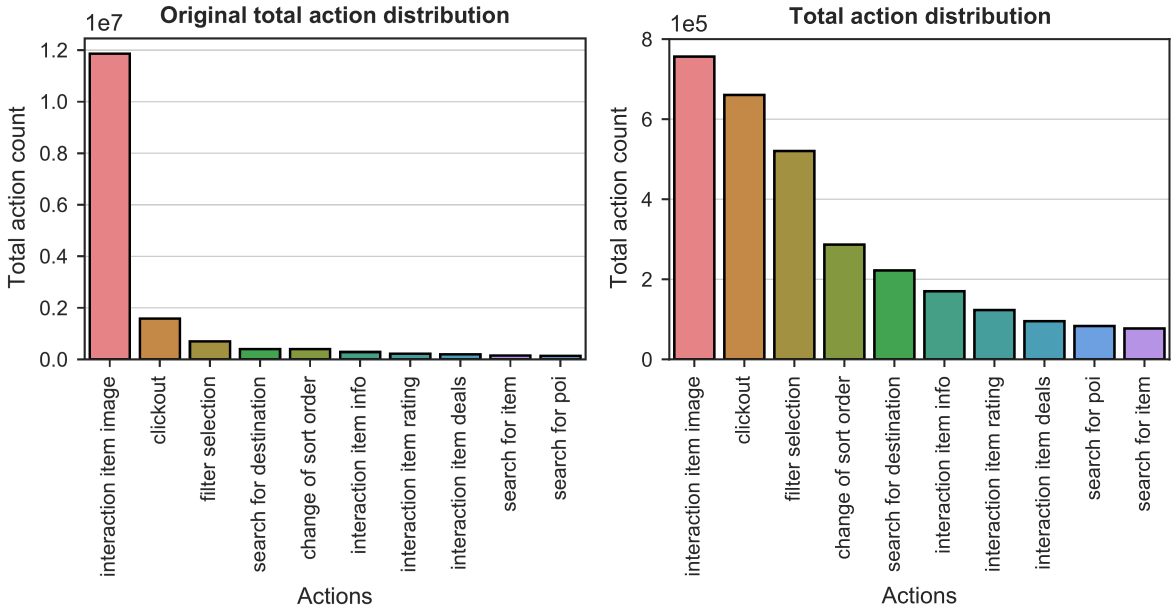
4.2.1 Preprocessing

The initial preprocessing stage consisted of general data structural modification, cleansing and filtering steps, mostly based on dataset exploration and visualization, as preparation for subsequent feature-based operations:

1. Duplicate log entries were deleted.
2. Invalid click events, where the clicked reference was not present in the impressions list, were dropped.
3. Helper counter features, `subsession` (incremented at time steps following click events, reset between sessions) and `substep` (incremented at each time step, reset between subsessions), were created, after sorting the data by time, to facilitate part of the subsession-based feature generation, providing the foundation for the data augmentation method described in Section 4.1.
4. Sessions/subsessions without clicks were removed.
5. User sort interactions were incorporated as filter selections, with undefined reference values being removed.
6. The original dataset's event distribution was dominated by image interactions (Figure 4.3a), which accounted for almost three quarters of the events largely due to similar consecutive actions with the same references. So as to not overpower the interaction sequences, the

[3] Following `matplotlib 3.3.0`'s default settings, the displayed plot outliers are defined as points with values less than $Q_1 - 1.5\,\mathrm{IQR}$ and greater than $Q_3 + 1.5\,\mathrm{IQR}$, where Q_1 and Q_3 are the lower and upper data quartiles, and $\mathrm{IQR} = Q_3 - Q_1$ is the interquartile range.

data was looped over to group these consecutive actions into single events, generating a `frequency` feature, corresponding to each group's size, along with temporal endpoints - the initial and final group events' timestamps, respectively. This process had a major impact in balancing the dataset's action distribution displayed in Figure 4.3b.

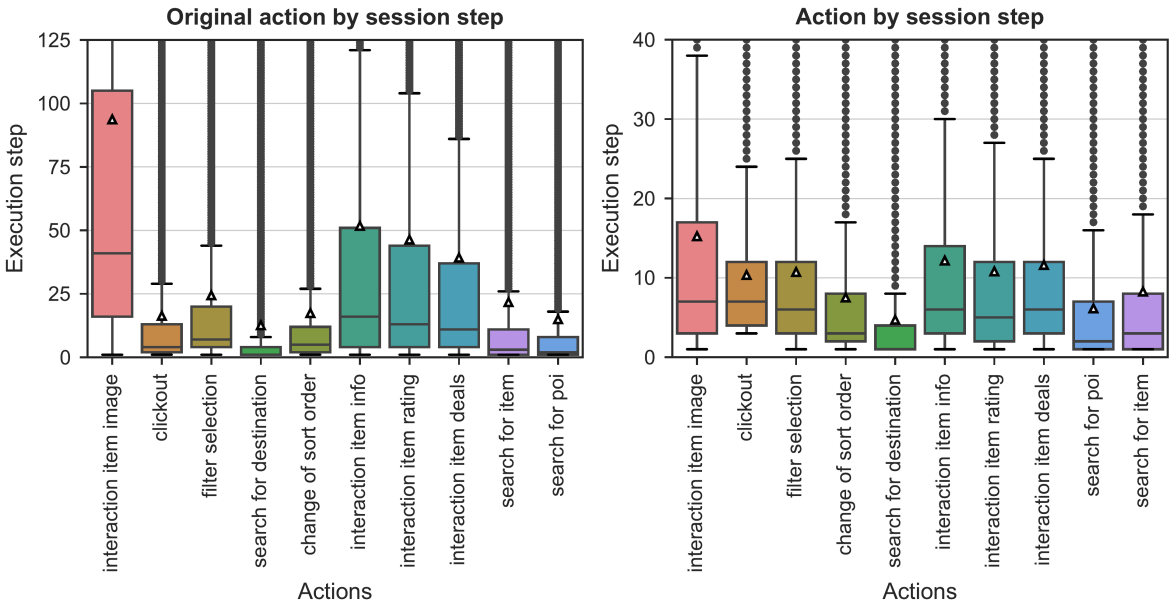


(a) Original dataset action count.

(b) Preprocessed dataset action count.

Figure 4.3: Original and preprocessed datasets' event count by action type. Preprocessing reduced the ratio of most to least frequent action count from $r_{\text{og}} = \frac{\texttt{\#image int.}}{\texttt{\#search poi}} = 86.3$ to $r_{\text{pre}} = \frac{\texttt{\#image int.}}{\texttt{\#search item}} = 9.8$.

7. Finally, sessions and first subsessions with less than three events were dropped. The implementation of a minimum time step threshold is not uncommon in the literature. Normally, as in [55, 61–63], sequences of length one are removed. In this case, while sequences of length one would also only introduce unwanted stochasticity by training the model on clicks without much relevant context besides that provided by the impression list characteristics, it was expectedly noted that the first session time steps mainly consisted of more general exploratory action types (Figure 4.4a), which when followed directly by a click would, once more, yield limited recommendation support. This is more clearly visible in Figure 4.5, in which action counts normalized for each time step, to account for the decreasing amount of interactions with the increase in session size, were plotted for the first 25 time steps. Non-item specific interactions like POI and destination searches or sort order changes demonstrate a decreasing relative count trend as the session progresses, with the latter accounting for 40% of the first event's total interactions. Therefore, in an effort to enhance the model's robustness, while still taking into account the primarily small sized original dataset's sessions (of which half had less than four time steps according to Figure 4.6), this method ensured the existence of at least two other actions with each click in every training example.



(a) Original dataset action distribution by session time step, zoomed in for clarity.

(b) Preprocessed dataset action distribution by session time step.

Figure 4.4: Original and preprocessed datasets' action distribution by session time step. The distribution's variance was reduced for most of the action types, with more exploratory-based ones maintaining a higher density towards the beginning of sessions.

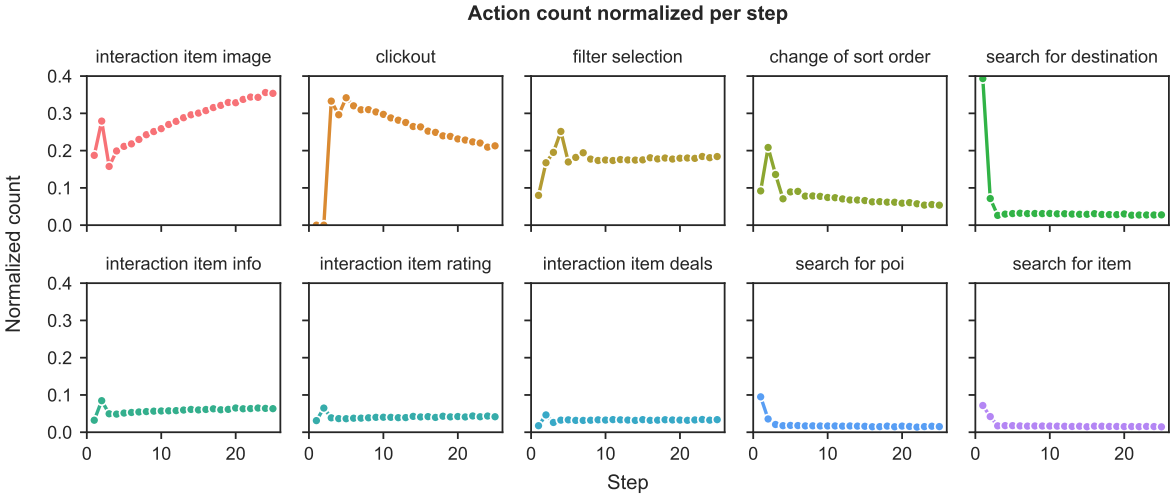


Figure 4.5: Normalized action count per time step. The action count values at each point sum up to one, showing the relative proportion of event types throughout a session. The sudden decrease in value for most of the action types in the third time step is explained by the imposed session/subsession size threshold.

The implementation of these preprocessing steps resulted not only in a less skewed distribution of time steps at which actions occur (Figure 4.4b), but also, unsurprisingly, in a more even action count distribution per session (Figure 4.7), expanding on what had been previously presented in Figure 4.3. The click distribution, unaffected due to their required presence in every sample for the prediction task, corresponds to that of subsessions per session for which the average count is two.

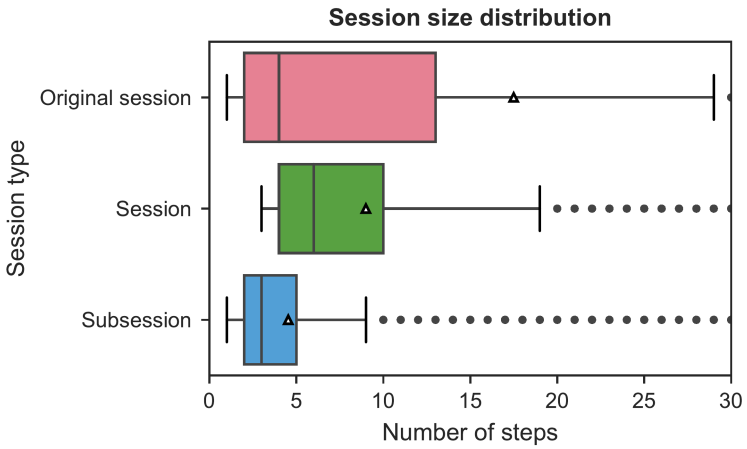
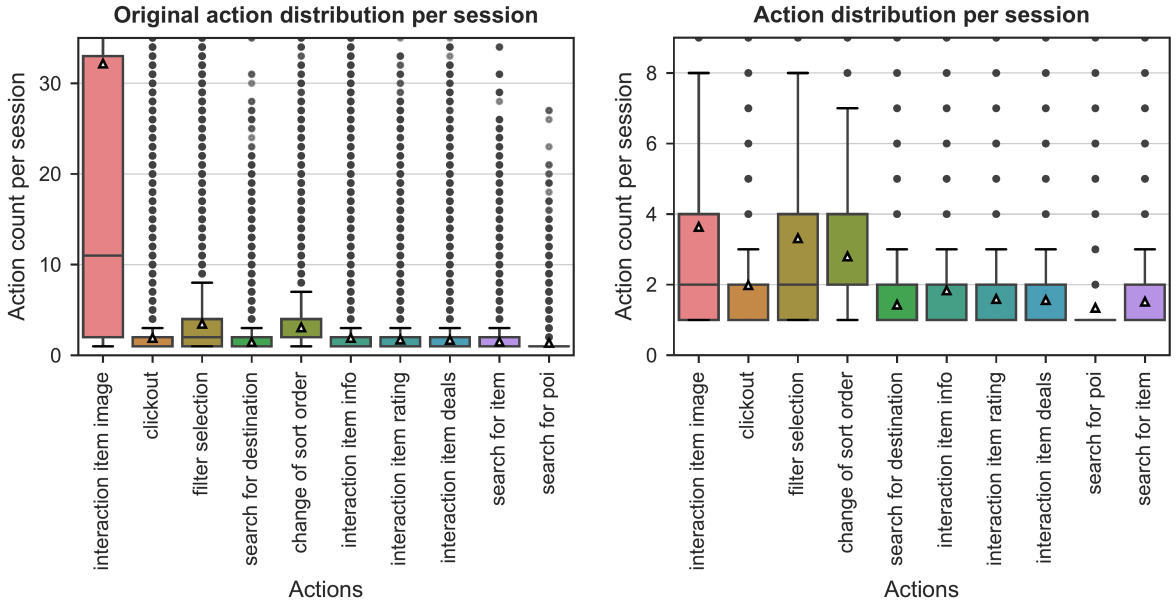


Figure 4.6: Session size (in time steps) distribution comparison between original and preprocessed datasets. The subsession distribution represented considered only non-overlapping events (i.e., number of time steps between consecutive clicks, including the last click event; using session S from Figure 4.2 as an example, the size value corresponding to subsession $s^{(2)}$ would be 2).



(a) Original dataset action distribution by session, zoomed in for clarity.

(b) Preprocessed dataset action distribution by session.

Figure 4.7: Original and preprocessed datasets' action distribution by session. Only sessions containing the respective actions contributed to the distribution, hence the inexistence of zero-valued data points. In reality, actions such as `interaction item rating`, `interaction item deals`, `search for poi` or `search for item` were rarely present in most sessions.

In the end, the original 910 683 user sessions were reduced by approximately 36% down to 332 849, containing 660 526 samples and 2 995 184 events. Processed sessions remained relatively small, with only 15% containing more than ten time steps, although their median event number increased from four to six.

4.2.2 Feature engineering

As mentioned at the beginning of Section 4.2, the predictive foundation necessary to model the noisy interaction patterns requires the extraction of meaningful representations from the heterogeneous preprocessed dataset, such that each training sample's information content is maximized [86].

Most data science competitions tend to promote feature engineering hyper-focus, with some of the best performing models generally making use of hundreds or even thousands of different carefully hand-designed and transformed inputs. While this might make sense when the emphasis is set on achieving the defining decimal points in the required offline evaluation metrics, it usually comes at the expense of real-life online applicability where generalization is important to keep up with rapidly changing objectives and tasks, as exemplified by the Netflix Prize's outcome [11].

In this work, a different, more balanced approach was taken, combining deep learning's knowledge representation ability and input robustness with limited, theoretically important feature generation focused on cross-domain information availability, based on literature guidelines. These features should help propagate important content and CF-based signals, while providing different types of contextualization surrounding the relevant click events, improving personalization and relieving the need for the model to learn the corresponding relations by itself [111]. Their performance impact was partly relayed to the optimization process, discussed in Section 4.3.1. The objective was to create a suitable modular baseline, which can then be complemented with additional multi-modal information in the future, without the need for ensemble training, and that can be easily adapted to different ranking tasks with minimal intervention.

The feature set was divided into three main different categories and two additional subcategories (expanded in Appendix Table B.1) according to their characteristics and behavioral structure across training examples:

- Interaction sequence features, $\boldsymbol{X}_{\text{seq}} \in \mathbb{R}^{n_{\text{seq}} \times m_{\text{seq}}}$, model dynamic sequential information across a session leading up to a given click event;

- Session features, $\boldsymbol{X}_{\text{ses}} \in \mathbb{R}^{m_{\text{ses}}}$, consist of static context signals that define each session/subsession sample;

  - Filter features, $\boldsymbol{X}_{\text{filt}} \in \mathbb{R}^{d_{\text{filt}}}$, comprise subsession-based search and sort filters selected by the users;

- Impression features, $\boldsymbol{X}_{\text{imp}} \in \mathbb{R}^{n_{\text{imp}} \times m_{\text{imp}}}$, describe and summarize properties and interactions for the impression list's items at every click;

  - Metadata features, $\boldsymbol{X}_{\text{meta}} \in \mathbb{R}^{n_{\text{imp}} \times d_{\text{meta}}}$, contain additional impression characteristics retrieved from the metadata database.

The sample axis was omitted such that the presented dimensions correspond to those of a single sample: n_{seq} is the maximum number of interacted sequence time steps, m_{seq} is the sequential feature space dimensionality; m_{ses} is the session feature space dimensionality, including the filters' embedding dimension d_{filt} (if active); n_{imp} is the maximum number of impression list items and m_{imp} is the impression feature space dimensionality, including the metadata attributes' embedding dimension d_{meta} (if active).

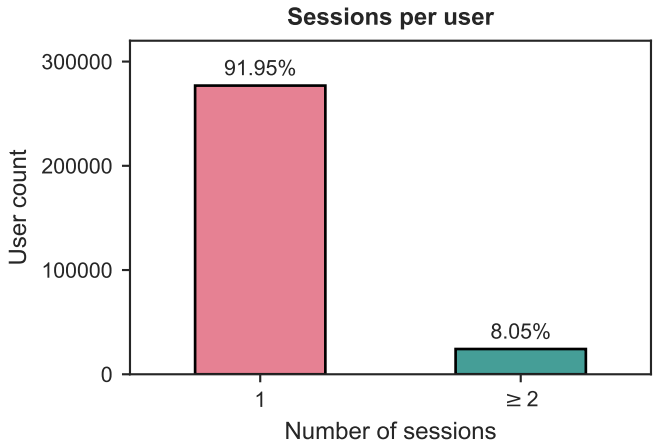


Figure 4.8: Number of sessions per user in the preprocessed dataset with multi-session value aggregation.

The inability to use additional, more specific, user-based features to aid in the recommendation task due to the overwhelming majority of single session users, one of the defining characteristics and biggest obstacles of session-based settings, can be observed in Figure 4.8. Nevertheless, the exploration of user-centered signals constitutes an interesting future work research topic for less sparse sequential recommendation, as the combination of long and short-term dependencies can be easily integrated into the model presented in the following section, allowing for an added layer of insight and personalization (FW.6).

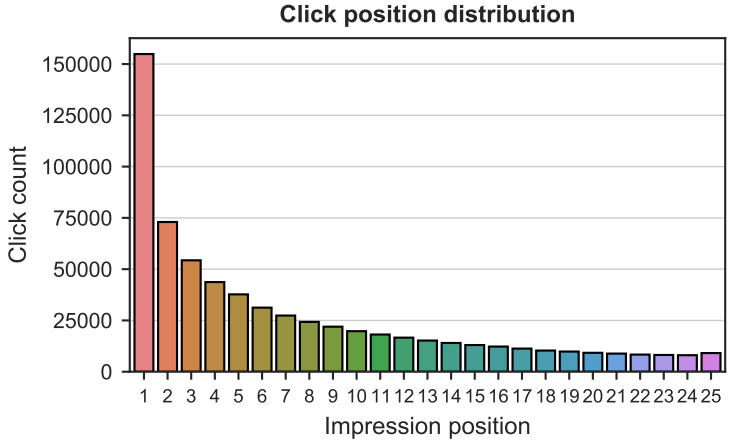


Figure 4.9: Imbalanced clicked item position distribution. Lower rank is equivalent to higher layout position.

Another fundamental dataset challenge concerns the impact of presentation bias (Section 2.2) on the distribution of clicked impression item positions, corresponding to the target labels, represented in Figure 4.9, resulting in a heavily imbalanced classification problem. Figure 4.10 additionally shows that this bias towards top ranked items is aggravated by shorter preceding interaction sequences, with almost a third of all clicks for sequences of length two occurring in the highest list position.

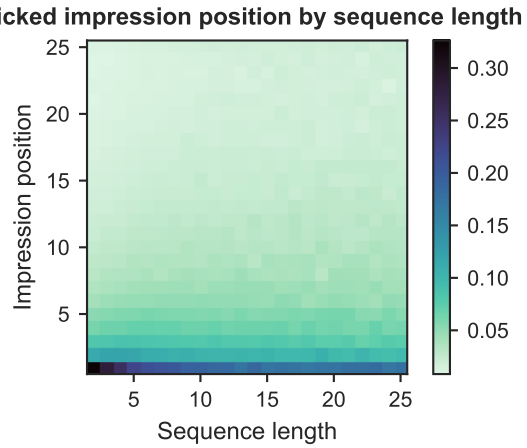


Figure 4.10: Normalized clicked impression item position distribution by past interaction sequence length.

Sequential features

The interactions leading up to the click events constitute the basis of users' preferences and provide insight into their purposes across sessions. As discussed in Chapter 2, the complexity of the intricate sequential pattern modeling task is aggravated by an array of factors including different relative interaction importance [45, 50], the events' implicit nature, the possibility of session intent change over time [44, 58, 62], and the sparsity of the domain defined by small windows of anonymized activity. Although the limited six day data span prevented the study of longer-term temporal dynamics like seasonality and user/item trend-based signals, known to be worth considering in other recommendation applications [44] (FW.7), the numerous short-term obstacles amplify the need for contextualization, which the following features seek to provide.

The sequence of items IDs a user has interacted with in a given session, `reference item ID`, constitutes the core log building block, and is the sole input considered in most session-based next interaction top-K models such as [55, 58, 62, 63].

Other features, namely the `action ID` of each event, central in Section 4.2.1, and its time `step` were extracted directly from the preprocessed dataset, along with the `frequency` values generated in Section 4.2.1.

Since different patterns of time gaps between past events were shown to have an important impact in session-based next item selection likelihood [61], in addition to the time elapsed at each time step since a given session's inception, `session time` (Equation 4.1a), the time delta between interactions [48], `time differential` (Equation 4.1b), was generated by looping over the dataset.

This concept was further expanded to contain the time elapsed since the previous action of each individual type with Equation 4.1c. For an arbitrary event $e^{(\tau)}$:

$$\Delta t_{\text{session}}^{(\tau)} = t_{\text{end}}^{(\tau)} - t_{\text{ini}}^{(1)}, \tag{4.1a}$$

$$\Delta t_{\text{dif}}^{(\tau)} = t_{\text{ini}}^{(\tau)} - t_{\text{end}}^{(\tau-1)}, \tag{4.1b}$$

$$\Delta t_{\text{actdif}_q}^{(\tau)} = \begin{cases} \sum_{j=(\tau_{\Omega_q}+1)}^{\tau} \Delta t_{\text{dif}}^{(j)} & \text{if } \Omega_q^{(\tau)} \neq \{\varnothing, a^{(\tau)}\}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.1c}$$

$$\Delta t_{\text{action}}^{(\tau)} = t_{\text{end}}^{(\tau)} - t_{\text{ini}}^{(\tau)}, \tag{4.1d}$$

$$\Delta t_{\text{dwell}}^{(\tau)} = \Delta t_{\text{action}}^{(\tau-1)} + \Delta t_{\text{dif}}^{(\tau)}. \tag{4.1e}$$

In Equation 4.1c, $\Omega_q^{(\tau)}$ stands for the previous action of type q with corresponding time step $\tau_{\Omega_q} < \tau$. The difference between an event's initial and final timestamps, t_{ini} and t_{end} respectively (products of the preprocessing frequency grouping), formed the `action time` (Equation 4.1d). Equations 4.1b and 4.1d can then be combined to form the `dwell time` (Equation 4.1e), as detailed in the temporal subsession diagram of Figure 4.11. This diagram also alludes to the impossibility of calculating `time differential` and `dwell time` features for the last subsession events, as that would require knowledge of the following clicks' timestamps, unavailable in a causal prediction setting.

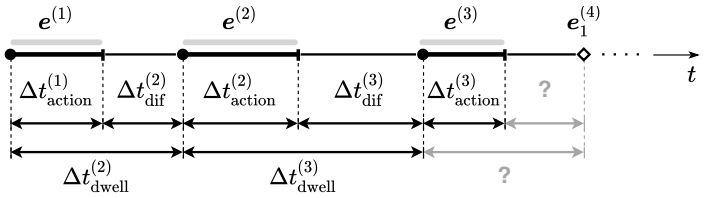


Figure 4.11: Temporal feature detail for a subsession with three interactions preceding the click event $e_1^{(4)}$. Each of the events is delimited by its initial and final timestamps, $t_{\text{ini}}^{(\tau)}$ and $t_{\text{end}}^{(\tau)}$. $\Delta t_{\text{dif}}^{(1)}$ and $\Delta t_{\text{dwell}}^{(1)}$ are undefined by definition, since $\tau \in \{1, ..., 4\}$. $\Delta t_{\text{dif}}^{(4)}$ and $\Delta t_{\text{dwell}}^{(4)}$ are unknown for the click prediction problem as $t^{(4)}$ can not be accessed without future temporal leakage, i.e., its use would result in a non-causal system.

To prevent data discontinuity, undefined values, such as differential and dwell times of first session events or action differential times of events that precede first session events of a given action type, were mapped to zero instead of an out-of-range value [112]. Additional binary feature vectors were created to indicate value existence in these scenarios.

Lastly, the dataset augmentation process introduced in Section 4.1, tasked with organizing the events into full subsessions, was finalized. The impossibility of iterating through the almost three million row DataFrame while keeping track of session blocks, due to computational limitations, required a vectorized approach capable of reordering and overlapping sequences without the need for memory-consuming auxiliary variables. As was done in [58, 62], a sequence threshold was set at 25 time steps preceding each click, given that 95% of the available sessions were smaller than this value, and

due to additional padding considerations later discussed in Section 4.2.4.[4] The developed solution combined `Pandas`' integer location indexing with row indices output by the function `vrange`[5], when given starting subsession and click event indices respectively, generating the final sequential feature DataFrame containing 6 066 328 event rows in 13.42s total runtime.

Session features

Regarding invariant subsession/sample features, the number of previous clicks in the session given by the `subsession` counter, the number of time steps since the previous session click event `substeps`, and aggregators for sequential features such as the total amount of session interactions or time `steps`, and the total `session time`, were lifted from the preprocessed dataset. These features and their corresponding sequential counterparts can provide helpful insight to shape the output probability space. Expectedly, lower-ranked search page items (with higher impression position) are often interacted with and clicked after those at the top, as supported by their associated higher average previous click, time step, and session time values displayed in Appendix Figures B.1, B.2, and B.3, respectively.

The variable number of possible output classes for each sample, defined by the impression list's length at every click, `impression length`, was created. This feature's value frequencies, plotted in Figure 4.12, showed that less than a quarter of presented lists were smaller than the possible maximum of 25.

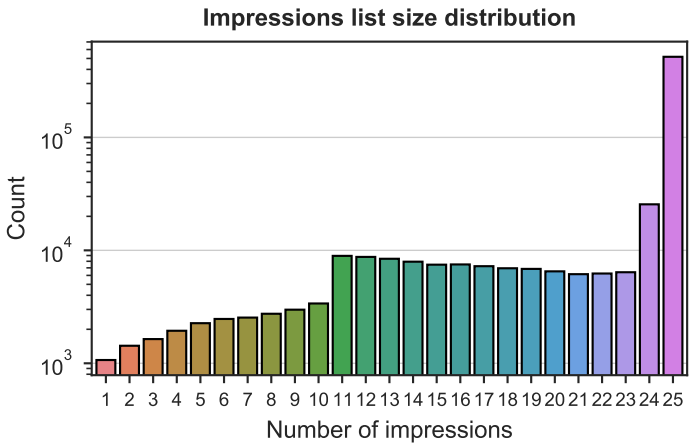


Figure 4.12: Log-scaled frequency distribution of impression list lengths.

As with demographic features in [7], the extremely location-dependent travel domain is likely to benefit from geographical attributes, including the `city` in which accommodation is searched for, and the website `platform` region accessed by the user, to provide priors, especially important in cold-start situations (i.e., clicks at initial session time steps without much added previous information) and exploratory-driven event sequences. Besides, location differentiation is always crucial in global

[4] When combined with the augmentation procedure, the imposed time step limit essentially creates a rolling time-window setup of maximum width 25 for each session.

[5] Function `vrange` provided by Gareth Rees at `https://codereview.stackexchange.com/a/84980`, last accessed on 2020-01-13.

platforms as different markets and cultures are bound to also have different preferences, needs and possibilities. An important aspect influenced by location is the pricing of available listings[6], as can be seen for the top ten most interacted cities of the dataset in Figure 4.13. The distribution simultaneously shows the slight but generally present user gravitation towards cheaper items.

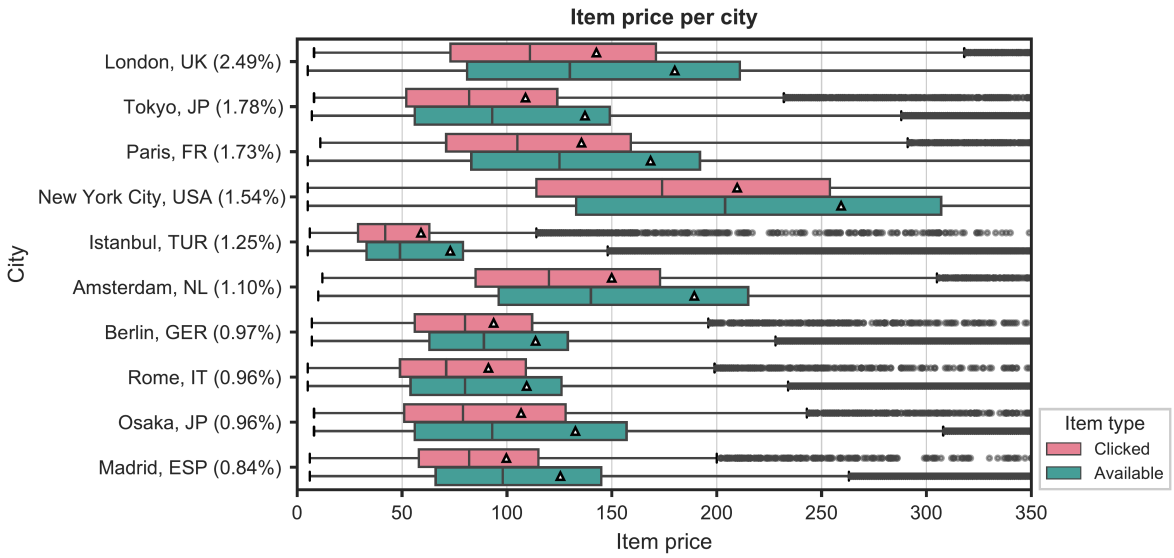


Figure 4.13: Comparison of clicked and available item prices for the ten most popular cities (by interaction percentage). The pricing differentiation between regions is clear. For example, hotels in New York City are, on average, two times more expensive than in Istanbul. The average delta between mean clicked and available prices for these top cities is 29.14.

Supplemental general features, like the type of `device` used by the user during the session can provide similar benefits. As mentioned in [34], mobile devices are likely to experience drastic position biases mostly due to smaller screen sizes. In this dataset, however, the difference is quite negligible, as can be seen in Figure 4.14. Tablet navigation seems to experience the opposite effect with a mean slightly shifted towards lower ranked item clicks than the remaining devices.

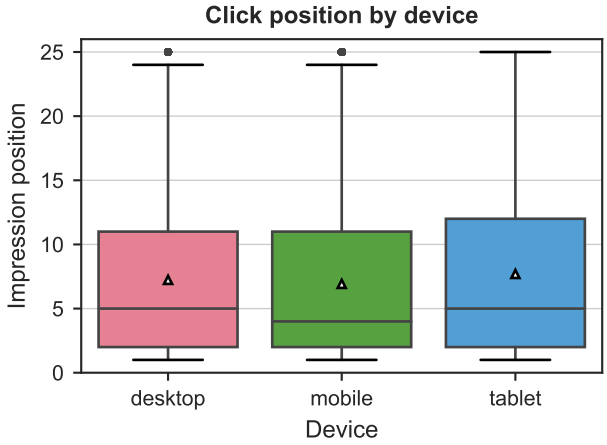


Figure 4.14: Position of the clicked item by type of device used during the session.

[6]No currency unit was provided.

Filter features `Filters` selected by users during the search process, directly related to the items' metadata attributes, are strong indicators of specific circumstantial interests.

The combined explicitly chosen filter selection and sort order change references demonstrate how portions of user searches were widely influenced by factors such as listing price (sort by price - the most chosen by far, best value), distance to a specific point of interest (sort by distance), rating (5 star, 4 star, sort by rating) and other characteristics (accommodation type, etc.), as can be observed in Table 4.1. As such, the set of active filters and sort interactions at each subsession's last time step were jointly added to the sample's session features.

Table 4.1: Selection distribution for the ten most chosen filters based on 654 984 total filter interaction events.

Filter/Sort	Selection frequency (%)
Sort by Price	25.01
Sort by Distance	8.18
Hotel	6.70
Best Value	5.48
5 Star	5.12
Resort	4.84
4 Star	4.76
Hostal (ES)	4.04
Motel	3.87
Sort by Rating	2.85

Impression features

Impressions are central to the problem as they define the interface of possible user interaction outcomes in relevant predictive events. Descriptive item features, popularity indicators and interaction aggregators for the variable length impression list presented to each user in a click event, that condition and provide the most differentiation from typical top-K recommendation, were ultimately found to be some of the most essential to the re-ranking task's performance. Their generation also proved to be one of the most laborious aspects of this work. The need to preserve the natural temporal dynamic of user sessions and prevent future data leakage, combined with the required tracking of each item's values in and across sessions without the ability to use algorithmic loops due to computational limitations required alternative vectorized implementation strategies.

The click events were first retrieved from the preprocessed dataset and the impression lists were expanded, such that the presented `impression item IDs` formed the row index of a new impression-based DataFrame with 15 263 546 entries.

Besides the `position` of each item (which might even be ambiguous given subsequent RNN processing, when bias impact is not analyzed), their `price`, which has already been discussed as being a key, interaction-driving characteristic, was also extracted from the logs. The click distribution per price rank of Figure 4.15 confirms the heightened price sensitivity of the domain noted by trivago in

[13] and also implied by the price-related top interacted filter in Table 4.1, displaying the expected user bias toward cheaper accommodation (as had been briefly suggested when discussing the clicked city prices of Figure 4.13).

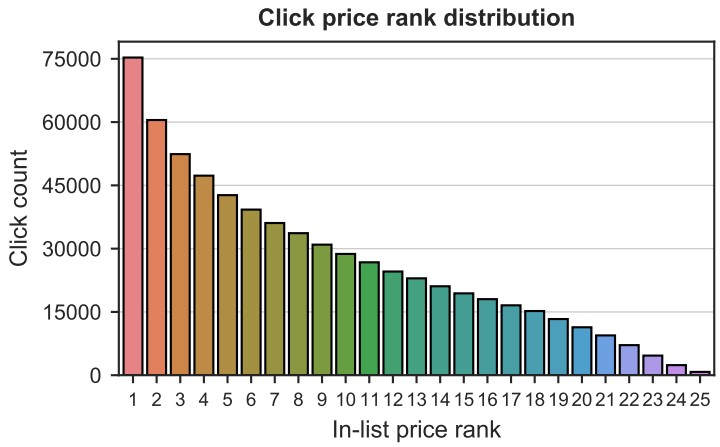


Figure 4.15: Clicked price rank distribution. Lower rank is equivalent to lower relative price and ranks are share between same-priced items.

The aforementioned item-specific popularity and previous interaction signals were captured in four additional features: `views`, representing the number of times each item had been in former impression lists (which can be regarded as previous item views, under the assumption that every item in a given list is viewed by the user), the number of past item `clicks` and `interactions` of other distinct types, and the aggregate past `dwell time` for each item (Equation 4.1e), as represented in Figure 4.1. It was assumed that an interaction with a specific item persisted until the start of another event with a different reference item. Each of these features was further subdivided into local (session-based) and global (inter-session) components. While the latter were mainly used to provide prior distributions, similarly to previous geographical and general session features, local variants were integrated to provide a more focused summarized insight into the user's session intent. Algorithm 2 represents the high-level pseudocode for the vectorized local and global `dwell time` generation, with a 317s total runtime.

For reference, a hybrid iterative and vectorized solution (with `Pandas`' `itertuples` and `groupby`) to generate the much more straightforward `clicks`, from the smaller sequential DataFrame (filtered for click events only, with 660 526 rows), would take approximately 11 days to complete (at around 1.43s per click event)[7].

[7] The fully vectorized solution generates `clicks` in 389s, around 2443 times faster than the hybrid approach.

Algorithm 2: High-level pseudocode for the vectorized dwell feature generation. The dwell value for a given action reference item is only calculated in the following time step (Figure 4.1), except if the next event is a click, which only provides initial context and not the completed action information (relevant timestamp). A column b from DataFrame A is denoted by `A:b`.

Input: `Sequential` (`Seq`) and `Impressions` (`Imp`) DataFrames, where each entry corresponds to an interaction event and a presented impression list item, respectively.

`Seq` : $\texttt{click\_bin}_{\text{aux}} \leftarrow$ binary column, 1 if `action` $=$ `clickout`, 0 otherwise
`Seq` : $\texttt{click\_subsession}_{\text{aux}} \leftarrow$ counter incremented with `click_bin` at each click event step, reset between sessions
`Seq` : $\texttt{dwell\_ref}_{\text{aux}} \leftarrow$ shifted action reference values to the following step (undefined for first session event)
`Seq` : $\texttt{dwell\_val}_{\text{aux}} \leftarrow$ calculated dwell for each `dwell_ref` at every step with Equation 4.1e

▷ *propagate values for interacted items over subsessions (global)/sessions (local)*

`Seq` : `seq_dwell` $\leftarrow$ cumulative sum over `dwell_val` for each `dwell_ref` (`local:` reset between sessions)

▷ *fix and propagate values for viewed items with previous interactions*

$\texttt{Last\_dwell}_{\text{aux}}$ and $\texttt{Last\_dwell\_click}_{\text{aux}} \leftarrow$ DataFrames with `seq_dwell` for the last interaction with each `dwell_ref` for every `subsession` and `click_subsession`, respectively (with `Imp`'s index)
delete last event from `Last_dwell_click` for every session (last session click values)
$\texttt{Dwell\_group}_{\text{aux}} \leftarrow$ left merge `Last_dwell` and `Last_dwell_click`
$\texttt{click\_dwell}_{\text{aux}} \leftarrow$ `Last_dwell:seq_dwell` where `action` $=$ `clickout`
replace `Last_dwell:seq_dwell` values with `Last_dwell_click:seq_dwell`, where `action` $=$ `clickout`
left merge `Imp` and `Dwell_group` DataFrames
fill `click_dwell` values forward in time for each impression list item (`local:` only inside each session)

▷ *account for consecutive click events*

delete filled `click_dwell` values in original non-filled locations (click values)
refill `click_dwell` forward in time
`dwell` $\leftarrow$ `Last_dwell_click:seq_dwell` where `action` $=$ `clickout`
fill `dwell` values forward in time for each impressions list item (`local:` only inside each session)
fill empty `dwell` values with 0
replace `dwell` filled values with `click_dwell` filled values where `click_dwell` $>$ `dwell`

Table 4.2: Frequency contigency table for clicked item events (A) and local interactions events prior to clicks (B).

		Interacted B	Not interacted $\bar{B}$
Clicked	A	303 404	357 122
Not Clicked	$\bar{A}$	839 106	13 763 914

The impact of local features on the recommendation task can be exemplified with the contingency Table 4.2, containing the frequencies of presented clicked items (A) and presented items with previous session interactions (B). The conditional probabilities $P(A|B)$ and $P(A|\bar{B})$, i.e., the probability of a sampled list item being clicked given that it had been previously interacted with in the session and the probability of an item being clicked without any previous local interaction, respectively, are

easily retrieved from the table using Equations A.1 and A.2:

$$P(A|B) = \frac{303\,404}{303\,404 + 839\,106} = 0.2656, \tag{4.2a}$$

$$P(A|\bar{B}) = \frac{357\,122}{357\,122 + 13\,763\,914} = 0.0253. \tag{4.2b}$$

These values show that a previous item interaction in the session substantially increases the click probability, by over an order of magnitude.

The combination of some of these previous features, such as `views` and `clicks`, can also be important to introduce churn in the recommendation, a powerful mechanism that helps balance the exploration/exploitation in engagement strategies [8], more relevant in user-based environments. In [7], for instance, non-interacted features previously presented to each user are demoted in the ranking (FW.6).

Although not available in RSC19, visual features decurrent from the page layout such as item thumbnails, are known to be crucial user interaction influencers [7, 60], can be easily integrated into the model's architecture and joint-trained with, for instance, an additional convolutional input (FW.8).

Metadata features The static item attributes provided by the metadata database were expanded into a DataFrame sharing the same item-based index structure of the remaining impression features. Although limited (no geographic pointers, for instance), the characterization provided by these features is important, especially when combined with active filters in a session, as not every item presented to the user actually satisfies the required filter conditions. Additionally, as can be observed in Table 4.3, some attributes also function as triggers for user interaction. The vast majority of clicked items are naturally characterized by standard amenities such as shower or WiFi availability, which are also present in the greatest accommodation share. When sorting by the ratio of clicked to availability percentages (to take less frequent amenities into account), however, it is noted that high-end attributes often lead to higher click probabilities.

Table 4.3: Metadata attributes sorted by presence in clicked items (left) and by clicked to general availability ratio, $r_{\text{meta}} = \frac{\text{Click. \%}}{\text{Avail. \%}}$, for a minimum availability of 1% (right).

Attribute	Clicked (%)	Avail. (%)	r_{meta}
Satisfactory Rating	77.66	57.52	1.35
Shower	69.19	46.04	1.50
Good Rating	68.60	51.98	1.32
WiFi (Public Areas)	67.37	43.09	1.56
Television	66.80	45.94	1.45
Car Park	66.65	52.62	1.27
WiFi (Rooms)	66.07	50.37	1.31
Hotel	60.18	40.91	1.47
Non-Smoking Rooms	57.38	37.32	1.54
Openable Windows	56.12	37.55	1.49

Attribute	Clicked (%)	Avail. (%)	r_{meta}
5 Star	3.29	1.15	2.86
Convention Hotel	7.99	3.05	2.62
Boutique Shopping	6.23	2.50	2.49
Romantic	20.76	8.38	2.48
From 4 Stars	19.81	8.13	2.44
Spa	11.56	4.77	2.43
Porter	14.43	5.95	2.43
Hammam	2.61	1.08	2.42
Body Treatments	5.95	2.49	2.39
Room Service (24/7)	9.73	4.09	2.38

4.2.3 Dataset partitioning setup

The impossibility of evaluating the performance of created models locally, due to the unavailability of ground truth data for omitted click references already mentioned in Section 1.2, resulted in the need to replicate the offline validation setup of a well-documented challenge solution to obtain comparable results.[8]

The data was split according to the process applied by the 7th overall placed team *Mustelideos* [113], which also framed the problem as multiclass classification with every click considered as a training sample, and reproduced a setting similar to that found in the original challenge's test dataset, omitting the labels for each user's last largest session clicks in the final two training days (05 and 06 November). For hyperparameter optimization purposes, a subset of the resulting training data was split into smaller validation and test sets, corresponding to the last largest session clicks in the third and fourth days (03 and 04 November), respectively.

The resulting sample distribution can be visualized in Figure 4.16, with further insights for each of the sets provided in Table 4.4. Although examples from the sixth day are used to train the model, which then evaluates fifth day samples, the sessions were assumed to be independent and no future data from the same users in the test set was considered during training.

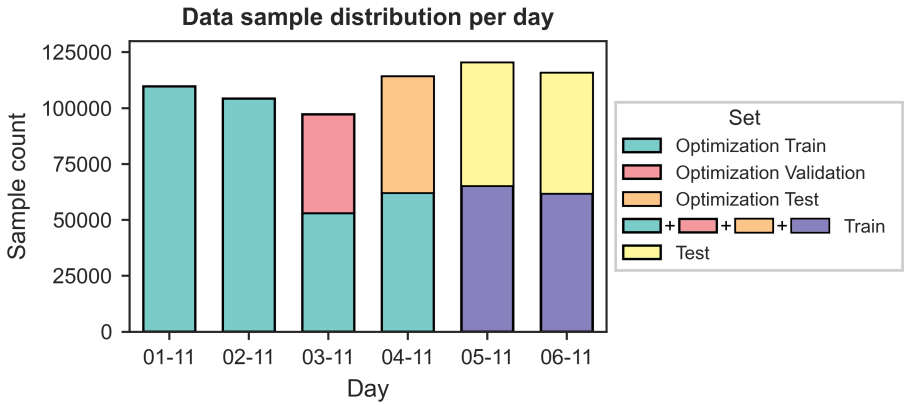


Figure 4.16: Dataset sample distribution per day for the training, validation and test sets.

Table 4.4: Training, validation and test set insights. The number of unique sessions is equal to the sample number in validation and test sets.

	Optimization sets			Full sets	
Data insight	Train	Validation	Test	Train	Test
Sample (subsession) count	328 618	44 206	51 875	551 667	108 859
Unique sessions	156 591	44 206	51 875	267 878	108 859
Avg. sequence length	8.44	7.27	7.38	8.35	7.35
Avg. impression list length	23.15	22.72	22.86	23.15	22.87

[8] None of the other challenge entries' feature or model architecture generation processes were referenced throughout the development of the model, before the final evaluation stage.

4.2.4 Feature representation

The processed dataset $\bar{\boldsymbol{X}}$, comprised of the enumerated continuous numerical and discrete categorical features, had to be transformed in order to create suitable numeric input representations for the deep learning model, $\boldsymbol{X} = \{\boldsymbol{X}_{\text{seq}}, \boldsymbol{X}_{\text{ses}}, \boldsymbol{X}_{\text{imp}}\}$, where $\boldsymbol{X}_{\text{filt}} \subset \boldsymbol{X}_{\text{ses}}$ and $\boldsymbol{X}_{\text{meta}} \subset \boldsymbol{X}_{\text{imp}}$.

Embedding nominal categorical features

One of the simplest categorical transformation methods available consists in one-hot encoding the categories such that each one is represented in a different dimension by a unit vector in $\mathbb{R}^{d_{\text{voc}}}$, where d_{voc} is the vocabulary size (number of categories), and each pair of categories is at Euclidean distance $\sqrt{2}$ from each other [38].[9] Nonetheless, this sparse representation quickly becomes inefficient when dealing with larger vocabulary sizes and does not capture any similarity between categories [75]. For these reasons, one-hot encoding was only used for the `device` and `platform` features.

Alternatively, the categories can be mapped into dense continuous vector embeddings in a lower dimensional $\mathbb{R}^{d_{\text{emb}}}$ space $(d_{\text{emb}} < d_{\text{voc}})$, capable of encoding inter-category relationships[10], such as those obtained for users and items in MF. These embeddings can be either learned jointly with the remaining model parameters from the data during training, or initialized and pre-trained separately, usually with methods based on Word2Vec [114] (contextual information) or GloVe [115] (co-occurrence counts), initially developed for NLP applications. In the recommendation domain, specialized skip-gram with negative-sampling (SGNS) [66] models such as Item2Vec [116] and Meta-Prod2Vec [117] might be worth exploring to reduce the impact of input embedding learnable parameter dominance in neural networks, consequence of the commonly large item vocabularies (FW.9).

An approach similar to that of [7] was taken, with features belonging to the same vocabulary/ID space sharing the same embedding layers but being separately input to the network, such that specialized representations are joint-learned per feature by deeper layers, with added efficiency and generalization benefits. In this case, four ID spaces were considered: the item ID space, shared by `reference item ID` and `impression item ID`, the attribute ID space, shared by `filters` and `item attributes`, and the remaining individual action ID and city ID spaces, for `action ID` and `city` respectively.

Table 4.5: Vocabulary sizes (unique IDs) for the considered embedding ID spaces.

	Item ID	Attribute ID	Action ID	City ID
Vocab. size	713 602	168	10	20 268

[9] For example, the second category of a categorical feature with five-dimensional vocabulary may be represented by the vector with a 1 at index 2, $[\,0\;1\;0\;0\;0\,]$.

[10] Early word embedding applications noted that the learned representation space grouped similar words closer together (with similar vectors), additionally disclosing other hidden syntactic and semantic correlations [66].

These vocabularies, whose sizes are displayed in Table 4.5, were created after a full data pass[11] and consisted of lookup tables mapping the original IDs into sequential integers, which are then subsequently mapped to the corresponding vector embeddings by `TensorFlow`'s embedding layer implementation as part of the model. Out-of-vocabulary (OOV) values, such as non-accommodation reference IDs, were mapped to the 1 integer token.

Although popular in the literature [7, 62] and suitable for serving conditions (in case popularity bias or limited coverage are not of concern), retaining only an arbitrary top percentage of most popular items by number of log interactions to reduce embedding parameters was not implemented due to the resulting favorable prediction bias.

Normalizing continuous and ordinal features

Despite their theoretical robustness to uninformative and correlated features [77], with the capacity to automatically learn what details can be discarded when given enough training examples [38, 111], neural networks can still be sensitive to input distributions and scaling. Data normalization is a full pass processing technique that scales input features such that their magnitude ranges become similar[12] [75], known to promote SGD efficiency [86], and even being found critical for convergence in [7].

The Min-max normalization method is typically used to linearly rescale each variable to the [0,1] range. However, unbounded counter or time-based features are especially prone to legitimate extreme observations, sometimes highly influential in the distributions, which might contain valuable information that would be wrongly discarded if artificial value limits were to be imposed [118] (FW.10). Therefore, Quant, the non-linear method applied in [7, 41], consisting of a uniform distribution mapping from an estimate of the feature's cumulative distribution function, was implemented instead, with `scikit-learn 0.23`'s `QuantileTransformer` [119], yielding better results (Section 5.2.2).[13] Statistics were computed on the training set only, as to prevent information leakage from the test data.

Padding and masking sequences

Since the neural network implementation requires fixed-length inputs[14], sequences were post-padded with zeros, in the case of embedding inputs, and negative ones, for the remaining features where zeros were meaningful values in the data. Padded time steps were then masked so that they were not considered by the network without it having to learn its irrelevance.

In the previous Section 4.2.2, it was mentioned that interaction sequences (in $\boldsymbol{X}_{\text{seq}}$) were truncated at 25 time steps. This value also corresponds to the maximum impression length available (in

[11] It was assumed that the ID spaces were immutable in the timespan considered.

[12] Preferably with averages close to zero.

[13] Increased performance was noted even when restricting min-max usage to ordinal features, as to prevent linear correlation distortion, see M15 in Section 5.2.2.

[14] Mostly for efficiency, as `TensorFlow` requires predefined shapes for some operations. To make use of the faster GPU RNN variants (cuDNN-based), such as `cuDNNGRU`, the inputs must be strictly post-padded and recurrent dropout is not available. In `https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU`, last accessed on 2020-03-24.

$\boldsymbol{X}_{\text{imp}}$), and as embeddings are shared between `items IDs` in these two feature blocks, additional unnecessary padding is avoided.

Because every individual metadata `item attribute` and `filter` was mapped to its own embedding vector, the attributes characterizing a single item in the impression lists and the active filters in a single session corresponded to variable length embedding sequences. To concatenate them with the remaining impression and session features, these embedding sequences were averaged, as done in [7], creating non-sequential unified sample representations. To accomplish this, each sequence was first padded with zeros and masked to obtain fixed sequences of length 112 (the maximum number of simultaneous item attributes and active filters). Metadata sequences, for instance, were then input to a `TimeDistributed` Average Pooling layer (to average each of the available impression items' attributes) and then passed over by a `Lambda` layer to re-mask any `NaN` (Not a Number) values resulting from averages over zero-only sequences.

4.3 Model architecture

The deep learning model's structural core was designed following a multitask transfer learning architecture [38], with specialized modules inspired by the literature, processing the different feature inputs separately (with parameter sharing limited to the embeddings), whose output representations are then combined and shared by deeper layers. The final version of the model was obtained from a general possible hyperparameter space via the optimization process detailed below.

4.3.1 Hyperparameter optimization

The optimization process was designed to support a conceptualized highly conditional hyperparameter space centered on the available inputs, defining a wide variety of possible model complexity. This space, fully described in Table 4.6, is subdivided into three main specialized branches tasked with processing the impression ($\boldsymbol{X}_{\text{imp}}$), interaction sequence ($\boldsymbol{X}_{\text{seq}}$) and session ($\boldsymbol{X}_{\text{ses}}$) inputs, whose output representations are then combined, converging into an MLP (Section 3.1) connected to the final dimensionality-correcting softmax layer that produces the classifier's output (Section 3.3.1).

The sequential processing structures, for the impression and interaction sequence input branches, were influenced by [48, 50, 58, 61, 62], consisting of an RNN block followed by an optional dense (fully connected feedforward) layer. Four different GRU (Section 3.2) types were made available, corresponding to simple single layer, stacked (S-GRU, two-layers with equal number of units), bidirectional (Bi-GRU) and stacked bidirectional (Bi-S-GRU) versions.

The interaction sequence's RNN block was adapted to include an optional self-attention (Section 3.4) layer, implementing the four mechanisms presented in Table 3.1. Additive, Dot and Scaled Dot attentions were modified from [120], while the remaining Hierarchical attention was modified from [121], to properly support padded time step masking. Additionally, as in [122], Additive, Dot and Scaled Dot attention matrices were averaged over the time axis to produce single attention vectors (and consequent context vectors).

Table 4.6: Conditional hyperparameter space that controls the model's architecture and respective value ranges (adapted to suit memory and processing limitations, based on initial test runs and typical literature values). Indented children hyperparameters require active parents. Inputs are marked with (I), embedding dimensions with (E), and gated parameters that can become False (increase negative selection probability in mostly continuous variables and enable the optimizer to more easily discard full irrelevant blocks), with (B). Impression and interaction sequence features were divided into codes (item ID sequences) and attributes (remaining features). With this configuration, the impression list item embedding sequence is the only guaranteed input. Active O1/M1 hyperparameters are boldfaced.

	Hyperparameters	Codes	Range
Impressions	**Impression attributes (I)**	**H1**	{T, F}
	Item embedding (E)	**H2**	{2,...,50}
	Impression RNN type	**H3**	{GRU, S-GRU, Bi-GRU, Bi-S-GRU}
	Impression RNN units	**H4**	{10,...,250}
	Impression RNN dropout (B)	H5	[0,...,0.7]
	Impression Dense units (B)	H6	{10,...,250}
	Impression Dense dropout (B)	H7	[0,...,0.7]
	Metadata (I)	**H8**	{T, F}
	Metadata embedding (E)	**H9**	{2,...,30}
Interaction sequences	**Sequential codes (I)**	**H10**	{T, F}
	Sequential attributes (I)	**H11**	{T, F}
	Action embedding (E)	**H12**	{1,...,15}
	Sequential RNN type	**H13**	{GRU, S-GRU, Bi-GRU, Bi-S-GRU}
	Sequential RNN units	**H14**	{10,...,250}
	Sequential RNN dropout (B)	H15	[0,...,0.7]
	Sequential Self-attention type (B)	**H16**	{None, Add., Dot, Hierarch., Scaled}
	Attention dimension	**H17**	{32,...,320}
	Sequential Dense units (B)	H18	{10,...,250}
	Sequential Dense dropout (B)	H19	[0,...,0.7]
Sessions	**Session features (I)**	**H20**	{T, F}
	Filters (I)	**H21**	{T, F}
	City embedding (E)	**H22**	{2,...,30}
	Session Dense units	**H23**	{10,...,250}
	Session Dense dropout (B)	**H24**	[0,...,0.7]
Joint MLP	**Out 3 Dense units (B)**	**H25**	{25;125,...,500}
	Out 3 Dense dropout (B)	**H26**	[0,...,0.7]
	Out 2 Dense units (B)	H27	{150,...,1000}
	Out 2 Dense dropout (B)	H28	[0,...,0.7]
	Out 1 Dense units (B)	H29	{2x Out 2 Dense units}
	Out 1 Dense dropout (B)	H30	[0,...,0.7]
Params	**Dense activation functions**	**H31**	{ReLU, PReLU, LeakyReLU}
	Learning rate	**H32**	$[10^{-4},...,10^{-1}]$

The remaining session feature input is processed using a single dense layer. The joint MLP block, tasked with processing the concatenated embedded representations output by the three main branches, was based on [7]'s design and consists of three possible dense layers with decreasing number of available units.

Dropout (Section 3.3.5) was made available after every layer in the network. Two additional ReLU variants, PReLU and LeakyReLU represented in Figure A.1c, were also made available as activation functions for the dense layers.

Optimization setup The hyperparameter space was optimized using Bayesian Optimization with Gaussian Process Upper Confidence Bound (BO GP-UCB, Section 3.3.4), using a modified `Keras-Tuner 1.0`'s `BayesianOptimizationOracle` [123] implementation and `TensorBoard`'s `HParams` dashboard, to maximize the MRR objective function.

Default tuner σ^2_{noise} and β values of 0.0001 and 2.6 were used (Equations 3.27 and 3.33). *Matérn* kernel functions were used for the GP, with ν set to 5/2 (Equation 3.31), and inactive hyperparameters in optimization runs were reset to default minimum values, as in [90]. `Scikit-learn`'s `GaussianProcessRegressor` is used in the tuner's back end to optimize the kernel's length scale such that it maximizes the log marginal likelihood, with the L-BFGS-B algorithm.[15]

To cover the most amount of space in the limited time frame, taking the expensive sample evaluation into account, the batch size was not tuned but instead fixed at 512 (Section 3.3.3) and early stopping patience (number of allowed epochs with increasing validation loss versus the minimum obtained in the run, Section 3.3.5) was set to 1. The Adam optimizer (Section 3.3.3) was used, with a tuned initial learning rate given by H32 (remaining hyperparameters such as moment estimate decay rates were left with default values).

The final process, which consisted of 1530 total runs and whose results are described in next chapter's Section 5.1, was divided into three modes:

1. Random: The bayesian optimization process can be initialized with an arbitrary number of data points. This implementation generated 180 random samples as initial training data for the tuner, corresponding to four times the dimensionality of the hyperparameter space (increased from the default three due to high conditionality).

2. Bayesian: The tuner iteratively applied BO GP-UCB (Algorithm 1) for 1200 runs, from the random foundation.

3. Top: Due to the non-deterministic nature of neural networks, the top scoring hyperparameter set was not selected directly from the Bayesian section. Instead, the top 20 architectures were picked for five additional runs with different parameter initializations. The selection was then refined, with the top 10 being run an additional five times, for a more comprehensive performance overview.

[15] If this algorithm does not converge within 20 restarts by default, the tuner draws a random sample from the hyperparameter space. The UCB acquisition function was similarly optimized with L-BFGS-B, with a higher restart number of 50.

4.3.2 Final model

The architecture with the best final average performance on the optimization test set (Section 5.1), O1/M1 from Table C.1, is represented in Figure 4.17's diagram.

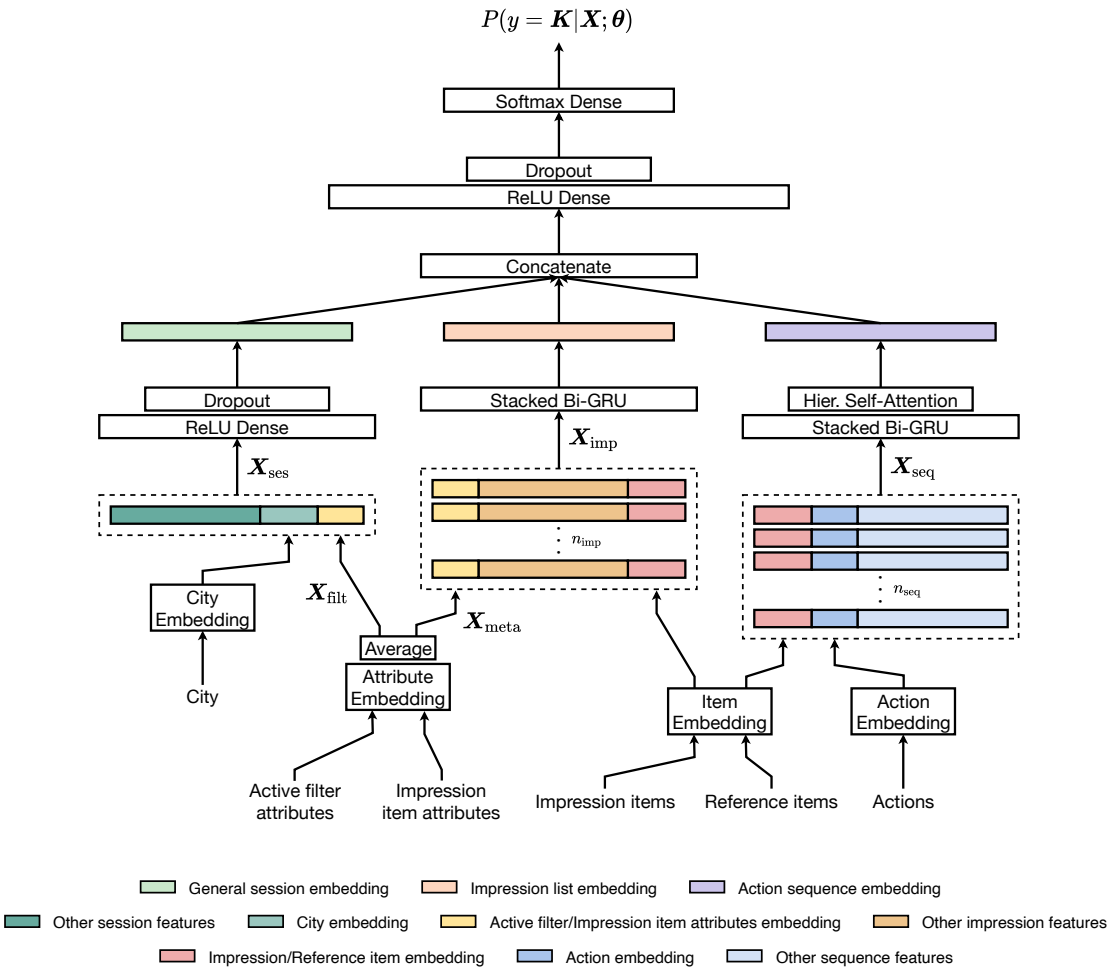


Figure 4.17: Final model diagram, corresponding to the O1/M1 architecture. Embedding, layer and other component dimensions not to scale.

With every input active, the model assumes a parallel three-branched structure. Stacked Bi-GRU modules process the sequential inputs, with the interaction sequences being further processed with a Hierarchical self-attention mechanism: the impression sequences get embedded into vectors in $\mathbb{R}^{2h_{\text{imp}}}$, the concatenated bidirectional final hidden states from the second GRU layer given by Equations 3.5 (a-d), where h_{imp} is the number of impression GRU hidden units (H4); the interaction sequences get similarly transformed but all the hidden states from every time step are passed in a $\mathbb{R}^{n_{\text{seq}} \times 2h_{\text{seq}}}$ matrix, which is then embedded into $\mathbb{R}^{2h_{\text{seq}}}$ by the self-attention with Equations 3.34, 3.35 and 3.39, where h_{seq} is the number of interaction sequences GRU hidden units (H14). H17 defines $\boldsymbol{u}_a$ and $\boldsymbol{W}_a$'s dimensionality in Equation 3.39.

The session features input is processed with a ReLU dense layer with dropout. The transformed feature representations are concatenated and processed by a ReLU dense layer with dropout and then input to the final 25 unit softmax dense layer to produce the click probabilities for the impression list items.

Table 4.7: Model O1/M1's learnable parameter distribution.

Layer name	Parameter count	Distribution	
Item Embedding	1 427 206	28.29%	34.74%
Attribute Embedding	507	0.01%	
Action Embedding	66	0.001%	
City Embedding	324 304	6.43%	
Seq. Bi-GRU 1	330 924	6.56%	24.44%
Seq. Bi-GRU 2	858 048	17.01%	
Hier. Self-Attention	43 800	0.87%	
Imp. Bi-GRU 1	402 000	7.97%	30.33%
Imp. Bi-GRU 2	1 128 000	22.36%	
Session Dense	4 828	0.10%	0.10%
Out Dense	512 525	10.16%	10.40%
Softmax	11 900	0.24%	
Total	5 044 108	100.00%	

The model contains 5 044 108 total learnable parameters, distributed as represented in Table 4.7. The largest portion of these (slightly more than a third) is allocated to the embeddings, as expected mostly due to the items' large vocabulary, reflecting what had already been noted in [7].

Chapter 5

Results and Discussion

The following sections are dedicated to the presentation and discussion of the results obtained throughout the recommender's development, addressing the proposed objectives and respective research questions from Section 1.3: Both of RQ.3's points are tackled in the optimization Section 5.1, where both the general and hyperparameter-specific performance distributions display the bayesian process' impact (RQ.3.1), and where the random data point initialization is compared to a fully bayesian one (RQ.3.2). Section 5.2 comprises the final model results (RQ.1). The attention mechanism's behavior is first analyzed in Section 5.2.1 (RQ.2.1), while its contribution to the model's performance (RQ.2.2) is explored in the ablation Section 5.2.2, along with that of the remaining model components (RQ.1.1). Finally, Sections 5.2.3 and 5.2.4 concentrate on RQ.1.2, comparing the model to simpler baselines and to the challenge's leaderboard submissions.

Note The arithmetic mean is used as the default averaging method when not stated otherwise.

5.1 Optimization results

Following the process setup described in the previous Section 4.3.1, the MRR results obtained in the optimization runs for the three different modes (Random, Bayesian, Top) are plotted in Figure 5.1, and an additional mode overview is presented in Table 5.1. The average time per run was of 431 seconds, with CPU-based[1] runs taking approximately 16 times longer than those completed on GPUs.

Table 5.1: Hyperparameter optimization result overview by mode.

Optimization mode	Count	Mean MRR (SD)	Max. MRR	Max. step
Random	180	0.47025 (0.07286)	0.61508	112
Bayesian	1200	0.62517 (0.01810)	0.63884	1373
Top	150	0.63641 (0.00186)	0.64091	1507

[1] CPU runs were required for some hyperparameter interval limit values, such as 50 dimensional item embeddings, due to memory leakage issues with some of the randomly allocated GPUs.

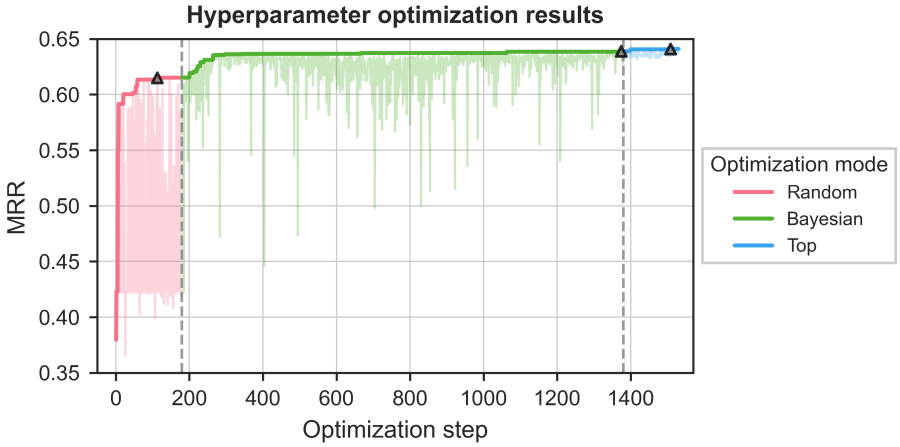


Figure 5.1: Hyperparameter optimization MRR results and maximum performance evolution per run. Triangles mark the best runs in each optimization mode.

While the Random mode got within 4.03% of the maximum value obtained, the MRR in its last 120 runs only increased by 0.29%. The Bayesian mode promoted a contrasting rapid performance increase of 3.29% in the following 85 runs, getting within 0.87% of the best result. Although the increase was much less significant during the mode's remaining runs, the fact that its best value was obtained close to the end signaled a possibility for further improvement, if time limits were not of concern. As additional advantages, this second mode combines significantly increased average performance values with an approximately four times smaller standard deviation, reflected in Figure 5.1's much lower MRR spread, where the observable lower peaks mostly correspond to the kernel optimization convergence failure runs mentioned in the setup, which draw random hyperparameter sets. The optimizer's exploratory ability is also maintained, as seen in the following Section 5.1.1.[2]

To more directly assess the Random mode's impact, the first 180 runs were repeated using BO GP-UCB without space initialization. Each run's results and both optimization modes' cumulative maximum values are plotted in Figure 5.2.

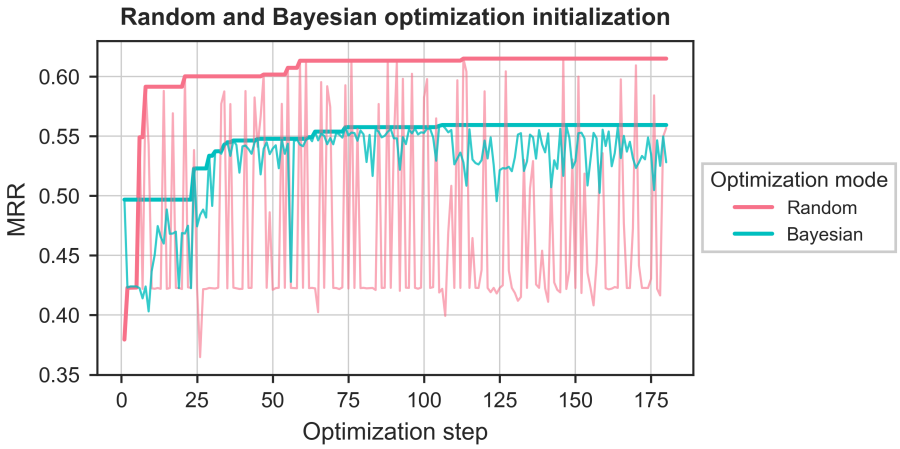


Figure 5.2: Random and Bayesian optimization initialization comparison.

[2]The following section's distributions also show that limiting a few impactful hyperparameters' values could significantly improve the Random mode's performance, a property that would require manual effort but that is exploited automatically in the Bayesian mode.

The Random mode's higher exploratory potential is favorable, providing a good foundation for future runs. With a constant UCB β of 2.6, the more exploitative bayesian optimizer's performance is intimately tied to its initial most randomized runs, which, in this case, resulted in a 9% lower best MRR at the end of the 180 iterations. A possibly better performing fully bayesian alternative could make use of an adaptive β_n model, replacing the fully randomized section [92, 94] (FW.4).

5.1.1 Hyperparameter-specific results

Hyperparameter-specific ranking performance distributions of the optimization process, colored by mode, are presented in Appendix C.2. The top architectures are detailed more extensively in the following section.

The input's influence over the ranking is displayed in Figure C.1, which shows the result plots obtained for each boolean condition of the impression feature (H1), metadata feature (H8), interaction sequence (H10), sequential feature (H11), session feature (H20), and filter feature (H21) hyperparameters. Apart from the impression item-describing features, which yield some of the worst overall results when disabled, the comparatively limited contribution of the remaining feature sets is noted, but it is nevertheless clear that proper usage of the full input space is beneficial, as supported by the top performing configurations. The sequential block's inclusion follows in terms of significance, with its feature, and remaining metadata, session and filter inputs presenting very similar impact. This can also be observed in the averaged results of Figure 5.3 (generally and per mode), inspired by [61].

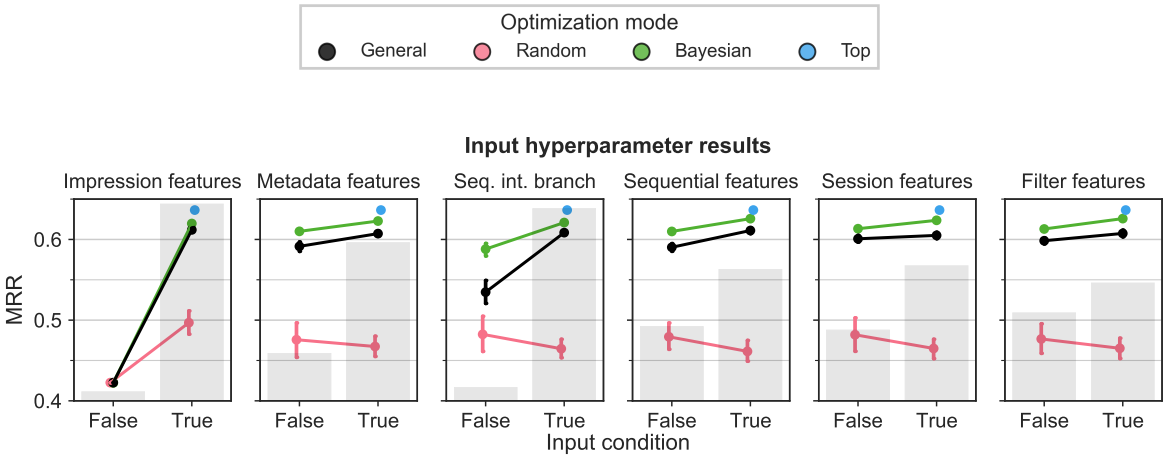


Figure 5.3: Averaged input optimization ranking performance distribution with 95% confidence intervals obtained over 1000 bootstraps. Relative frequency distribution axes are omitted. General averages disregard the optimization mode.

The overlaid relative frequency bars for each input condition mostly reflect the bayesian optimizer's behavior on the non-conditional hyperparameters (H1, H10, and H20). Since metadata, sequential and filter features are dependent on their respective parent hyperparameter's availability, their inactive frequency is naturally higher.[3]

[3]It should be noted that the mean values presented are only able to provide a rough general comparative baseline as these are heavily influenced by condition frequency, i.e., a very low amount of inactive examples might translate into lower average condition values due to a greater impact of the generally worse random runs.

Performance correlations are more challenging to detect in the almost uniform embedding dimensionality distributions, presented in Figure C.2, with peaks often corresponding to more focused bayesian runs, as clearly evidenced by the higher value frequencies shown in the average plots of Figure 5.4. The lack of relative support for the remaining values results in the larger confidence intervals. With best value points scattered throughout the entire interval, the action embedding dimensionality demonstrates a considerable degree of independence from the MRR. The remaining plots show similarities in that the optimizer was able to obtain good results in the more explored range extremes of every space, although it can be observed that the best performing data points are found mainly at lower interval values, resulting in median dimensions of 3, 3, and 10 for the Top mode's item, attribute, and city embeddings, respectively.

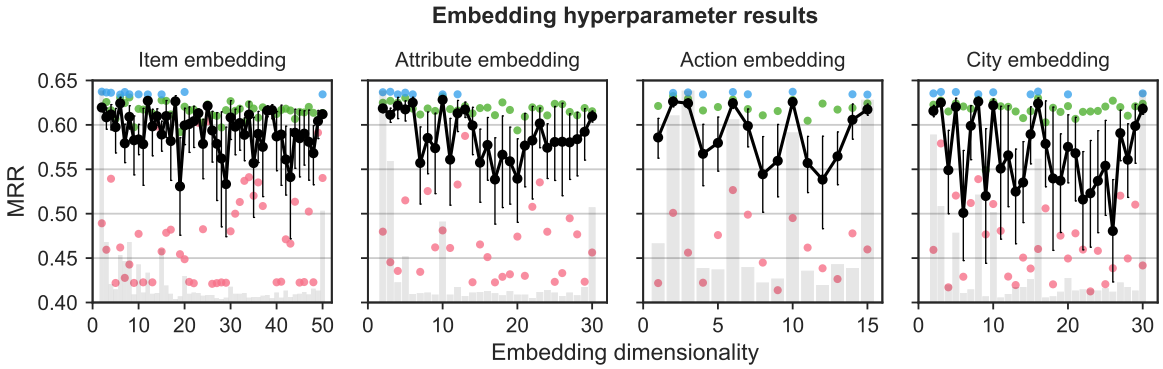


Figure 5.4: Effect of embedding latent dimensionality on ranking performance. 95% confidence intervals obtained over 1000 bootstraps. Relative frequency distribution axes are omitted.

With respect to the impression branch results of Figure C.3, it is clear that more complex recurrent architectures with more units lead to better performance. The first dropout layer was found to have a negligible effect and was skipped by the best model, with the remaining in the top ten only using up to 0.15, obtaining similar results. The dense block's inclusion only resulted in MRR decrease and was therefore not included in any of the Top mode's models.

The interaction sequence branch distributions are presented in Figure C.4. While better values are again tied to increased recurrent architecture complexity, the difference between GRU types is much less substantial than in the previous impression branch case. Comparatively, the unit count's impact is deeply reduced, with the top ten architectures not taking advantage of the full possible amount and instead focusing on the 124 to 231 range, for an overall median value of 201 recurrent units considering the best twenty models. Dropout was also found to have limited influence, having been skipped by every top ten configuration. Regarding the self-attention mechanisms, all the tested variants display a very similar but effective improvement over the base sequential block. In terms of attention dimensionality, the top bayesian results are evenly distributed across the range. Isolating the Top mode's results by mechanism type, it can be observed that the best values are returned for lower dimensions in the Hierarchical and Scaled versions, with medians of 100 and 77, when compared to the Additive and Dot's 279.5 and 248 median dimensions. Used by two of the Top

mode's models, with one employing the maximum number of units available, the following dense block proved not to be useful and was skipped by the top ten. Its dropout layer was found to only negatively impact the predictions.

The performance data points returned from the session branch (Figure C.5) show a more particular concentration of best results in the mid-value dense unit range, between 100 and 200. A similar behavior is verified in this block's dropout distribution, with the top ten using from 0.19 to 0.34, following a bayesian focus on the 0.3 value.

Regarding the joint MLP block's results from Figure C.6, it can be seen that the first possible (and widest, Out 1) layer's use has strictly negative consequences. With a mostly uniform distribution upwards of approximately 300 units, the best results obtained with the active second conditional dense layer (Out 2) were close but still inferior to those displayed without it, by the top ten models. Its specific dropout was not found to be helpful. By contrast, the third possible layer (Out 3) was used by every Top mode configuration, with unit values scattered throughout the range between 160 and 475 (used by the best model), with a median of 294 units. Its dropout layer was the only other consistently used by the best models, besides the session branch's, and the only used to produce every top twenty result, with values between 0.1 and 0.3.

Finally, Figure C.7 shows the distributions obtained for different Dense ReLU activation variants and initial learning rates. While PReLU demonstrated a slightly lower value spread, all the types obtained very similar results, with the best model using the base ReLU version. Adam's initial learning rate proved to be one of the overall most influential hyperparameters, with a best value region concentrated in the 0.001 to 0.003 range and a sharp performance decline for values over 0.01.

Visible across most plots is an unusual horizontal concentration of results around the 0.42 MRR region, corresponding to those of models that emphasize top position information (Section 5.2.3). Most often, these data points result from either a lack of impression features input, a large impressions block dropout value, a large initial learning rate or a combination of these. As alluded to in the previous section, preventing these hyperparameter choices would have a major positive impact in the process, at least substantially improving the Random results.

It is clear that such an intricate space would benefit from more exploratory coverage than that provided by the 1380 runs, to help draw more conclusive performance relations from some of the hyperparameters. Additionally, the large number of good results obtained by the optimizer near or at the ranges' limits point towards better possible results with less constrained value intervals on the same underlying architecture, a hypothesis which could be tested in a less computationally limited setup.

5.1.2 Best performing architectures

The top ten model architectures derived from the Top mode, O1 to O10, are detailed in Appendix C's Table C.1, and their corresponding optimization performance results are available in the Table 5.2 below.

Table 5.2: Top ten model metric values averaged over ten runs with SD values in parenthesis. Best and second-best results for each metric are boldfaced and underlined, respectively.

Model	MRR	Macro F1	Weighted F1	Accuracy	NLL
O1	**0.63829 (0.00133)**	0.45773 (0.00237)	**0.50563 (0.00155)**	**0.52141 (0.00169)**	**1.79424 (0.00990)**
O2	0.63792 (0.00122)	0.45640 (0.00214)	0.45640 (0.00214)	0.52039 (0.00220)	1.79556 (0.00934)
O3	0.63745 (0.00083)	0.45559 (0.00221)	0.45559 (0.00221)	0.52032 (0.00142)	1.79976 (0.00790)
O4	0.63722 (0.00118)	**0.45782 (0.00364)**	0.45782 (0.00364)	0.52006 (0.00154)	1.80059 (0.00612)
O5	0.63719 (0.00137)	0.45482 (0.00357)	0.45482 (0.00357)	0.51956 (0.00231)	1.79549 (0.00652)
O6	0.63705 (0.00104)	0.45588 (0.00228)	0.45588 (0.00228)	0.51939 (0.00198)	1.79794 (0.00598)
O7	0.63704 (0.00147)	0.45668 (0.00293)	0.45668 (0.00293)	0.52017 (0.00246)	1.79903 (0.00957)
O8	0.63690 (0.00111)	0.45544 (0.00217)	0.45544 (0.00217)	0.51923 (0.00183)	1.79841 (0.01036)
O9	0.63682 (0.00153)	0.45561 (0.00228)	0.45561 (0.00228)	0.51979 (0.00264)	1.79952 (0.01275)
O10	0.63680 (0.00128)	0.45600 (0.00186)	0.45600 (0.00186)	0.51902 (0.00229)	1.79877 (0.00483)

O1, previously described in Section 4.3.2, obtained the best average MRR performance over the ten final runs, with a 0.06% increase over O2's, and was therefore chosen for the final test set evaluation, explored in the following section. Classification-wise, however, the model was not the best performing on the most infrequent classes but a close second instead, after O4, as indicated by the macro F1. Its biggest advantage is associated to the highest list position class results, with the weighted F1 value demonstrating a considerable ten percentage point increase over O4's.

5.2 Final model results

The O1 architecture is denoted by M1 for all the final test set results. As was done at the end of the optimization process, the M1 model was evaluated over ten runs with different initialization parameters but now taking advantage of the full RSC19 dataset (refer to Figure 4.16). Lowering the batch size down to 128 was found to slightly improve the overall performance, with any further reduction only negatively impacting the training time. Early stopping patience was increased to five epochs in discarded initial runs monitoring the validation error on the previous section's optimization test set, one of which has its validation and accuracy curves represented in the Appendix Figure D.1. It was noted that the model would consistently obtain the lowest error value around the second epoch, point at which the subsequent runs were stopped, producing the results presented in Table 5.3. The average performance values show increases of two (MRR) to three and a half percent (NLL cost) when compared to the optimization results. Model training took an average of 618 seconds (309 seconds per epoch), while generating and evaluating the test set predictions, with a default batch size of 32, took an average of 58 seconds (0.53 milliseconds per sample).

An arithmetic MRR class average value of 3.81 can be calculated for M1*, the best ranking weight combination, from the individual predicted ranks, whose distribution is shown in Figure 5.5. This plot simultaneously represents the predictions' position deviation[4] from the desired target of 1, showing a satisfactory concentration of 75% of the clicked items within the first four impression positions.

[4]Subtracting one from the horizontal axis' values. A perfect predictive model would assign a rank of 1 to every label (0 rank deviation).

Table 5.3: Final M1 model architecture test results. Average and standard deviation values for 10 runs with different initialization parameters. M1* corresponds to the best ranking model from the run set. Specific recall and precision values were not recorded during optimization and are therefore not included in the delta comparison values.

M1 Model	MRR	M. F1	Wt. F1	M. Rec.	M. Prec.	Wt. Prec.	Acc.	NLL
M1*	**0.65148**	0.47277	**0.52095**	**0.42011**	0.55582	0.53698	**0.53505**	**1.72953**
Average	0.65113	**0.47346**	0.51923	0.41682	**0.56471**	**0.53941**	0.53465	1.73079
O1 Delta	+2.01%	+3.44%	+2.69%	-	-	-	+2.54%	-3.54%
SD	0.00052	0.00116	0.00009	0.00413	0.01448	0.00595	0.00109	0.00118

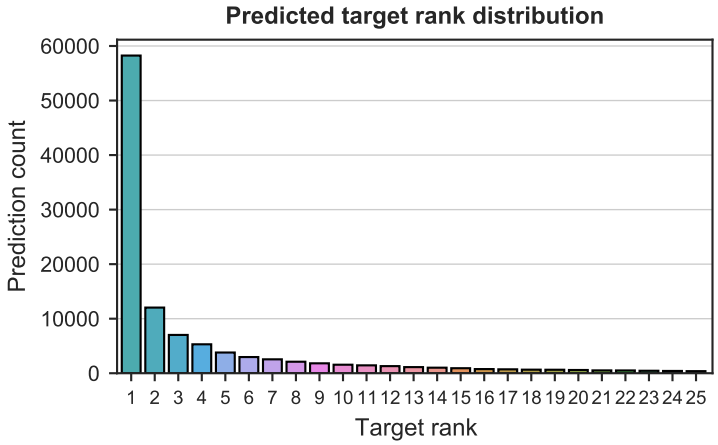


Figure 5.5: M1*'s predicted target rank distribution.

As had been previously noted, greater ranking performance does not directly translate to strictly better classification results. M1* demonstrates an increased correct classification count but displays precision values below M1's averaged runs. The individual class performance results obtained with M1* can be observed in Table 5.4. While the MRR values approximately reflect the class support's distribution, the F1-score displays an abrupt drop from 0.67 into a relatively stable average of approximately 0.46 right after the first position, due to the combined effect of the recall and precision distributions, naturally impacted by presentation bias. The recall displays a sharp decrease similar to F1's but bigger in magnitude, with values following the second class only achieving close to or less than half of the 0.83 maximum. In contrast, the best precision values are tied to lower interface positions, hinting that the models' predictions are more contextualized in this region.[5]

Specific factors including circumstances in which non-item based interactions correspond to last interaction sequence events, contribute to skew the predictions towards upper positions, with clicks for the top position specifically increasing by 21.3% in these situations. Searching for new points of interest or changing sorting methods, for instance, can signal changes in the intent and objective of the session and introduce 'soft resets', meaning that the contextualization provided by the preceding sequence is negatively impacted. The extreme bias introduced by shorter interaction sequences in this dataset, previously discussed in Figure 4.10, directly impacts the ranking performance, as can

[5]The increased number of false positives for top classes with higher click probabilities can be easily observed in the appendix confusion matrix of Figure D.3.

Table 5.4: M1* architecture test results per class (clicked impression list position). Best and second-best results per metric are boldfaced and underlined, respectively.

Class	Support	MRR	F1	Recall	Precision
1	30172	**0.89689**	**0.66572**	**0.82988**	0.55578
2	12040	<u>0.71069</u>	0.48643	<u>0.52467</u>	0.45338
3	8671	0.61191	0.47487	0.43536	0.52227
4	6842	0.60271	0.45705	0.43350	0.48330
5	5986	0.56370	0.46638	0.39977	0.55964
6	4860	0.51642	0.45120	0.38189	0.55123
7	4170	0.53509	0.44956	0.39592	0.52000
8	3663	0.49696	0.45190	0.37701	0.56390
9	3308	0.53378	0.44234	0.41626	0.47190
10	3040	0.50186	0.45803	0.38684	0.56134
11	2845	0.49474	0.44427	0.38735	0.52079
12	2450	0.49495	0.44926	0.40204	0.50904
13	2321	0.47409	0.46746	0.38690	0.59040
14	2147	0.48151	0.45647	0.39683	0.53720
15	1914	0.49226	<u>0.49059</u>	0.41536	0.59910
16	1849	0.46417	0.44473	0.37426	0.54790
17	1738	0.49790	0.47235	0.40794	0.56092
18	1522	0.48916	0.48589	0.40736	0.60194
19	1428	0.47946	0.48609	0.39776	0.62486
20	1390	0.49808	0.46378	0.41223	0.53006
21	1322	0.45305	0.46242	0.36536	<u>0.62973</u>
22	1236	0.47553	0.46967	0.38835	0.59406
23	1232	0.47116	0.47609	0.37581	**0.64937**
24	1247	0.47141	0.47348	0.38653	0.61090
25	1466	0.51047	0.47332	0.41746	0.54643

be observed in Figure 5.6. Although longer sequences are able to provide more predictive context, the expanded impression position possibilities and reduced number of training samples negatively affect the results.

The model's behavior can be more precisely observed in the nine specific session results presented in Appendix Section D.3, which include the predicted click probabilities for each of the impression items and the attention weights for the sequential interactions. Of the three incorrect prediction examples shown (D.3.4, D.3.6, D.3.9), only D.3.9 contained a target item that had been previously interacted with by the user. In both cases (D.3.6, D.3.9) where the model missed the first impression position target, the previous interactions were located further down in the list, with items in the 12th and 4th positions respectively. In these situations, interaction probabilities with higher items is naturally reduced. It should be noted, however, that the first positions still corresponded to the third and second-ranked item predictions. The remaining session input D.3.4 shows the influence of a 'soft reset' event in the model. After the final destination search event, none of the interacted items is available in the final impression list and the prediction probabilities assume an initialized distribution similar that of the class supports.

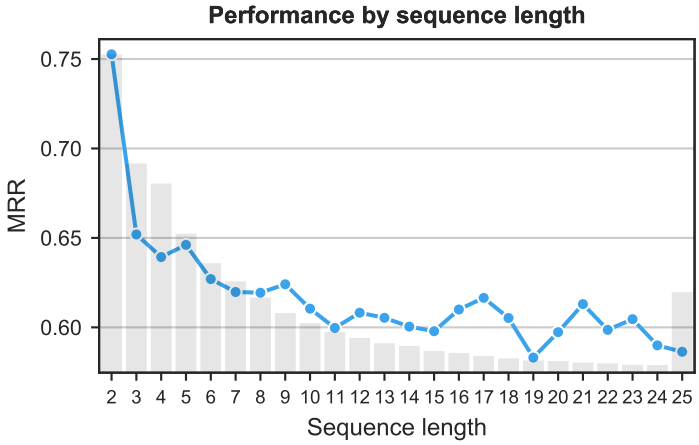


Figure 5.6: Model M1*'s ranking performance by interaction sequence length and relative sample distribution with omitted axes. The visible increased number of 25 time step samples is a product of the sequence thresholding procedure.

5.2.1 Attention performance

M1*'s Hierarchical attention weights were averaged[6] for the different possible maximum interaction sequence sizes and plotted in Figure 5.7's heatmap.

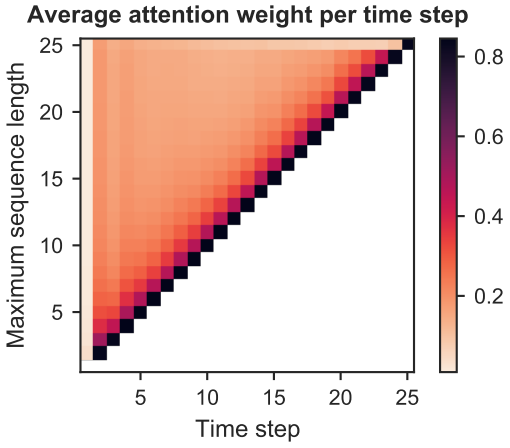


Figure 5.7: Average attention weights per time step for given maximum interaction sequence lengths.

The obtained attention values reflect a behavior similar to that of [64]'s SASREC on their sparsest Amazon Beauty dataset, but much more biased towards the last sequence events, which can be observed in most of the examples in Appendix Section D.3.[7] Further data exploration helped justify this distribution, by revealing that 37.9% of the clicked items correspond to the references of last item-based interactions. Similarly high importance weights attributed to more general last events, as in D.3.4, can possibly result from their 'soft reset' potential discussed in the previous section.

The weights appear less skewed towards final interactions and the sequential branch contributes with a more general sequence representation whenever a given item is re-interacted with after inter-

[6]The average attention for each time step is based on valid, non-padding values only.
[7]The remaining attention types produced similar distributions.

action gaps with other items (skip behaviors [45]), as seen in D.3.6 and D.3.9. In certain situations, like D.3.8, the mechanism is also able to single out specific item interactions.

Table 5.5: Average attention weights by action type. Biggest and second-biggest values for each column are boldfaced and underlined, respectively.

Action type	Avg. attention weight	Last interact. (%)	r_{att}
Interaction item deals	**0.2369**	0.0648	3.66
Interaction item info	<u>0.2001</u>	0.0908	2.20
Search for item	0.1955	0.0387	**5.05**
Interaction item image	0.1794	**0.3590**	0.50
Clickout	0.1751	<u>0.1617</u>	1.08
Interaction item rating	0.1516	0.0602	2.52
Search for POI	0.0980	0.0225	<u>4.36</u>
Filter selection	0.0975	0.1434	0.68
Search for destination	0.0738	0.0449	1.64
Change of sort order	0.0198	0.0140	1.41

Overall, item-based actions were given bigger average attention weights than their more general counterparts, as can be observed in Table 5.5. The quotient between the average attention weight and the normalized last event incidence, r_{att}, was taken for each action, to introduce the last item bias effect. Sorting by r_{att}, although clicks and image interactions get pushed to the list's bottom three, the remaining four item-based actions are still concentrated in the top five. This focus on specific action types is reflected in the examples D.3.5 and D.3.7, for instance, where the item search (9) was emphasized over the final item image (5) and rating (7) interactions.

5.2.2 Model ablation

An ablation study inspired by [6, 58, 62, 64] was performed to assess the impact of individual model components and processing methods. Fourteen model variants based on M1 were tested under the full model setup conditions described at the beginning of Section 5.2, divided into three categories:

- **Model component removal** Single component removal, identified by the respective hyperparameter codes[8]. (**M2**) No impression features, H1; (**M3**) No interaction sequences, H10; (**M4**) No interaction sequence features, H11; (**M5**) No attention, H16; (**M6**) No session features, H20; (**M7**) No filter features, H21; (**M8**) No metadata features, H8; (**M9**) No joint dense layer, H25.
- **Model baselines** With the inability to directly compare full corpus session-based ranking recommenders to M1 (discussed in Section 2.3.3), two modified architectures which influenced its item embedding processing foundation, conditioned on the impression list items, were added as baseline references. (**M10**) Only interaction sequence and impression item embedding inputs, processed by the respective sequential blocks and joint MLP, inspired by GRU4Rec+ [58]; (**M11**) M10 with attention, inspired by NARM's local encoder [62].

[8] Relevant hyperparameter H# codes are retrieved from Table 4.6

- **Processing** Changes in data processing methods. (**M12**) No data augmentation, only last session clicks used for training; (**M13**) Class balancing method assigning more influence to minority classes in the cost function, with the class weights parameter as in [124]; (**M14**) Min-max normalization instead of Quant; (**M15**) Min-max normalization for ordinal features, Quant for the remaining.

Each configuration's results, corresponding to performance averages over five runs with different initialization parameters, are shown in Table 5.6.

Table 5.6: Model ablation results with average values over 5 runs with different initialization parameters (except for M1's which correspond to Table 5.3's 10 run averages) and a batch size of 128. Delta corresponds to the MRR decrease from M1's. Best and second-best values are boldfaced and underlined, respectively.

Model	MRR	Delta (%)	M. F1	Wt. F1	M. Rec.	M. Prec.	Wt. Prec.	Acc.	NLL
M1	**0.65113**	**-**	**0.47346**	**0.51923**	0.41682	0.56471	0.53941	**0.53465**	**1.73079**
M2	0.47368	-27.25	0.08335	0.25107	0.08875	0.12400	0.23646	0.33909	2.42430
M3	0.64039	-1.65	0.46184	0.50972	0.41436	0.53666	0.52150	0.52199	1.77917
M4	0.64251	-1.32	0.46257	0.50870	0.40935	0.54868	0.52322	0.52223	1.77788
M5	0.64659	-0.70	0.46785	0.51421	0.40945	**0.56613**	**0.53992**	0.53046	1.74843
M6	0.64587	-0.81	0.46951	0.51518	0.41549	0.55599	0.53252	0.52811	1.77264
M7	0.64719	-0.61	0.46865	0.51565	0.41242	0.56258	0.53437	0.53025	1.74149
M8	0.64537	-0.89	0.46740	0.51239	0.41431	0.55236	0.53362	0.52878	1.75648
M9	0.64615	-0.77	0.46337	0.51081	0.40950	0.55229	0.53065	0.52554	1.75511
M10	0.45088	-30.75	0.06637	0.22062	0.07429	0.10170	0.20325	0.31191	2.50279
M11	0.45633	-29.92	0.06659	0.22060	0.07409	0.11903	0.21354	0.31525	2.48960
M12	0.63548	-2.40	0.45388	0.50331	0.40206	0.54027	0.52239	0.51871	1.79999
M13	0.59687	-8.33	0.43000	0.48901	**0.43655**	0.43245	0.51323	0.48034	2.01977
M14	0.63976	-1.75	0.44603	0.50290	0.38818	0.54916	0.52676	0.52114	1.78524
M15	0.64673	-0.68	0.47272	0.51866	0.41932	0.55976	0.53432	0.53186	1.76813

It can be noted that the overall most impactful model component corresponds, by a wide margin, to the impression features input, reflecting the optimization distribution analysis of Section 5.1.1. With macro and weighted F1 values 82.4% and 51.64% lower than M1, respectively, M2 is significantly worse classification-wise. Its also considerable ranking performance decrease of 27.25% is followed by M3's lack of sequential interaction input, with a much less significant (16.5 times smaller) drop.

On the other hand, M7 and M5's lack of filter features and attention mechanism, respectively, result in the smallest MRR decreases. The 0.70% ranking performance gain attributed to the Hierarchical attention implementation is complemented by improved minority class identification, however at a cost of slightly lower precision, with the increased prediction count in this class region consequently resulting in a larger amount of false positives.

The simplest baseline M10 and M11 models seem to over-rely on sequence-induced position information, obtaining only marginally better results than a model limited to top position predictions (POS, in the next Section 5.2.3). With a 30.75% ranking decrease, 7.1 and 2.35 times smaller macro and weighted F1s, compared to M1's average values, M10 was the worst performing configuration.

The Hierarchical attention mechanism's inclusion in M11 improved its ranking performance by 1.21%.

Regarding processing methods, the increased training information provided by the data augmentation procedure was found to have a key positive effect in the overall performance of the model, corresponding to the second most important M1 ranking component. Although comparatively less influential in the ranking, proper data normalization was likewise found to have an important role in the results, showing clear benefits of the non-linear quantile transformation. Min-max usage in the whole dataset was linked to the third biggest MRR decrease when considering single M1 component edits. Its usage is disadvantageous even when the transformation is reduced to ordinal features. The attempt to balance the problem using the class weights parameter, which emphasizes minority predictions in the loss function, saw an increase in the ability of the classifier to find minority samples, as indicated by the 4.73% macro recall increase over M1's, but also a simultaneously significant decrease in every other metric, including the fourth biggest ranking-wise. Alternatives, such as different sampling methods, were not explored as most, especially binary extensions that balance according to the biggest or smallest class, might not be suitable for multiclass settings [125]. Imbalance impact on evaluation metrics can also be considered in follow-up work [126] (FW.11).

5.2.3 Baseline comparison

The development of suitable baselines to further contextualize M1's performance faced numerous challenges, expanding on those mentioned in Section 2.3.3.

User and utility-based solutions, including matrix factorization methods, are not suitable for sequential session-based environments and were not considered [70]. Neighborhood-based approaches are limited by their often large memory requirements, aggravated by RSC19's large item space dimensionality. Session-based kNN (s-kNN) and its variants have obtained decent results in some session-based problems [69, 70] but their output scores based on item interaction occurrence and session similarity are oriented for next-item single-interactions predictions, predominant in e-commerce datasets. Since most impression items are not interacted with in RSC19's sequential events, s-kNN would not be able to produce recommendation scores for these without any type of modification. Additionally, only an adapted version such as [70]'s that accounts for a mere 500 neighbors within the 1000 most recent sessions would be feasible as producing pairwise similarity scores between the full session corpus would require an almost 66 billion entry matrix (FW.12).[9]

Therefore only simple, static baselines without learnable parameters were considered, based on those used in [55, 58, 62, 67] and the one provided by the challenge's organizers, (**CL-L**), which always recommends the most globally clicked items. The problem with CL-L is that it does not account for causality, using future click information to make predictions. This was fixed in (**CL**), which only uses each item's previously recorded clicks until the relevant click events' timestamps. (**S-CL**) applies the same concept to local session-based click information, using CL values to break ties. (**POP**) extends CL to make use of the remaining views, interactions and dwell time counts

[9] $N_{\text{test}} \cdot N_{\text{train}} + \frac{1}{2} N_{\text{test}} \cdot (N_{\text{test}} - 1) =$ 65 979 004 464 - similarity scores between test sessions and previous train sessions plus the lower triangular pairwise similarity matrix entries between test sessions.

for a measure of global item 'popularity', with (**S-POP**) doing the same for S-CL. Given the much discussed bias towards top positions, (**POS**), which recommends only the top position's items, was added. Following the results of Section 5.2.1, the (**LAST**) model was created to recommend the last interaction sequence reference item, using S-POP values for non item-based actions. Finally, (**PRICE**) always recommends the cheapest impression item and a random predictor, (**RAND**), was added for reference.

Performance results for each baseline model are presented in Table 5.7, sorted by MRR.

Table 5.7: Baseline results and comparison to the best model M1*.

Baseline	MRR	M. F1	Wt. F1	M. Rec.	M. Prec.	Wt. Prec.	Acc.
M1*	**0.65148**	**0.47277**	**0.52095**	0.42011	**0.55582**	**0.53698**	**0.53505**
Delta	+13.72%	+11.88%	+6.82%	-3.21%	+34.71%	+8.71%	+10.64%
LAST	0.57288	0.42255	0.48770	**0.43406**	0.41260	0.49395	0.48358
S-POP	0.48889	0.24980	0.34504	0.24834	0.25176	0.34591	0.34446
POS	0.42420	0.01736	0.12030	0.04000	0.01109	0.07682	0.27717
CL-L	0.27397	0.09071	0.15718	0.09457	0.08867	0.17229	0.14914
S-CL	0.25488	0.09039	0.15016	0.09275	0.08914	0.16151	0.14376
POP	0.25439	0.07015	0.12223	0.07613	0.06900	0.15108	0.11035
CL	0.23116	0.07278	0.13066	0.07587	0.07138	0.14541	0.12283
PRICE	0.18965	0.05521	0.08882	0.07147	0.06135	0.13193	0.07691
RAND	0.15359	0.03246	0.04902	0.04040	0.04094	0.11612	0.04071

Obtaining only slightly better results than the random predictor, PRICE was the second-worst performing model. Although the provided baseline, CL-L, secured the fourth-best ranking result, it was underwhelming when compared to the 1.55 times bigger value required to enter the top three. Unsurprisingly, the modified CL version performed worse, with relative drops of 15.6%, 19.8% and 16.9% for the MRR, macro F1 and weighted F1 respectively. S-CL's results were, however, only marginally worse than CL-L's, demonstrating the impact of local, session-based information in RSC19. POP's extension of CL resulted in a 10% MRR increase but a slight overall decrease in the remaining classification metrics, aside from weighted precision. Nevertheless, as with click information, the session-based S-POP version performed better. In fact, the local interaction popularity indicators proved to be important enough for the increase to be much more significant in both ranking (1.92 times bigger MRR) and classification (3.56 and 2.82 times multipliers for macro and weighted F1s), leading to the second-best baseline result. The benefits of having multiple evaluation metrics are apparent in POS' case, where the MRR (only 0.065 lower than S-POP, 1.55 times CL-L's) and also elevated accuracy might lead to misleading positive conclusions driven by the biased nature of the label distribution.[10] With the exception of weighted F1, which is still decently influenced by the correct first-class predictions, the remaining classification values expectedly demonstrate the opposite, with results inferior to those of RAND. In the end, the best baseline result was achieved by LAST, with considerably better values than S-POP in every metric and even obtaining a narrowly better macro

[10]Still, when the focus is placed solely on MRR performance, POS already beats the provided baseline by a decent margin.

recall than M1*. Nonetheless, the remaining M1* delta values show that the deep learning model still significantly outperforms the baseline, whose MRR is lower than even the average returned by the Bayesian mode on the smaller optimization dataset, from Table 5.1. Regardless, given how simple the logic behind each of the baselines is, the obtained results are impressive and signal the possibility for potential competitive performances with further focus on more complex configurations.

5.2.4 RecSys19 performance

Applying [113]'s data split resulted in the ability to use their noted average difference between local and deployed submission results in the challenge's online test set of +1.7% to generate a final MRR estimate for M1* of 0.66255. This value is represented in the leaderboard's result distribution by position of Figure 5.8.

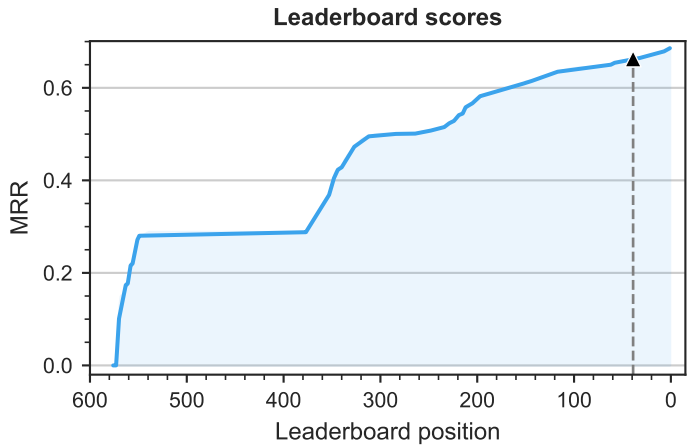


Figure 5.8: Leaderboard MRR distribution for the 575 submissions. The predicted M1* score on the online test set is signaled with a triangle.

The MRR distribution for the 575 submissions displays two significant value step concentrations starting at approximately positions 550 and 310. The former, and also more predominant one, corresponds to CL-L baseline results, which are in fact 4.83% higher than the local test set values obtained in the previous Section 5.2.3, while the latter more closely approximates the predicted S-POP result, with a 2.35% difference. M1*'s estimated performance would obtain the 39th position (one higher than M1's average), corresponding to the 93.5% and 85.7% quantiles when considering the full submission space and only values above CL-L's, respectively.

Although the gap to the top ten was still of a significant, but expected, 2.11%, as displayed in Table 5.8, the obtained results are quite decent, especially when taking into account the unfeasible computational requirements, model complexity, and feature generation focus of the best-performing entries, dominated by decision tree ensembles, versus M1*'s representation learning emphasis.[11]

[11] For reference, LogicAI's first place model [127] consisted of a 37 Multiple Additive Regression Tree (MART) ensemble with 250 different features, plus augmentations, trained on virtual machines with 96 vCPUs and 624GB RAM; Layer6's second place result [128] was obtained by a linear blend of LSTM, Transformer, and Gradient Boosting Machine (GBM) models, with 330 features, running on a 256GB RAM machine with a Titan V GPU; the remaining entries in the top five [129–131] used a stacking GBM ensemble, an [42]-inspired neural network and GBM ensemble, and a GBM, Doc2Vec, MF and BPR hybrid, respectively, with the latter making use of 518 different features.

Table 5.8: M1* compared to top leaderboard scores.

Position	MRR	Delta (%)
1	0.68571	+3.50
7	0.67884	+2.46
10	0.67655	+2.11
20	0.66860	+0.91
30	0.66497	+0.36
Predicted M1* score		
39	0.66255	-

Some of the feature-based strategies used by the preceding models, such as the generation of additional ranking, augmentation, aggregation and boolean inputs, which frequently rank the highest in decision tree model importance, could be used to most likely easily close the small 0.36% and 0.91% intervals to the top 30 and 20 positions without the need for architectural changes. Other transformations such as the input of squared, square root and log versions of continuous features, are reported to increase the network's expressive power in [7], could also be tested for the same effect (FW.10).

With respect to possible relevant structural changes for ranking performance improvement, besides ensemble implementations, the adoption of the transformer architecture, which returned the best neural network-based results in [128], would be a priority. The implementation of different attention types besides self-attention, such as those used in [113]'s 7th place solution, could also be used to enhance the model and enable it to use the sequential interactions (Keys, Values) to attend over the impression items (Query) directly, for instance (FW.2). Factorization-based interaction layers like those in [42, 130] and residual connections, used in [54, 64], also seem to offer significant advantages in e-commerce datasets (FW.14).

The exploration of additional regularization strategies is also necessary. The training curves returned by the M1 test runs from Section 5.2, like those from Appendix Figure D.1, show a quick overfitting behavior, which [113] also noted in their L2-regularized model. Higher dropout rates and lower initial learning rates lead to curves similar to those in Figure D.2, which seem better-behaved but lead to worse results (FW.5).

All of the enumerated changes would, however, still have difficulty placing the new model near the top ten, as the higher 2.11% MRR delta is also linked to the usage of non-causal information by the best performing models, namely the time difference between clicks and previous sequential events which is consistently reported as a top contributing feature [113, 127, 128, 130, 131], not used by M1.

Chapter 6

Conclusions

This final chapter presents a review of this work's achievements, alongside notable future research directions.

6.1 Achievements

This work encompassed the development of a deep learning re-ranking recommender system with self-attention for session-based environments, subject to an automated modular architectural bayesian optimization process, successfully accomplishing the objectives proposed in Section 1.3.

Chapter 5's results translate into the following briefly summarized answers to the research questions associated with each objective:

RQ.1 The developed model was able to obtain a concentration of 75% of the clicked items within the first four predicted rank positions in the local test set, performing considerably better (+29.9% MRR) than an adapted state-of-the-art attentional architecture based solely on sequential interaction information. The click-based predictive objective is heavily influenced by the imbalanced data distribution reflecting the effects of presentation and other implicit biases, promoting better ranking and classification performances in top classes: The MRR values approximate the decreasing class support distribution; The F1 values experience a more drastic negative drop right after the first class, with the model's ability to detect lower positioned class instances decreasing but with the proportion of correct positive predictions slightly increasing instead. The higher number of false positives associated with higher class predictions were also found to be correlated with additional sequential circumstances, such as last events with non-item interactions and events with major interface control, including sort changes and new destination searches.

The incremental addition of properly transformed feature inputs is linked to the most prominent positive performance impact, particularly those which directly describe the presented impression items and their interaction profiles. This property emphasized the benefits of increased personalization and the power of representation learning, which would still prove limited against the ability of models exploiting hundreds of features and subsequent augmen-

tations. Besides other structural components like the self-attention module, the training data augmentation process also contributed to the 13.7% ranking gap over the best performing simple baseline that leveraged data bias, combining last interaction information with session-based markers.

The estimated final MRR of 0.66255 for the RecSys19 test set, which translates into a 0.91% ranking difference to the leaderboard's top 20, can also be considered relatively good when taking into account the structural and computational complexity of better-ranked models, not to mention the unfeasible results supported by non-causal information, reported by some.

RQ.2 Although the 0.7% ranking performance increase attributed to the self-attention mechanism was not as significant as that of other model components, it promoted an increase in less supported class region predictions, leading to better recall values at the expense of slightly lower precision. Furthermore, the learned attention weight distributions provided insight into the model's sequential branch prediction contribution, improving recommendation interpretability. Two of the most interesting data characteristics captured by the mechanism include the relative higher importance of last sequential events[1] and the greater impact of specific item-based actions, including item searches and deal interactions. In addition, some sample examples showed skip-behavior awareness and the mechanism's ability to isolate interactions with specific items.

RQ.3 The bayesian optimization mode provided a rapid MRR increase of 3.29% in its first 85 runs over a previous relatively stable maximum value, while also obtaining a superior overall average ranking performance (+1.33x) with four times smaller standard deviation compared to the random datapoint initialization. Additionally, the optimizer was able to automatically avoid some of the worst-performing hyperparameter configurations while maintaining a decently exploratory behavior with some focused runs on potential high-value targets, displayed in the hyperparameter-specific distributions. With such a complex configuration space, the random mode was nevertheless found to provide significant benefits as the optimization's foundation, outperforming a more exploitative fully-bayesian approach, highly dependent on initial run results.

While the predictive representations learned by the model are undeniably powerful, the re-ranking objective's applicability in this domain is limited by the lack of dynamic impression list presentation [131]. Unlike in YouTube or Amazon, where recommendations in dedicated video or product pages can change with every interaction, additional insight in trivago's recommendations can only be reflected after drastic interface events such as sort changes. Furthermore, although RSC19 most closely resembled a real-world scenario with the availability of metadata and other contextual features mostly disregarded in other e-commerce datasets (which often focus solely on item ID sequences),

[1] This behavior is verified in other sparse datasets [64] and includes non-item-specific interactions, which generally prompt probability distributions loosely proportional to the class supports.

the lack of crucial information such as item-specific location data or interface details[2] hindered the modeling potential and prevented full ranking applications, which would have had been arguably more useful.

6.2 Future work

Future work topics mentioned throughout the document are presented in order of appearance. Starred entries are mostly framed for different datasets or environments with data unavailable in RSC19.

FW.1 **Multi-objective modeling** Explore multi-objective modeling to estimate different types of user behavior, enabling the creation of various parallel predictive intent representations. One possible approach is the use of Mixture-of-Experts model achitectures, such as those used in [34, 50] to process different temporal ranges and predict numerous engagement and satisfaction tasks on YouTube.

FW.2 **Attention and transformer architectures** It would be interesting to evaluate the impact of additional mechanism types such as multi-head attention and non-self-attention approaches, that could expand on the observed scaled dot behavior with a focus on different sequence properties and provide more inter-branch interaction by attending to sequences with respect to other contextual information. A further step would consist of a transformer architecture implementation, prevalent in the most recent state-of-the-art approaches in recommendation (and a variety of other fields, such as NLP), reported to have contributed with the best DL-based results in RecSys19 [128].

FW.3 **Other losses** While its combination with the softmax function introduces list-based properties, the negative log loss is still essentially a pointwise loss in the learning-to-rank framework. Different pairwise and listwise losses might achieve better results at the expense of efficiency. Alternatively, a regression-based solution to the problem can also be explored for more control over specific interaction weights or penalties contributing to a given item score, for instance.

FW.4 **Bayesian optimization** The usage of a bayesian optimizer proved advantageous given the complexity of the conditional hyperparameter space. However, the impact of several of its components was left unexplored. In the future, more focus would be given to the GP kernel's hyperparameter tuning process and to the study of other kernel types, including those of conditional nature. Additionally, other acquisition functions, and the implementation of an adaptive UCB β would also be explored.

FW.5 **Regularization** Consider additional regularization methods to tackle overfitting, including L2, L1, batch and layer normalizations.

[2] Interface information was only available at click time and is challenging to estimate for other time steps. In some situations, interacted items were not present in impression lists even without any interface change signal, which constrained navigation evolution analysis, for example.

FW.6 **User profiles*** Expand model to leverage user specific features and long-term signals in different sequential environment. Make use of user-specific exploration-exploitation components (ranking demotion, engagement strategies, and further personalization) [7].

FW.7 **Temporal trends*** Consider greater timespan data to introduce item evolution and time-windowed popularity indicators [44, 50].

FW.8 **Interface features*** Descriptive platform interface features have an unsurprisingly considerable influence in predictive tasks but were quite limited in the RecSys19 dataset. Thumbnails, for instance, play a major role in YouTube users' item selection process. Different types of visual information, for example, can be easily processed by including a parallel convolutional branch in an adapted M1-type architecture. Most user decisions are also made in a relative environment against other selection possibilities so relative comparison features such as those used in [127] for items in the vicinity of one currently being interacted with should also be considered in the future.

FW.9 **Embeddings** Focus on embedding pre-training or initialization strategies (with possible incorporation of the metadata information for item embeddings, location information for city embeddings), and visualization exploration.

FW.10 **Data transformation and outliers** Extend feature generation process to include further personalization with relative rank, boolean transformations (especially important as continuously growing counters become unsustainable in larger timespans) and additional transformed continuous inputs for expressiveness (squares, square roots). Input decorrelation and other normalization methods should also be investigated, along with outlier impact and processing.

FW.11 **Data imbalance and bias*** Check additional data balancing methods for both interaction sequence sizes and clicked positions (classes), including sampling methods, and further explore imbalanced metric impact. In addition, expand study on bias influence and removal methods (with adversarial training as in [34] instead of only inputting position as a feature, for selection/presentation bias).

FW.12 **Baselines** Implement additional baseline algorithms, such as modified s-kNN variants, and other more complex non-DL architectures, including session-based factorization, MDP, and the Recsys19 dominant decision tree boosting methods (XGBoost, LightGBM).

FW.13 **Ranking and A/B*** Extend the task to full ranking and incorporate A/B setting experiments, introducing online dynamics.

FW.14 **Model architecture** Adapt the model to support residual connections, factorization layers, and alleviate some hyperparameter range limits in a less constrained computational environment.

FW.15 **Time efficiency** Focus more deeply on processing and serving time efficiency, targeting online settings.

FW.16 **Additional topics*** Delve into cold-start warm up and more recent recommendation topics including diversity impact, gamification introduction, cross-domain analysis, conversational systems, and explainability.

Bibliography

[1] D. Jannach and M. Jugovac. Measuring the business value of recommender systems. *ACM Transactions on Management Information Systems (TMIS)*, 10(4):1–23, 2019.

[2] A. Frangoul. With over 1 billion users, here's how YouTube is keeping pace with change, March 2018. URL https://www.cnbc.com/2018/03/14/with-over-1-billion-users-heres-how-youtube-is-keeping-pace-with-change.html. Last accessed on 2020-03-12.

[3] J. E. Solsman. YouTube's AI is the puppet master over most of what you watch, January 2018. URL https://www.cnet.com/news/youtube-ces-2018-neal-mohan/. Last accessed on 2020-03-16.

[4] N. McAlone. Why Netflix thinks its personalized recommendation engine is worth 1 billion dollars per year, June 2016. URL https://www.businessinsider.com/netflix-recommendation-engine-worth-1-billion-per-year-2016-6. Last accessed on 2020-06-20.

[5] I. MacKenzie, C. Meyer, and S. Noble. How retailers can keep up with consumers, October 2013. URL https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers. Last accessed on 2020-06-19.

[6] S. Zhang, L. Yao, A. Sun, and Y. Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[7] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[8] M. Neal. Beyond trust: Psychological considerations for recommender systems. In *Proceedings of the International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government (EEE)*, page 1, 2011.

[9] A. Alslaity and T. Tran. Towards persuasive recommender systems. In *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*, pages 143–148. IEEE, 2019.

[10] N. Seaver. Captivating algorithms: Recommender systems as traps. *Journal of Material Culture*, 24(4): 421–436, 2019.

[11] D. Jannach, P. Resnick, A. Tuzhilin, and M. Zanker. Recommender systems—beyond matrix completion. *Communications of the ACM*, 59(11):94–102, 2016.

[12] P. Knees, Y. Deldjoo, F. Bakhshandegan Moghaddam, J. Adamczak, G.-P. Leyson, and P. Monreal. RecSys Challenge 2019: Session-based Hotel Recommendations. In *Proceedings of the Thirteenth ACM*

Conference on Recommender Systems, RecSys '19, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6243-6/19/09. doi: 10.1145/3298689.3346974. URL https://doi.org/10.1145/3298689.3346974.

[13] J. Adamczak. RecSys Challenge 2019, March 2019. URL https://tech.trivago.com/2019/03/11/recsys-challenge-2019/. Last accessed on 2019-09-24.

[14] Our Product. URL https://company.trivago.com/our-product/. Last accessed on 2020-03-12.

[15] A. Rajaraman and J. D. Ullman. *Mining of massive datasets*, chapter 9. Cambridge University Press, 2011.

[16] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[17] J. Bennett, S. Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.

[18] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. 2020. https://d2l.ai.

[19] Y. 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*, pages 426–434, 2008.

[20] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[21] Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, 2009.

[22] Y. Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81(2009):1–10, 2009.

[23] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[24] A. Mnih and R. R. Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[25] S. Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[26] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[27] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[28] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*, pages 3146–3154, 2017.

[29] H. Fang, D. Zhang, Y. Shu, and G. Guo. Deep Learning for Sequential Recommendation: Algorithms, Influential Factors, and Evaluations. *arXiv preprint arXiv:1905.01997*, 2019.

[30] X. Wang, M. Bendersky, D. Metzler, and M. Najork. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 115–124, 2016.

[31] A. Sinha, D. F. Gleich, and K. Ramani. Deconvolving feedback loops in recommender systems. In *Advances in neural information processing systems*, pages 3243–3251, 2016.

[32] T. Joachims, A. Swaminathan, and T. Schnabel. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 781–789, 2017.

[33] H.-H. Chen, C.-A. Chung, H.-C. Huang, and W. Tsui. Common pitfalls in training and evaluating recommender systems. *ACM SIGKDD Explorations Newsletter*, 19(1):37–45, 2017.

[34] Z. Zhao, L. Hong, L. Wei, J. Chen, A. Nath, S. Andrews, A. Kumthekar, M. Sathiamoorthy, X. Yi, and E. Chi. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 43–51, 2019.

[35] O. Chapelle and Y. Zhang. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th international conference on World wide web*, pages 1–10, 2009.

[36] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[37] Re-ranking, February 2020. URL `https://developers.google.com/machine-learning/recommendation/dnn/re-ranking`. Google Recommendation Course. Last accessed on 2020-06-01.

[38] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618. URL `http://www.deeplearningbook.org`.

[39] G. K. Dziugaite and D. M. Roy. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

[40] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[41] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[42] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1754–1763, 2018.

[43] M. Quadrana, P. Cremonesi, and D. Jannach. Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018.

[44] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 495–503, 2017.

[45] J. Tang and K. Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 565–573, 2018.

[46] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820, 2010.

[47] R. He and J. McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 191–200. IEEE, 2016.

[48] A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, and E. H. Chi. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 46–54, 2018.

[49] Y. Li, T. Liu, J. Jiang, and L. Zhang. Hashtag recommendation with topical attention-based LSTM. Coling, 2016.

[50] J. Tang, F. Belletti, S. Jain, M. Chen, A. Beutel, C. Xu, and E. H. Chi. Towards neural mixture recommender for long range dependent user sequences. In *The World Wide Web Conference*, pages 1782–1793, 2019.

[51] C. Zhou, J. Bai, J. Song, X. Liu, Z. Zhao, X. Chen, and J. Gao. Atrank: An attention-based user behavior modeling framework for recommendation. *arXiv preprint arXiv:1711.06632*, 2017.

[52] T. Chen, H. Yin, H. Chen, R. Yan, Q. V. H. Nguyen, and X. Li. Air: Attentional intention-aware recommender systems. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 304–315. IEEE, 2019.

[53] C. Pei, Y. Zhang, Y. Zhang, F. Sun, X. Lin, H. Sun, J. Wu, P. Jiang, J. Ge, W. Ou, et al. Personalized re-ranking for recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 3–11, 2019.

[54] S. Sun, Y. Tang, Z. Dai, and F. Zhou. Self-Attention Network for Session-Based Recommendation With Streaming Data Input. *IEEE Access*, 7:110499–110509, 2019.

[55] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[56] D. Ben-Shimon, A. Tsikinovsky, M. Friedmann, B. Shapira, L. Rokach, and J. Hoerle. Recsys challenge 2015 and the yoochoose dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 357–358, 2015.

[57] B. Hidasi and A. Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 843–852, 2018.

[58] Y. K. Tan, X. Xu, and Y. Liu. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 17–22, 2016.

[59] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 130–137, 2017.

[60] B. Hidasi, M. Quadrana, A. Karatzoglou, and D. Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 241–248, 2016.

[61] E. Smirnova and F. Vasile. Contextual sequence modeling for recommendation with recurrent neural networks. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, pages 2–9, 2017.

[62] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428, 2017.

[63] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang. STAMP: short-term attention/memory priority model for session-based recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1831–1839, 2018.

[64] W.-C. Kang and J. McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206. IEEE, 2018.

[65] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252, 2005.

[66] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26:3111–3119, 2013.

[67] M. F. Dacrema, P. Cremonesi, and D. 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*, pages 101–109, 2019.

[68] X. Amatriain and J. Basilico. Netflix Recommendations: Beyond the 5 stars (Part 1), April 2012. URL `https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429`. Last accessed on 2020-05-02.

[69] M. Ludewig, N. Mauro, S. Latifi, and D. Jannach. Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 462–466, 2019.

[70] M. Ludewig and D. Jannach. Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, 28(4-5):331–390, 2018.

[71] Class notes for CS231n: Convolutional Neural Networks for Visual Recognition, Spring 2020, Dept. of Comp. Sci., Stanford University, Palo Alto, CA, USA. URL `https://cs231n.github.io/`. Last accessed on 2020-04-16.

[72] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[73] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. doi: 10.1037/h0042519.

[74] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL http://neuralnetworksanddeeplearning.com/. Last accessed on 2019-11-03.

[75] Lecture notes for CS224n: Natural Language Processing with Deep Learning, Winter 2020, Dept. of Comp. Sci., Stanford University, Palo Alto, CA, USA. URL http://web.stanford.edu/class/cs224n/readings/. Last accessed on 2020-04-02.

[76] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[77] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-24796-5. doi: 10.1007/978-3-642-24797-2. URL http://link.springer.com/10.1007/978-3-642-24797-2.

[78] C. Olah. Understanding LSTM Networks, 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs. Last accessed on 2019-10-14.

[79] A. Karpathy. The unreasonable effectiveness of recurrent neural networks. 2015. URL http://karpathy.github.io/2015/05/21/rnn-effectiveness. Last accessed on 2019-12-21.

[80] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[81] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[82] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

[83] A. Madsen. Visualizing memorization in RNNs. *Distill*, 2019. doi: 10.23915/distill.00016. URL https://distill.pub/2019/memorization-in-rnns. Last accessed on 2020-04-02.

[84] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[85] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[86] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[87] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[88] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[89] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[90] J.-C. Lévesque, A. Durand, C. Gagné, and R. Sabourin. Bayesian optimization for conditional hyperparameter spaces. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 286–293. IEEE, 2017.

[91] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[92] F. Berkenkamp, A. P. Schoellig, and A. Krause. No-regret bayesian optimization with unknown hyperparameters. *arXiv preprint arXiv:1901.03357*, 2019.

[93] M. Hoffman and A. Navarro. Bayesian optimization, connections, applications, and other sundry items, October 2015, presented at MLG RCC, University of Cambridge, Cambridge, England. URL `http://cbl.eng.cam.ac.uk/pub/Intranet/MLG/ReadingGroup/rcc_bayesopt_slides1.pdf`. Last accessed on 2020-02-13.

[94] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[95] Class notes for CSE 515T: Bayesian Methods in Machine Learning, Spring 2015, Dept. of Comp. Sci. and Eng., Washington University, St. Louis, MO, USA. URL `https://www.cse.wustl.edu/~garnett/cse515t/spring_2015/`. Last accessed on 2020-04-05.

[96] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[97] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[98] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[99] S. Wager, S. Wang, and P. S. Liang. Dropout training as adaptive regularization. In *Advances in neural information processing systems*, pages 351–359, 2013.

[100] L. Weng. Attention? Attention! *lilianweng.github.io/lil-log*, 2018. URL `http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html`. Last accessed on 2020-04-25.

[101] C. Olah and S. Carter. Attention and Augmented Recurrent Neural Networks. *Distill*, 2016. doi: `10.23915/distill.00001`. URL `http://distill.pub/2016/augmented-rnns`. Last accessed on 2020-04-16.

[102] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[103] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[104] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[105] J. Cheng, L. Dong, and M. Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[106] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[107] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[108] Metrics and scoring: quantifying the quality of predictions, 2020. URL `https://scikit-learn.org/stable/modules/model_evaluation.html`. Scikit-learn 0.23.2 documentation. Last accessed on 2020-06-01.

[109] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from `https://www.tensorflow.org/`.

[110] W. McKinney. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. "O'Reilly Media, Inc.", 2012.

[111] S. S. Haykin. *Neural networks and learning machines*. Pearson Education, third edition, 2009.

[112] Representation: Qualities of Good Features, February 2020. URL `https://developers.google.com/machine-learning/crash-course/representation/qualities-of-good-features`. Google Machine Learning Crash Course. Last accessed on 2020-04-08.

[113] R. Gama and H. Fernandes. An attentive RNN model for session-based and context-aware recommendations: a solution to the RecSys challenge 2019. In *Proceedings of the Workshop on ACM Recommender Systems Challenge*, pages 1–5, 2019.

[114] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[115] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[116] O. Barkan and N. Koenigstein. Item2vec: neural item embedding for collaborative filtering. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2016.

[117] F. Vasile, E. Smirnova, and A. Conneau. Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 225–232, 2016.

[118] A. F. Zuur, E. N. Ieno, and C. S. Elphick. A protocol for data exploration to avoid common statistical problems. *Methods in ecology and evolution*, 1(1):3–14, 2010.

[119] Preprocessing data, 2020. URL `https://scikit-learn.org/stable/modules/preprocessing.html`. Scikit-learn 0.23.2 documentation. Last accessed on 2020-06-01.

[120] Z. HG. Keras Self-Attention, June 2020. URL `https://github.com/CyberZHG/keras-self-attention`. CyberZHG github repository. Last accessed on 2020-07-02.

[121] M. Kränkel and H.-E. Lee. Text Classification with Hierarchical Attention Networks, February 2019, Information Systems Seminar, Humboldt-Universität zu Berlin, Berlin, DE. URL `https://humboldt-wi.github.io/blog/research/information_systems_1819/group5_han/`. Last accessed on 2020-07-03.

[122] tf.keras.layers.Attention, 2020. URL `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Attention`. TensorFlow Core API documentation. Last accessed on 2020-08-15.

[123] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, et al. Keras Tuner, 2019. URL `https://github.com/keras-team/keras-tuner`.

[124] Classification on imbalanced data, 2020. URL `https://www.tensorflow.org/tutorials/structured_data/imbalanced_data`. TensorFlow Core API documentation. Last accessed on 2020-02-03.

[125] B. Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.

[126] A. Luque, A. Carrasco, A. Martín, and A. de las Heras. The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognition*, 91:216–231, 2019.

[127] P. Jankiewicz, L. Kyrashchuk, P. Sienkowski, and M. Wójcik. Boosting algorithms for a session-based, context-aware recommender system in an online travel domain. In *Proceedings of the Workshop on ACM Recommender Systems Challenge*, pages 1–5, 2019.

[128] M. Volkovs, A. Wong, Z. Cheng, F. Pérez, I. Stanevich, and Y. Lu. Robust contextual models for in-session personalization. In *Proceedings of the Workshop on ACM Recommender Systems Challenge*, pages 1–5, 2019.

[129] Z. Wang, Y. Gao, H. Chen, and P. Yan. Session-based item recommendation with pairwise features. In *Proceedings of the Workshop on ACM Recommender Systems Challenge*, pages 1–5, 2019.

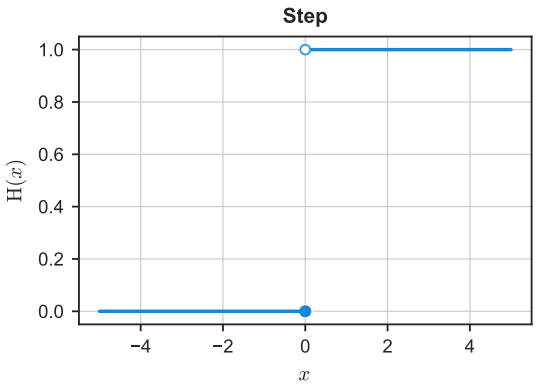
[130] H. Kung-Hsiang, F. Yi-Fu, L. Yi-Ting, L. Tzong-Hann, C. Yao-Chun, L. Yi-Hui, and L. Shou-De. A-HA: A hybrid approach for hotel recommendation. In *Proceedings of the Workshop on ACM Recommender Systems Challenge*, pages 1–5, 2019.

[131] M. Ludewig and D. Jannach. Learning to rank hotels for search and recommendation from session-based interaction logs and meta data. In *Proceedings of the Workshop on ACM Recommender Systems Challenge*, pages 1–5, 2019.

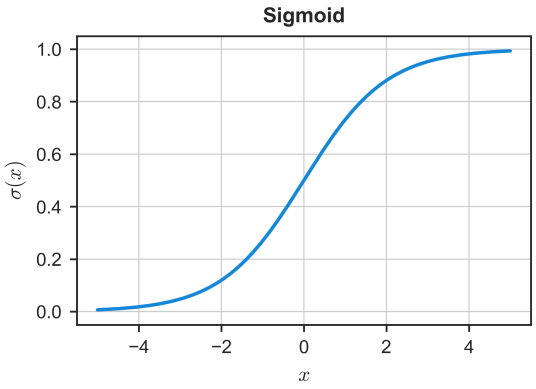
Appendix A

Mathematical Background

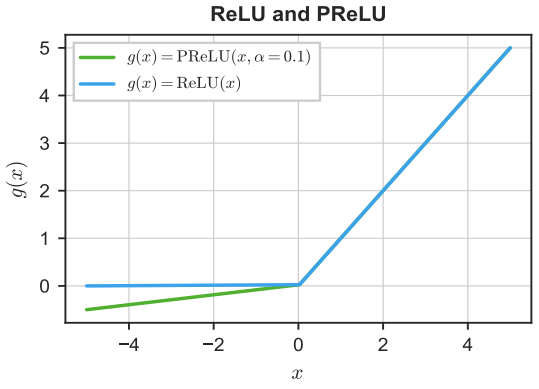
A.1 Activation functions



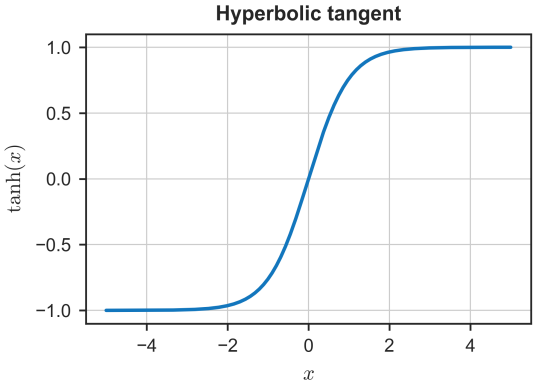
(a) Heaviside step function with $\mathrm{H}(0) = 0$.



(b) Logistic sigmoid function, $\sigma(x) = 1/(1 + \exp(-x))$.



(c) The parametric ReLU is given by $\mathrm{PReLU(x)=max(0,x)+\alpha min(0,x)}$, where the α is learned during training. Hard-setting α to 0 or 0.01 returns the original ReLU and Leaky ReLU versions, respectively.



(d) Hyperbolic tangent function, $\tanh(x)$.

Figure A.1: Referenced non-linear neural network activation functions plotted in the interval $x \in [-5, 5]$.

A.2 Probability and Information theory

Definitions based on Goodfellow et al. [38]. Probability mass functions are denoted with $P(\cdot)$, and probability density functions with $p(\cdot)$. In A.7, Q represents a separate probability distribution. f and g are arbitrary functions. Subequations in A.1 and A.4 denote continuous and discrete cases, respectively.

Marginal probability distribution

$$p(x) = \int p(x, y)\, dy \tag{A.1a}$$

$$P(x) = \sum_{y} P(x, y) \tag{A.1b}$$

Conditional probability

$$p(x|y) = \frac{p(x, y)}{p(y)} \tag{A.2}$$

Bayes' rule

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{A.3}$$

Expected value

$$\mathrm{E}_{x\sim p}[f(x)] = \int p(x)f(x)\, dx \tag{A.4a}$$

$$\mathrm{E}_{x\sim P}[f(x)] = \sum_{x} P(x)f(x) \tag{A.4b}$$

Covariance

$$\mathrm{Cov}(f(x), g(y)) = \mathrm{E}[(f(x) - \mathrm{E}[f(x)])(g(y) - \mathrm{E}[g(y)])] \tag{A.5}$$

Variance

$$\mathrm{Var}(f(x)) = \mathrm{Cov}(f(x), f(x)) = \mathrm{E}[(f(x) - \mathrm{E}[f(x)])^2] \tag{A.6}$$

Cross-entropy

$$H(P, Q) = -\mathrm{E}_{x\sim P}[\log Q(x)] \tag{A.7}$$

A.2.1 Gaussian Identities

Based on Rasmussen and Williams [91].

The multivariate Gaussian (or Normal) distribution $\mathbf{x} \sim \mathcal{N}(\mathbf{m}, \Sigma)$, in D dimensions, has a joint probability density given by

$$p(\mathbf{x}|\mathbf{m}, \Sigma) = (2\pi)^{-\mathrm{D}/2}|\Sigma|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{m})^\top \Sigma^{-1}(\mathbf{x} - \mathbf{m})\right), \tag{A.8}$$

where $\mathbf{m}$ is the mean vector and Σ is the symmetric, positive and definite covariance matrix.

Let $\mathbf{x}$ and $\mathbf{y}$ be jointly Gaussian random vectors,

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_x\\ \boldsymbol{\mu}_y\end{bmatrix}, \begin{bmatrix}A & C\\ C^\top & B\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_x\\ \boldsymbol{\mu}_y\end{bmatrix}, \begin{bmatrix}\tilde{A} & \tilde{C}\\ \tilde{C}^\top & \tilde{B}\end{bmatrix}^{-1}\right). \tag{A.9}$$

The marginal distribution of $\mathbf{x}$ and the conditional distribution of $\mathbf{x}$ given $\mathbf{y}$ are

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_x, A), \tag{A.10}$$

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_x + CB^{-1}(\mathbf{y} - \boldsymbol{\mu}_y), A - CB^{-1}C^\top) \tag{A.11a}$$

$$\sim \mathcal{N}(\boldsymbol{\mu}_x - \tilde{A}^{-1}\tilde{C}(\mathbf{y} - \boldsymbol{\mu}_y), \tilde{A}^{-1}). \tag{A.11b}$$

The product of two Gaussians gives another un-normalized Gaussian

$$\mathcal{N}(\mathbf{x}|\mathbf{a}, A)\mathcal{N}(\mathbf{x}|\mathbf{b}, B) = Z^{-1}\mathcal{N}(\mathbf{x}|\mathbf{c}, C), \tag{A.12}$$

where $\mathbf{c} = C(A^{-1}\mathbf{a} + B^{-1}\mathbf{b})$ and $C = (A^{-1} + B^{-1})^{-1}$. The resulting Gaussian has a precision (inverse variance) equal to the sum of the precisions and a mean equal to the convex sum of the means, weighted by the precisions. The normalizing constant also looks like a Gaussian (in $\mathbf{a}$ or $\mathbf{b}$)

$$Z^{-1} = (2\pi)^{-\mathrm{D}/2}|A + B|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{a} - \mathbf{b})^\top (A + B)^{-1}(\mathbf{a} - \mathbf{b})\right). \tag{A.13}$$

Appendix B

Feature generation

B.1 Feature space

Table B.1: Feature space. *Original features, $^{+}$Vectorized inter and in-session time accumulators, ♦Accompanied by boolean features for value existence.

Sequential ($\boldsymbol{X}_{\text{seq}}$)	Session ($\boldsymbol{X}_{\text{ses}}$)	Impressions ($\boldsymbol{X}_{\text{imp}}$)
• Ref. item ID*	• Subsession	• Imp. item ID*
• Action ID*	• Total substeps	• Position
• Step*	• Total time steps	• Price*
• Frequency	• Total session time	• Views^{+}
• Session time♦	• Imp. list length	• Clicks^{+}
• Time dif.♦	• City ID*	• Interactions^{+}
• Dwell time♦	• Platform*	• Dwell time^{+}
	• Device*	• Metadata ($\boldsymbol{X}_{\text{meta}}$)*
	• Filters ($\boldsymbol{X}_{\text{filt}}$)*	

B.2 Data visualization

95% confidence intervals were obtained over 1000 bootstraps in the following plots, and relative frequency distributions are presented with omitted axes.

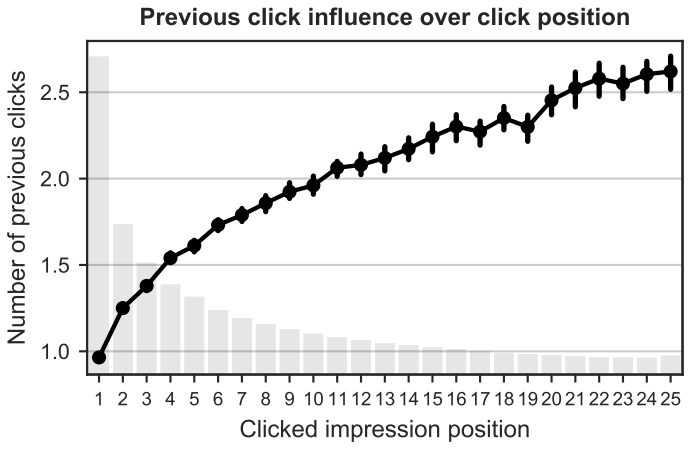


Figure B.1: Average number of previous clicks per clicked impression item position.

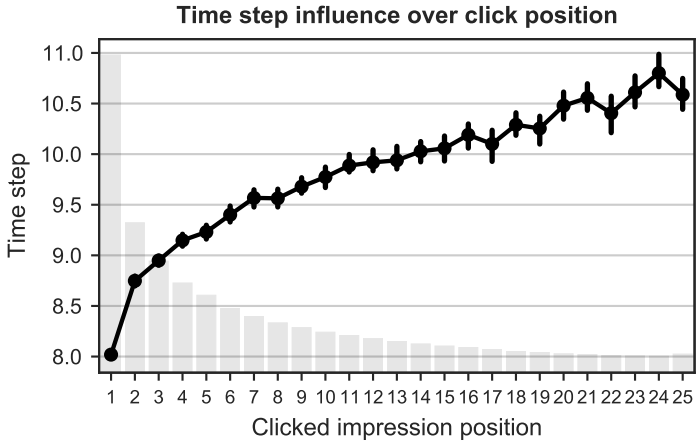


Figure B.2: Average click time step per impression item position.

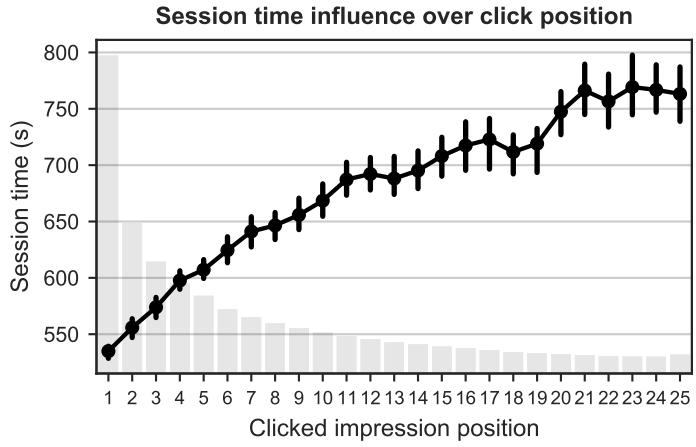


Figure B.3: Average session time per clicked impression item position.

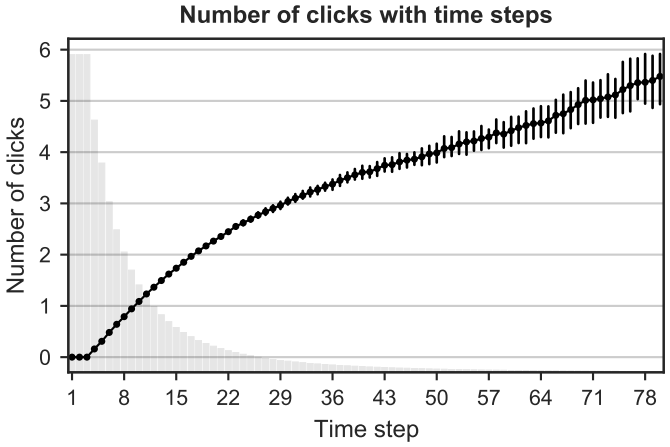


Figure B.4: Average number of clicks per time step.

Appendix C
Model Optimization

C.1 Hyperparameter codes

Table C.1: Top ten optimization model architectures. Only active hyperparameter codes (from Table 4.6) are displayed, besides the inputs which are all True for every architecture.

Model	H2	H3	H4	H5	H9	H12	H13	H14	H16	H17	H22	H23	H24	H25	H26	H31	H32
O1/M1	**2**	**Bi-S-GRU**	**250**	**0**	**3**	**6**	**Bi-S-GRU**	**218**	**Hierarch.**	**124**	**16**	**142**	**0.33**	**475**	**0.10**	**ReLU**	**0.0012**
O2	2	Bi-S-GRU	250	0	3	10	Bi-S-GRU	132	Hierarch.	86	16	124	0.31	338	0.10	ReLU	0.0018
O3	3	Bi-S-GRU	250	0.10	2	10	Bi-S-GRU	194	Additive	282	10	135	0.28	399	0.23	Leaky	0.0010
O4	2	Bi-S-GRU	250	0	3	6	Bi-S-GRU	231	Hierarch.	119	10	123	0.34	294	0.3	ReLU	0.0030
O5	2	Bi-S-GRU	250	0	3	6	Bi-S-GRU	201	Additive	277	5	132	0.22	279	0.10	Leaky	0.0011
O6	7	Bi-S-GRU	250	0.15	2	3	Bi-S-GRU	145	Scaled	77	3	140	0.19	422	0.17	PReLU	0.0030
O7	20	Bi-S-GRU	250	0	2	3	Bi-S-GRU	219	Hierarch.	211	5	114	0.30	324	0.14	ReLU	0.0014
O8	7	Bi-S-GRU	250	0	2	10	Bi-S-GRU	124	Dot	248	16	104	0.39	257	0.12	ReLU	0.0022
O9	2	Bi-S-GRU	250	0.10	2	10	Bi-S-GRU	194	Hierarch.	282	5	135	0.33	399	0.16	Leaky	0.0010
O10	2	Bi-S-GRU	250	0	3	2	Bi-S-GRU	203	Additive	102	30	182	0.25	415	0.12	ReLU	0.0014

C.2 Hyperparameter optimization results

Display note The best overall configuration (O1) is represented by a yellow marker in continuous distributions.

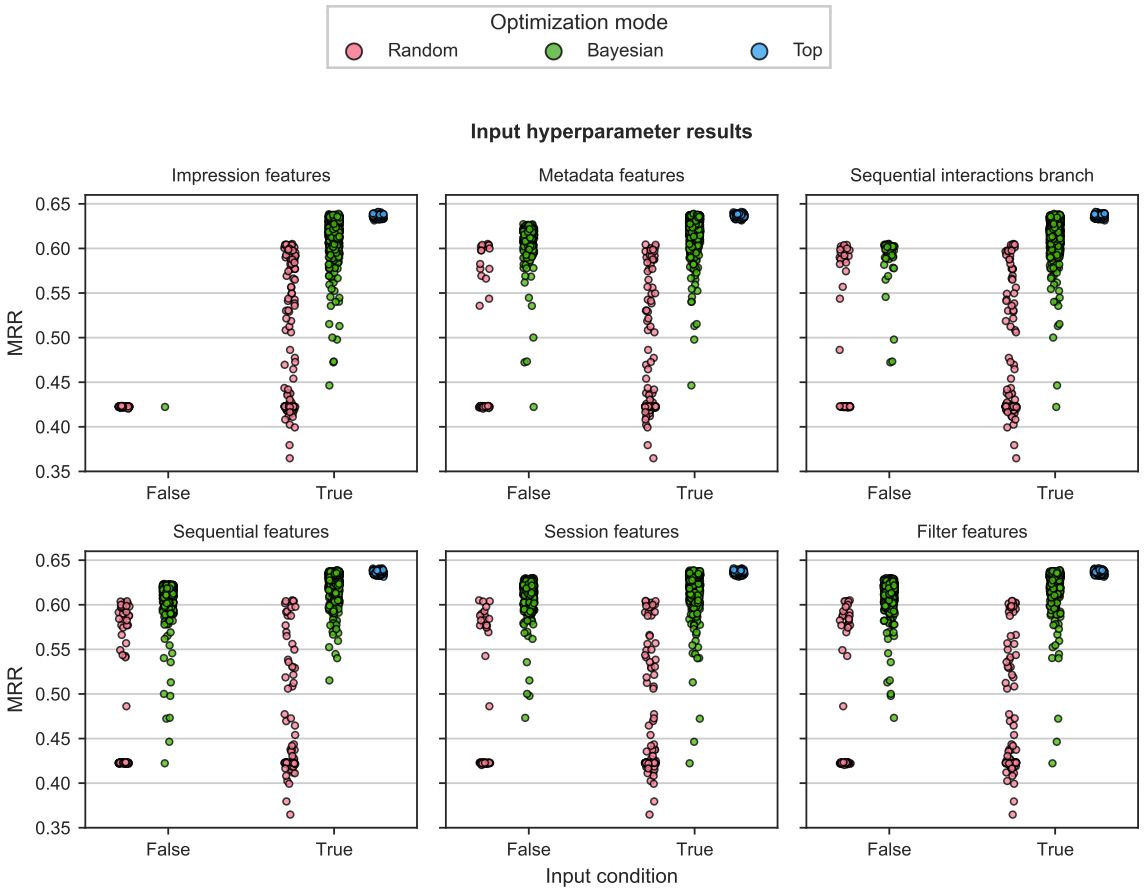


Figure C.1: Input optimization ranking performance distribution.

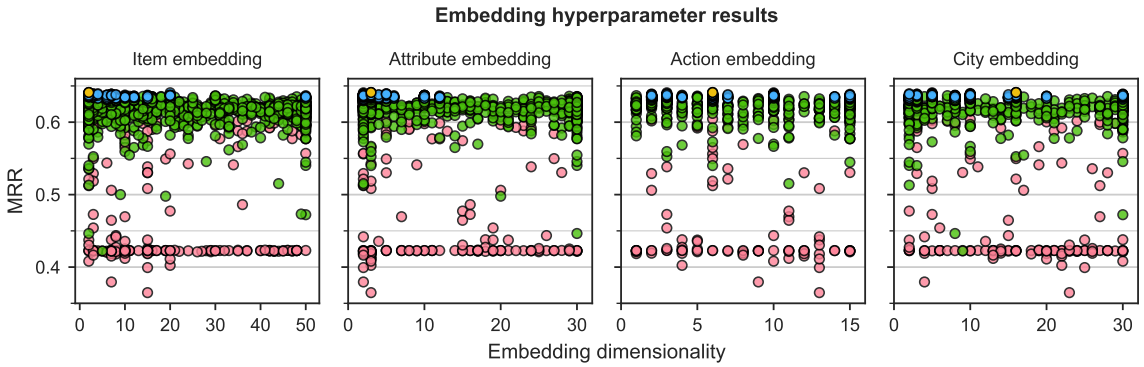


Figure C.2: Embedding dimensionality optimization ranking performance distribution.

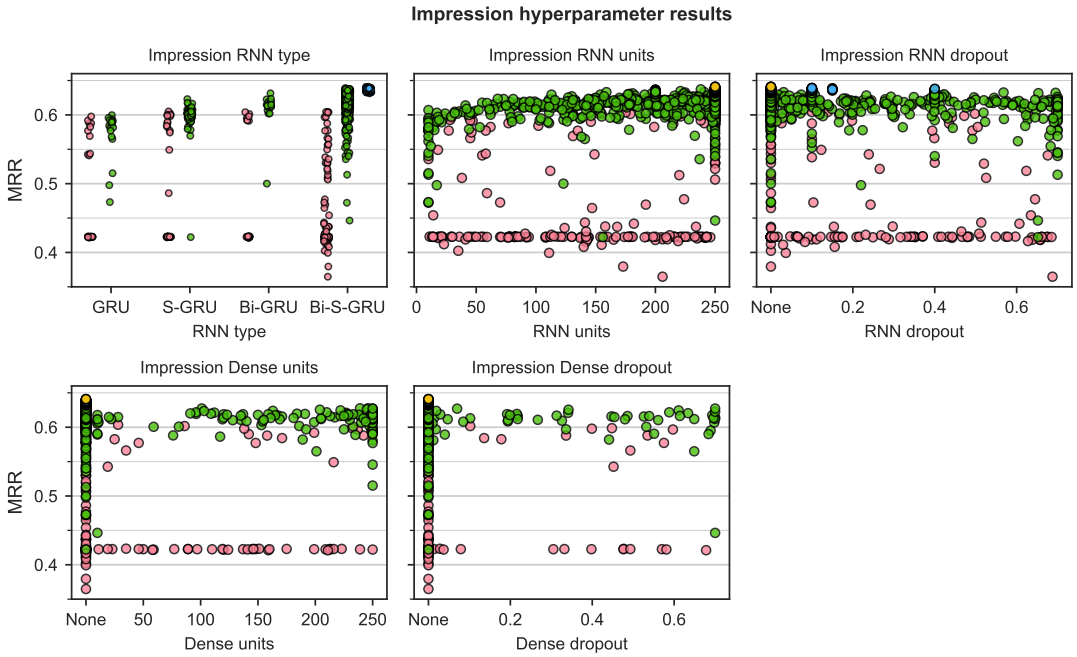


Figure C.3: Impression features branch optimization ranking performance distribution.

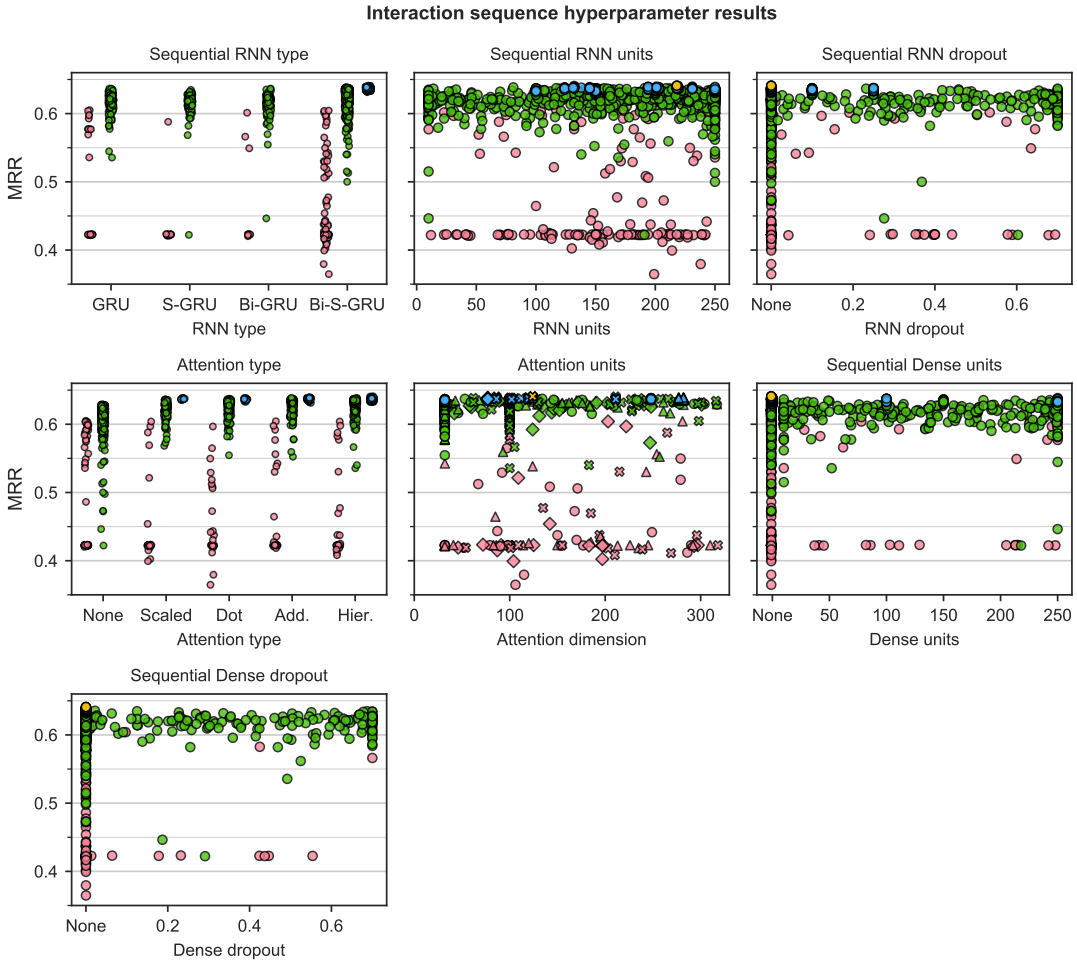


Figure C.4: Sequential interaction features branch optimization optimization ranking performance distribution. In the attention dimension plot, cross marks represent the Hierarchical, triangles the Additive, circles the Dot and diamonds the Scaled Dot types.

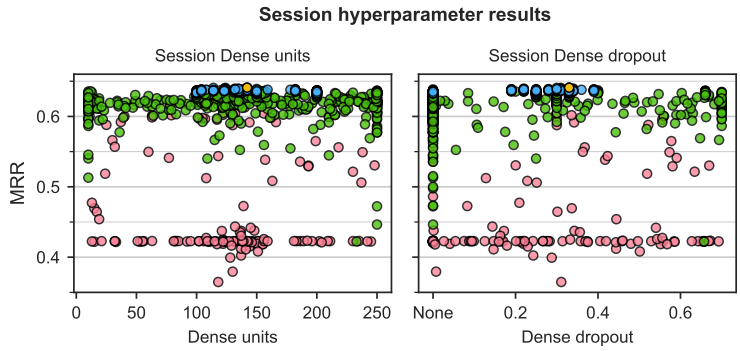


Figure C.5: Session features branch optimization optimization ranking performance distribution.

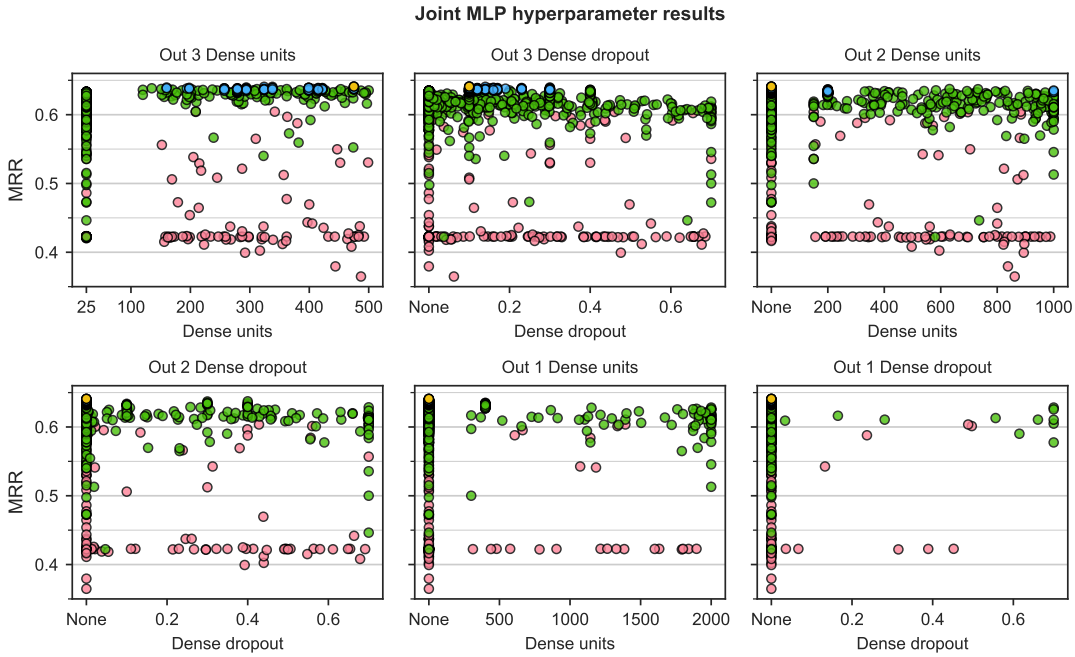


Figure C.6: Joint MLP structure optimization ranking performance distribution.

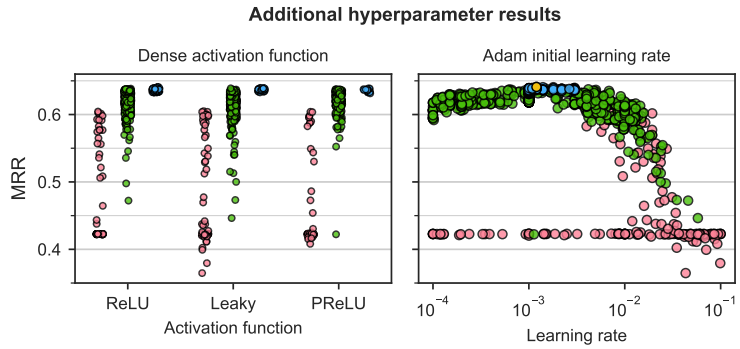


Figure C.7: Activation function and initial learning rate optimization ranking performance distributions.

Appendix D

Final Model Results

D.1 Training curves

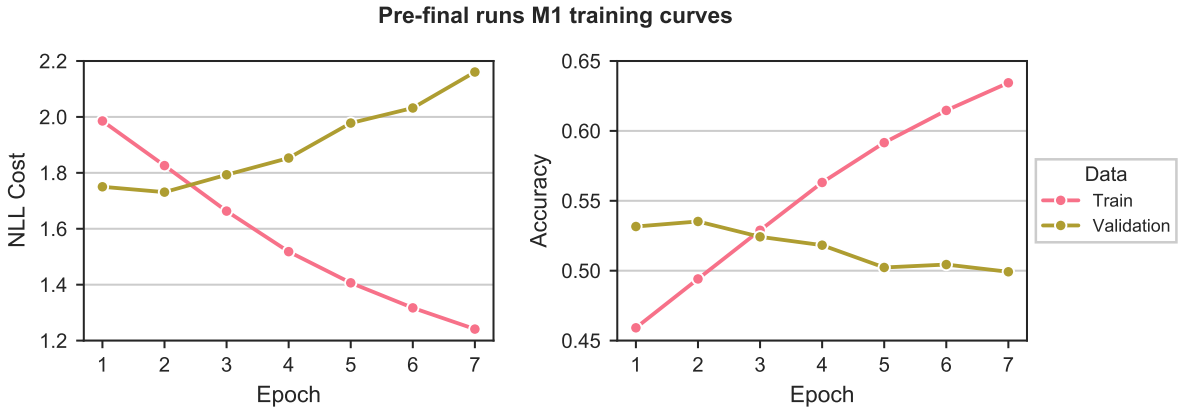


Figure D.1: Pre-training M1 validation and accuracy curves for the base architecture.

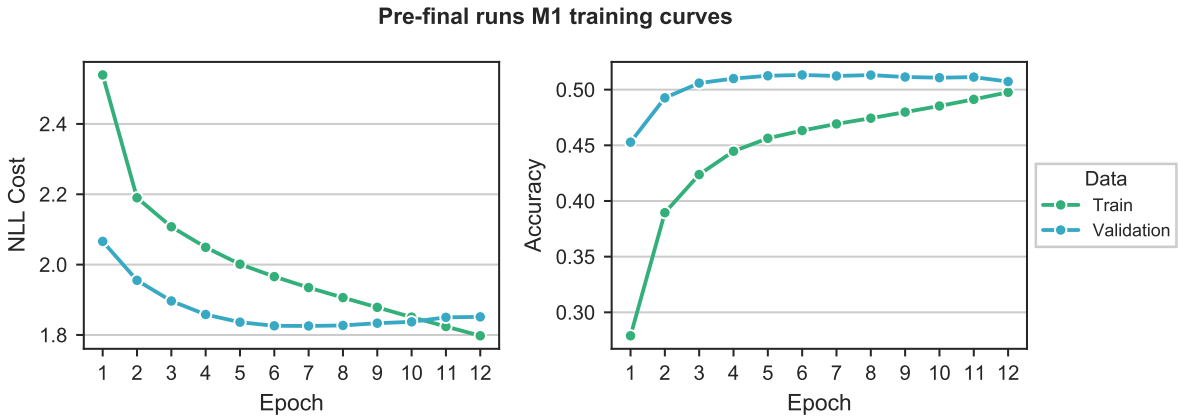


Figure D.2: Pre-training modified M1 validation and accuracy curves with 0.5 global dropout and 0.00005 initial learning rate.

D.2 Confusion matrix

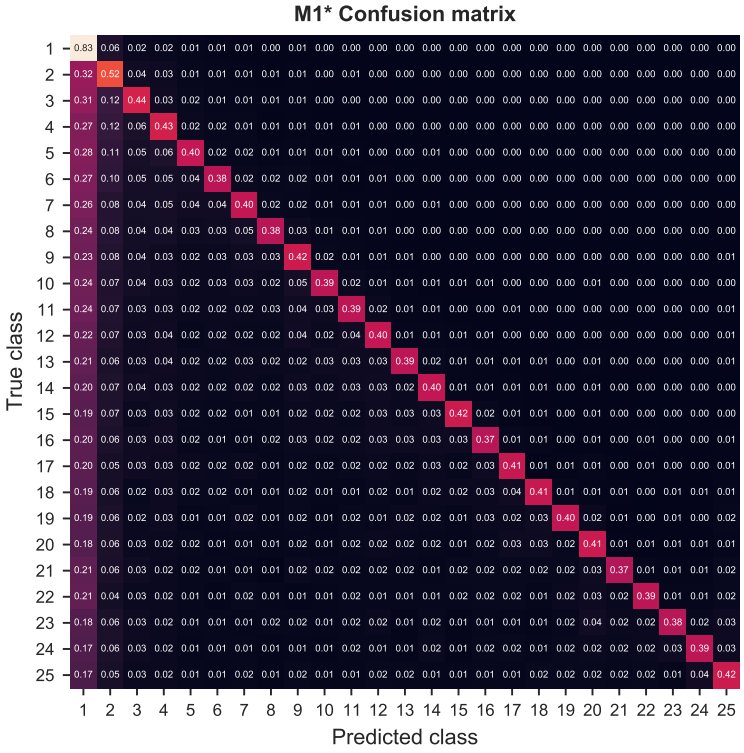


Figure D.3: Final model M1*'s confusion matrix, normalized over the true conditions (rows).

D.3 Specific examples

D.3.1 Session 0029af4c7ac6d

Seq. act.	16	8 2	1 2	1 2	3	2	2	2	2	2	2	2	2	2
Seq. itm.	16	1 105934	1 178153	1 105020	1	111289	390366	140591	239924	187971	200687	178428	194374	175199
Imp. itm.	25	111289 105020	390366 52384	572635 172001	140591 390081	239924 276642	187971 178290	200687 178120	178428 111272	194374 291251	175199 219713	105934 217983	178153 260626	239886
Pred.	y	15	$\hat{y}_{\max}$	15	Conf.	0.6288								

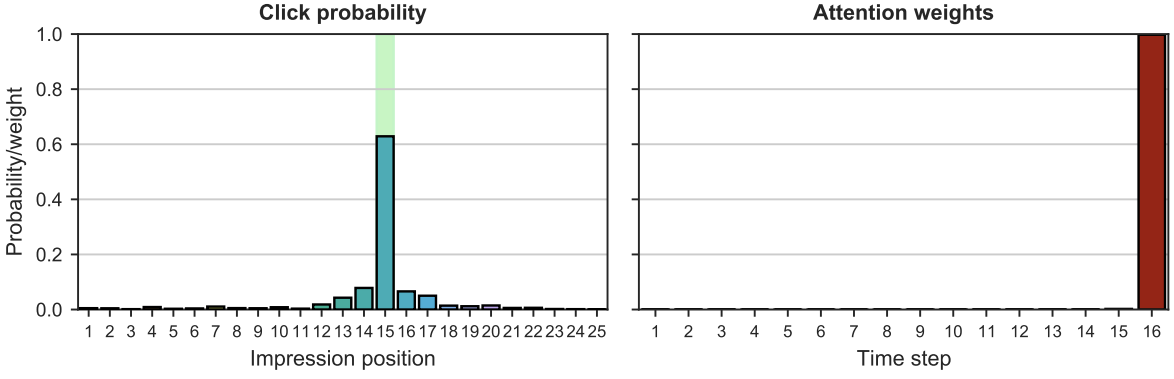


Figure D.4: Click probabilities and attention weights for session sample 0029af4c7ac6d.

D.3.2 Session 0034e1fff6628

Seq. act.	11	1	1	3	1	1	3	5	7	6	4	5			
Seq. itm.	11	1	1	1	1	1	1	468450	468450	468450	468450	468450			
Imp. itm.	25	212737	220819	482449	348378	652338	468450	119821	467364	113368	236390	298812	689796	119837	
		175234	211926	391358	140173	284254	249161	143167	479549	313192	205153	203496	211922		
Pred.	y	6	$\hat{y}_{\max}$	6	Conf.	0.7199									

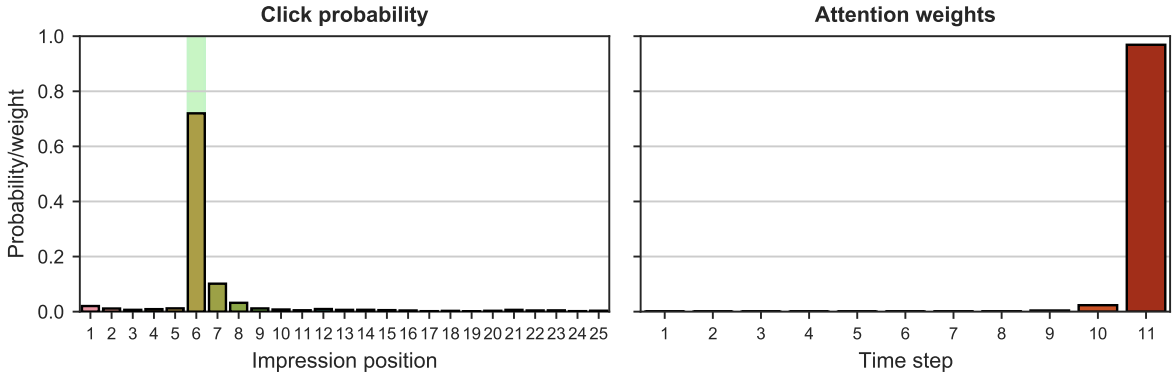


Figure D.5: Click probabilities and attention weights for session sample 0034e1fff6628.

D.3.3 Session 0043391e99388

Seq. act.	5	5	5	5	5	7									
Seq. itm.	5	310012	182932	310012	182932	182932									
Imp. itm.	21	310012	182932	300554	323430	370445	266902	309823	204104	283656	167294	493753	713452	167345	
		685402	560973	544210	405548	487539	460180	565518	413072						
Pred.	y	2	$\hat{y}_{\max}$	2	Conf.	0.4223									

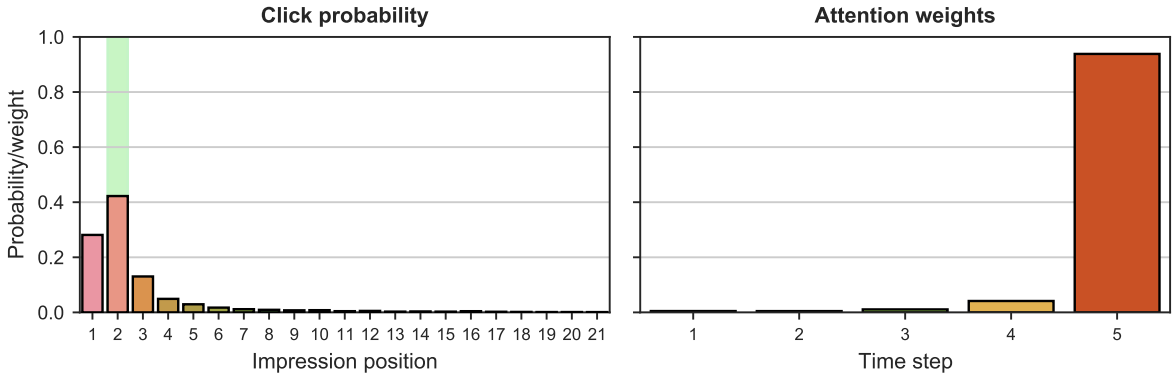


Figure D.6: Click probabilities and attention weights for session sample 0043391e99388.

D.3.4 Session 0048f148890c5

Seq. act.	8	8	8	5	5	2	5	2	8						
Seq. itm.	8	1	1	513432	125659	154236	154236	304914	1						
Imp. itm.	25	183095	46477	51280	51286	51295	51296	57500	57537	61703	69172	88331	88408	88707	
		88915	93722	104938	107541	117315	117316	117317	117318	117319	117321	117322	117328		
Pred.	y	2	$\hat{y}_{\max}$	1	Conf.	0.1546	y Conf.	0.1195							

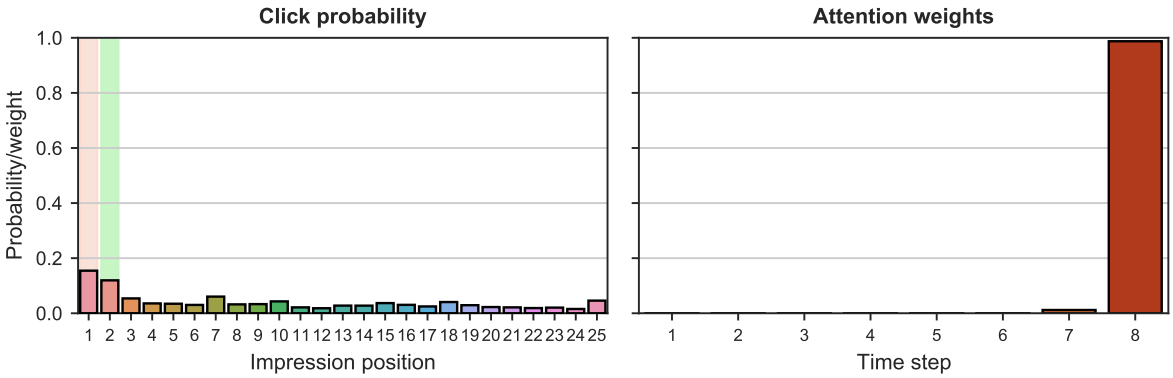


Figure D.7: Click probabilities and attention weights for session sample 0048f148890c5.

D.3.5 Session 1ef61eeb4ccaf

Seq. act.	4	8	5	9	5									
Seq. itm.	4	1	4122	4587	4122									
Imp. itm.	25	4587 209494	4122 4156	20548 6905	288477 19797	20481 6927	48741 19929	101298 178418	132888 146619	472999 19323	19715 19263	19885 6876	19863 20601	5660
Pred.	y	1	$\hat{y}_{\text{max}}$	1	Conf.	0.8820								

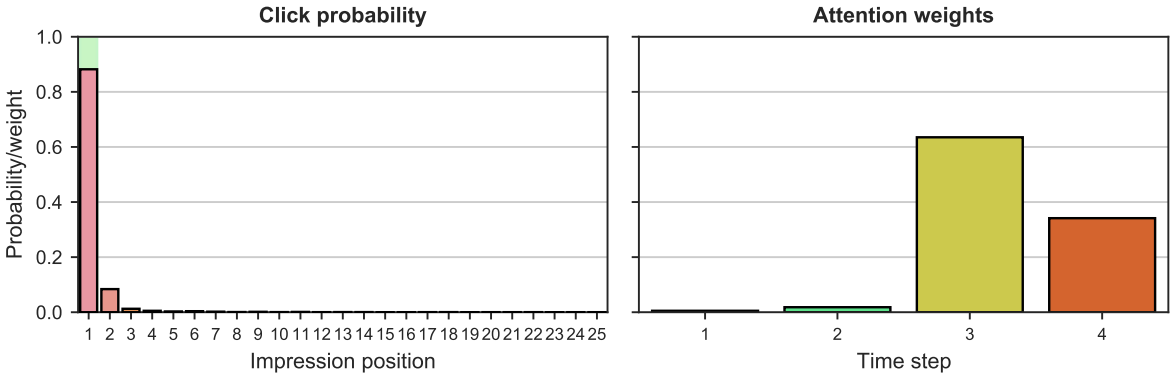


Figure D.8: Click probabilities and attention weights for session sample 1ef61eeb4ccaf.

D.3.6 Session 553764ffd8cd1

Seq. act.	4	7	7	7	7									
Seq. itm.	4	79040	35736	102245	79040									
Imp. itm.	15	31992 132535	32283 195255	146997	127375	144417	25799	83293	102245	41524	120643	35736	79040	80124
Pred.	y	1	$\hat{y}_{\text{max}}$	12	Conf.	0.5724	y Conf.	0.0713						

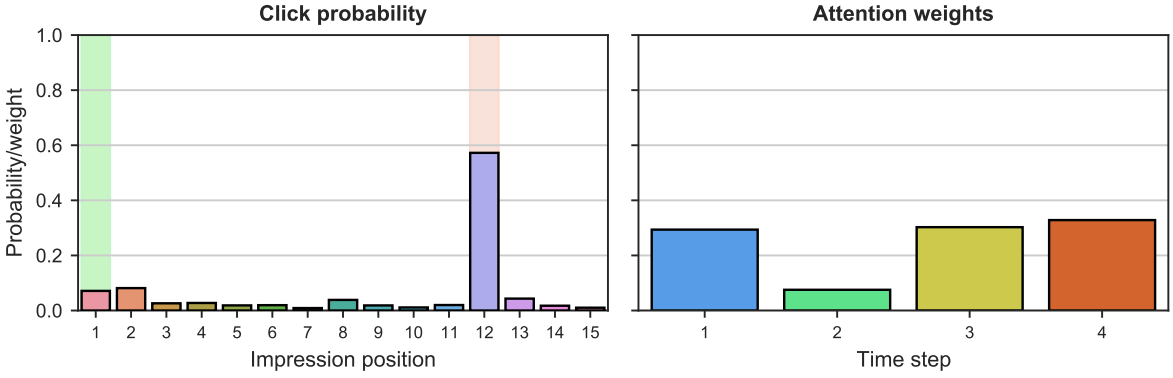


Figure D.9: Click probabilities and attention weights for session sample 553764ffd8cd1.

D.3.7 Session 66aa59653d4e6

Seq. act.	5	5	5	2	9	7								
Seq. itm.	5	53594	78000	78000	537906	16632								
Imp. itm.	25	537906 51174	16632 455033	16642 15011	236969 54155	16618 53033	16633 44298	53527 78000	227312 151258	91898 53599	292444 108348	391715 410465	150637 16542	312449
Pred.	y	1	$\hat{y}_{\max}$	1	Conf.	0.5686								

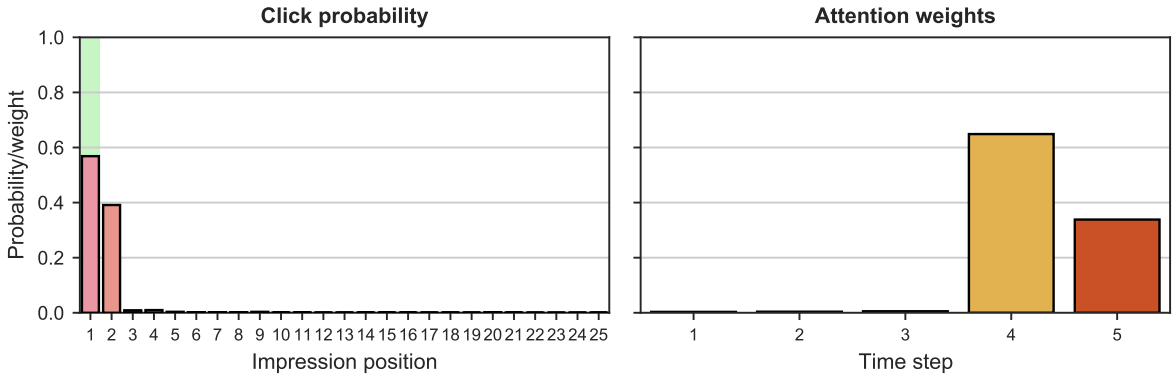


Figure D.10: Click probabilities and attention weights for session sample 66aa59653d4e6.

D.3.8 Session 73b9d04887b5f

Seq. act.	25	5 3	7 5	6 7	5 4	6 5	7 7	5 6	2 5	5 7	7 6	6 7	1 4	1
Seq. itm.	25	196154 1	196154 567334	196154 567334	235192 567334	235192 540000	235192 540000	306697 540000	306697 48605	158559 48605	158559 469336	158559 469336	1 469336	1
Imp. itm.	25	567334 71602	42151 469336	55514 58116	124905 3126	195321 61093	3127 269110	46481 410744	540000 55900	344879 131578	48605 376500	178219 55515	161710 62668	55513
Pred.	y	15	$\hat{y}_{\max}$	15	Conf.	0.3683								

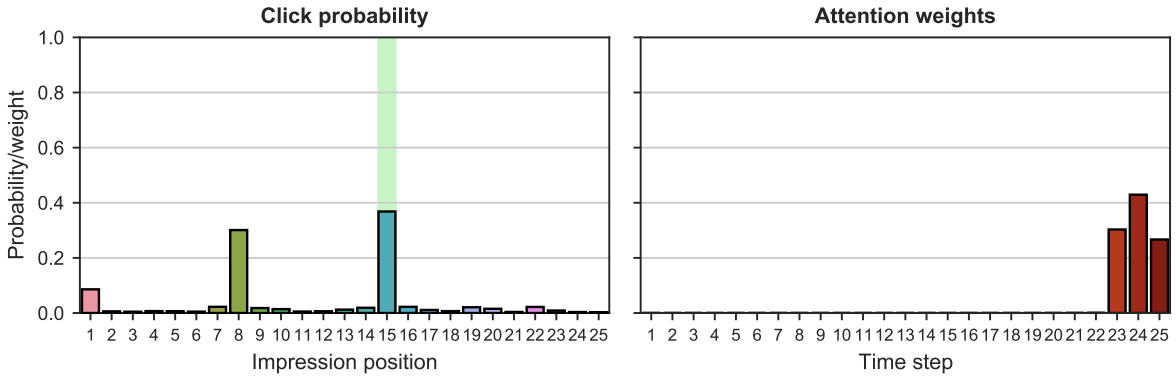


Figure D.11: Click probabilities and attention weights for session sample 73b9d04887b5f.

D.3.9 Session e229614dd7ea2

Seq. act.	4	7	7	7	7									
Seq. itm.	4	168276	276667	168276	260341									
Imp. itm.	13	276667	294614	168276	260341	194607	238934	123142	378663	191562	158406	590863	123144	637332
Pred.	y	1	$\hat{y}_{\max}$	4	Conf.	0.4962	y Conf.	0.1325						

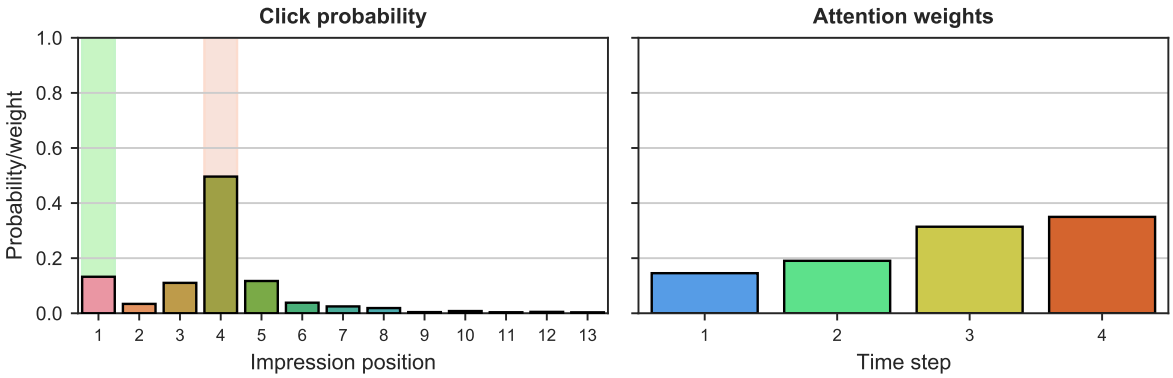


Figure D.12: Click probabilities and attention weights for session sample e229614dd7ea2.

Example display notes Each session example contains a table with the sequential interactions and corresponding item references leading up to the respective click event in the first two rows. The third row corresponds to the presented impression list. The final prediction row contains the label y, the position index of the predicted clicked item $\hat{y}_{\text{max}}$, its click probability (Conf.) and the click probability attributed to the label in case of a wrong prediction (y Conf.).

Item vocabulary codes are displayed instead of the original IDs. The action vocabulary codes are presented in Table D.1 below.

Table D.1: Example action vocabulary codes.

Code	Action
1	Change of sort order
2	Clickout
3	Filter selection
4	Interaction item deals
5	Interaction item image
6	Interaction item info
7	Interaction item rating
8	Search for destination
9	Search for item
10	Search for POI