UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Maksym Semikin

Jointly Tackling User and Item Cold-start with Sequential Content-based Recommendations

Master's Thesis (30 ECTS)

Supervisor: Tambet Matiisen, MSc
Supervisor: Carlos Bentes, MSc

Tartu 2019

Jointly Tackling User and Item Cold-start with Sequential Content-based Recommendations

Abstract:
Deep learning has been successfully used in the context of recommender systems. Sequential recommender systems are a class of algorithms which model user-item interactions and their temporal relationship in order to generate relevant personalized recommendations. Recurrent neural networks have become the state-of-the-art approach for sequential modeling, but current approaches in the context of recommendation systems are tightly coupled with the catalog size and item identifiers. This imposes a problem when new items are to be incorporated into the list of recommendable products, the entire model needs to be retrained. Feature-rich item metadata has been successfully used to improve recommendation quality with both sequential and non-sequential recommenders. However, to the best of our knowledge, no attempt has been made to tackle the problem of newly encountered user and item in a sequence aware model with personalized recommendations. This work presents a novel architecture for context-aware item prediction based on embeddings. The model combines item embeddings within a sequence to dynamically predict an item embedding for the next interaction. This allows to incorporate new items without model retraining. Moreover, the proposed architecture implicitly models the user preferences from user-item interactions and is able to provide item embedding predictions that are personalized to the context of a user and therefore produce personalized recommendations. The results are compared with GRU4Rec and TransRec in the next interaction prediction task using the Amazon reviews public dataset, and our experiments show comparable or better results than state-of-the-art personalized models, with the added benefit of being able to add items or users without model retraining.

Keywords:
Recurrent neural networks; Recommender systems; Cold-start; Sequential recommendations; Personalized ranking; Content-based recommendations.

CERCS: P170 Computer science, numerical analysis, systems, control

Lahendus uute kasutajate ja toodete lisamiseks sessioonipõhistes soovitusüsteemides

Lühikokkuvõte:
Sügavaid närvivõrke on edukalt kasutatud mitmetes soovitussüsteemides. Sessioonipõhised soovitussüsteemid on nende üks alaliik, milles modelleeritakse kasutajate ja toodete interaktsioone (klikke), selleks et genereerida kasutajale isikupäraseid soovitusi. Rekurrentsed närvivõrgud on viimastel aastatel muutunud eelistatuimaks lahenduseks mitmesuguste jadaandmete modelleerimisel, sh kasutajasessioonid, kuid olemasolevate lahenduste puuduseks on see, et need on jäigalt seotud tootekataloogi ja selles olevate toodetega. Uute toodete lisandumisel tuleb kogu mudel uuesti treenida. Üks võimalik lahendus sellele on toodete metainfo (pealkiri, kirjeldus, pilt) kasutuselevõtt, mis võimaldab tooteid identifitseerida nende sisu põhjal, mitte identifikaatori järgi. Samas teadaolevalt ei ole hetkel välja pakutud meetodit, mis lahendaks korraga nii uue toote kui ka uue kasutaja lisandumise probleemi sessioonipõhistes soovitussüsteemides.

Töös pakutakse välja uudne arhitektuur sessioonipõhise soovitussüsteemi jaoks, mis kasutab toodete metainfol põhinevaid vektoresitusi. Mudelis kombineeritakse sessiooni jooksul külastatud toodete vektoresitused, selleks et ennustada järgmise toote vektoresitust. Selline lahendus võimaldab lisada tootekataloogi uusi tooteid ilma mudelit uuesti treenimata. Täiendavalt kasutatakse kasutaja sessiooni tema eelistuste modelleerimiseks, mis tähendab, et ennustatud järgmine toode sõltub kasutaja varasematest interaktsioonidest ja seega on tegemist isikupärase ennustusega. Eksperimendid viidi läbi Amazoni kasutajaarvustuste andmestiku peal ning tulemusi võrreldi GRU4Rec ja TransRec mudelitega. Pakutud lahendus saavutas võrreldavaid või paremaid tulemusi kui varasemad parimad mudelid ning võimaldab seejuures lihtsustada uute toodete või kasutajate lisamist.

Võtmesõnad:
Rekurrentne närvivõrk, soovitussüsteemid, külmkäivitus, sessioonipõhised soovitused, isikupärastatud järjestus, sisupõhised soovitused.

CERCS: P170 Arvutiteadus, arvanalüüs, süsteemid, kontroll

Contents

1 Introduction

E-commerce recommender systems suffer from both item cold-start and user cold-start problems. We define complete cold-start (CCS) to be the case when no prior interactions are available for the user or item and cold-start (CS) when there are just a handful of those interactions such as first couple of clicks in the session.

User CCS and CS occur when a user interacts with a system for the first time. Many e-commerce customers use web shops without being logged in and therefore only session level information can be linked together. Each time an anonymous user visits the website, the system has no record of their previous actions. The large total proportion of anonymous users makes it crucial for the recommendation engine to handle them appropriately. A simple solution to the user CS and CCS problems is to make the model non-personalized. However, personalized customer experience is an indispensable part of modern online platforms and helps with long-term business goals as well as keeps the users more engaged. While no personalization is possible for user CCS, some level of personalization is possible with just a few initial interactions. Modelling sequential dependencies between items as well as co-purchase information has become a successful solution for cold-start users by providing high-quality complementary suggestions.

Item CS and CCS similarly refer to the situation when a new item is added to the catalog. Webshop owners are especially interested in recommending latest additions to the catalog given that those products are often most relevant to customers, even though there is still not enough interactions on those products making many recommendation approaches incapable of incorporating those items. Both item CS and CCS problems can be alleviated by hybrid approaches which extract item profiles from item content features based on user interactions with content-wise similar products.

Considering the above, we argue that handling new users, handling new catalog additions and providing personalized experience where possible are all necessary parts of an e-commerce recommendation engine and neither of them can be neglected.

A considerable number of hybrid and sequential factorization methods were proposed to deal with item cold-start and user cold-start. However, factorization methods have been recently surpassed in accuracy by newer deep learning architectures. Combatting user and item cold-start jointly has been shown to be non-trivial in recent deep learning experiments, which showed that models significantly rely on item identifiers to achieve good performance [HQKT16]. Therefore, achieving comparable results without relying on item identifiers is possible only via especially effective use of content features. While rich content information is often available for items, learning to extract meaningful high-level representations out of those features requires either time-consuming feature engineering or substantial training time with deep learning models. We tackle this problem by jointly using large scale pretrained language models and image recognition models. We represent items with embeddings learned from features extracted from the pretrained models. We acquire user preferences from their sequence of interactions using a recurrent neural network. With that we let go of explicit user and item identifiers completely to enable continuous learning on upcoming events and items. The network does not need to be retrained from scratch every time the catalog changes which simplifies the deployment in real-world setting. In addition, the model is able to provide personalized recommendations by learning from the first few user interactions and represent the user in the recurrent hidden state. Full implementation is publicly available[1].

We show that the proposed method performs on par with state-of-the-art sequential methods on a large public e-commerce dataset while having the additional benefit of being able to easily introduce new users and new items to the already trained system. The network is supervised with implicit feedback.

List of contributions:

- We propose EmbRec, a simultaneous approach for item and user side cold-start problem while preserving personalization for non-cold users.
- We train a model without explicit user and item identifiers, enabling online incorporation of new users and new items without reinitialization.
- We employ transfer learning from large-scale pretrained language models and image recognition models enabling the algorithm to work on smaller recommendation datasets.

The work is divided into 4 parts:

- Chapter 2 defines recommendation systems and provides motivations for their use, outlines the high-level landscape of popular recommendation methods and elaborates on the literature most related to this work.
- Chapter 3 defines the notation used to describe the model structure and optimization, followed by the description of the proposed method, evaluation methodology and further details about feature extraction and training process.
- Chapter 4 presents numerical evaluation as well as analyzes how the model performs qualitatively, followed by detailed discussion of architectural considerations encountered in experimentation.
- Chapter 5 concludes the work and provides a summary of the results as well as outlines possibilities for future work.

[1] `https://github.com/msemikin/embrec`

2 Background

2.1 Recommender systems

In the last couple of decades users have started to face the problem of overwhelming choice ever more often. Being presented with a wider variety of options has been considered to be positive for both the user and the business. However, sudden growth of catalog sizes made it infeasible for the user to adequately interact with the catalog, which reduces satisfaction and might prevent them from fulfilling the goals within the system, also known as overchoice problem [Sch04]. Recommender systems are decision support systems designed to filter the catalog for the user in order to reduce the cognitive load. Thus, the main task of a recommendation system is to generate a list of most relevant suggestions to the user given all of the information available about items and users in the system. Adoption of recommendation systems has been rapidly growing in the past decades as goods and service providers began to address the overchoice problem.

In the era of internet businesses, vast amounts of data about user interactions are being continuously generated and recorded in databases. Businesses willing to make use of this data have turned to recommender systems to achieve multiple goals [RRS11]:

- Increase revenue. Usually achieved by increasing sales from the tail of the catalog distribution. Recommender systems try to match products to users which increases the probability that rare items will be bought.

- Improve customer experience. Recommender systems enable efficient exploration of the catalog and help to fulfill further the user needs.

- Improve customer fidelity. Personalized and relevant experience with the platform is an important value proposition to compel users to continue using the system above competitors.

- Better understand the user needs. Recommendation systems model user preferences and needs in the underlying context which may itself be used beyond the recommendation task, for instance to separate the audience into segments and target them with relevant campaigns.

Recommender systems are widely used in industries such as e-commerce, online advertising, music and video streaming platforms, online media and travel. This work focuses on the domain of e-commerce but the principles discussed are applicable across many different domains.

Recommendation systems are also highly valuable to customers. Previous research has established that users have different intents when using suggestions from a recommender systems and can derive value in different scenarios. Herlocker et al. [HKTR04] highlights some of them:

- Find some good items. This is the core problem of the domain. In many domains users can give up finding all potential good options as long as they can find at least some of them given the inherently large catalog.
- Find all good items. In certain domains finding all positives is necessary and users can tolerate a bigger number of false positives.
- Find a sequence of items. In some domains users are looking for a coherent sequence of items, such as music playlists and the main judging criteria is how well the individual items play together as an ordered set.
- Find a bundle of items. In the case of e-commerce, people often buy multiple items that naturally complement each other. This may include whole clothes outfits or hardware components that are compatible.
- Browse the catalog. Exploration of the catalog without purchasing any items. In this context users might be looking for interesting or diverse products without any intention to buy any of them.
- Find a trustworthy suggestion engine. Users have reported that they trial recommender system platforms to see if they can figure out their personal taste and thus be a credible source of suggestions.

2.2 Types of user feedback

The task of recommender systems from the supervised learning point of view can be expressed in multiple ways depending on the kind of interactions data available. The primary information available for a recommendation system is an often very sparse user-item interaction matrix (see Figure 1). User feedback about the items can be explicit such as a text review about the product or a rating in some standard scale. In this case, the learning task can be formulated as rating prediction for user-item pairs. This learning task was popularized by the Netflix challenge [BL$^+$07]. More often, however, only implicit feedback is available in form of clicks, purchases and searches. Because of the binary nature of implicit feedback, we assume that the interaction with an item means a signal of preference. The numeric value of implicit interactions such as the number of interactions of a user with a specific item defines our confidence in the preference [HKV08]. While we cannot define explicit rating values for unobserved user-item pairs, we can set the implicit feedback numerical value for unobserved user-item pairs to some base value. We cannot make strong assumptions about the items the user has not interacted with, because unobserved user-item pairs might likely mean that the user does not know about the existence of the item as opposed to actively choosing against it. So unlike explicit feedback, true negative samples are not available for implicit data. In addition,

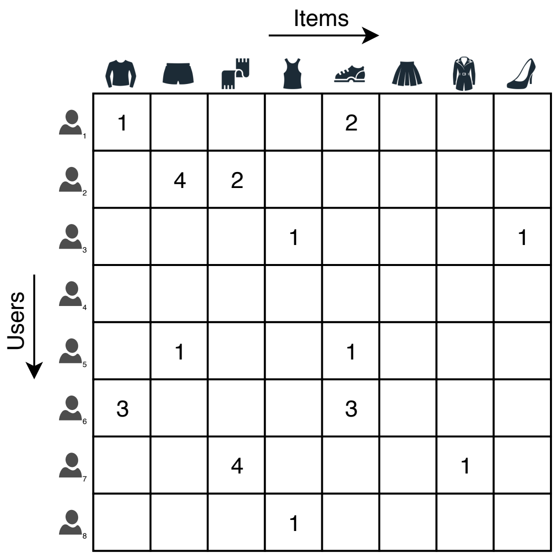


Figure 1. User-item interaction matrix. Numbers in the cells can mean explicit rating values or implicit preference counts such as number of purchases.

implicit feedback is inherently noisier than explicit feedback and is not always a signal of preference. Nevertheless, implicit data was successfully used for the recommendation task both alone and in conjunction with explicit ratings.

2.3 Types of recommender systems

The landscape of recommendation systems can be divided into several high-level groups.

Collaborative filtering (CF) This class of systems uses preferences expressed by other users to derive relevant items for the given user. The systems of this class are divided into model-free and model-based methods. Model-free systems such as user-based CF and item-based CF [SKKR01] use heuristic computations on the user-item interactions matrix to combine other people's ratings on different items to get high-scoring suggestion items for the current user. Model-based systems try to model the interactions matrix by supervised learning approaches. Latent factor models such as matrix factorization are one of the best-performing model-based methods. They usually try to reconstruct the interactions matrix with a low-rank approximation [KBV09].

Content-based filtering Side information such as item metadata can be incorporated into the recommender system to solve issues of the system encountering new items or new users. Pure content-based filtering approaches derive the most important item features for the user and learn to predict user preference for the items the user has not yet interacted with. Thus the system estimates a model for each user separately and does not use other users' interactions for modelling current user preferences [PB07].

Knowledge-based recommenders In this class of recommenders, the user defines their information need in a structured way and then guides the system with explicit feedback to further clarifying questions from the recommender. These methods are tightly related to knowledge bases. [Agg16]

Context-aware recommenders Such methods make use of context information such as previously searched or clicked items in the current session of the user, the sequence of items in recent history, current location of the user. [AT11]

Hybrid recommenders Combinations of two or more of the above approaches often yield a more accurate model, thus there is a whole class of models which incorporate both other users' interactions and content features. [Bur02]

It is noteworthy, that most high-performing recommendation methods usually combine aspects of multiple classes. For example, context-aware recommenders are usually built on top of collaborative filtering methods.

2.4 Related work

2.4.1 Sequential recommendations with latent factor methods

Sequential recommenders are a special case of context-aware recommenders. They take into consideration the relative order of user interactions and not necessarily the explicit timestamps of those events. Early work on sequential modelling of user preferences has been explored with association rules [ABAH14], MDPs [MR07] and Markov Chain models [ZCM01]. More recently, matrix and tensor factorization methods have been applied to model sequential and temporal dynamics. Factorization machines [Ren12], of which matrix factorization is a strict subset, provide a general framework for hybrid and context-aware recommendations and allow to incorporate any available side features including item content, user content, location data and sequential information as a single factorization task. Rendle et al. [RFST10] proposed a personalized factorization method for the first-order Markov Chain transition matrix, Factorized Personalized Markov Chain (FPMC), which suffers from user cold-start. Fossil [HM16] improved upon FPMC and combined higher-order Markov Chain factorization with factorization of the item similarity matrix introduced previously in FISM model [KNK13]. The method achieves personalization for non-cold users via learned neighbourhood of history items with a gradual shift to factorized Markov chain to handle cold users. While Fossil handles sparse datasets and incorporates sequential information to deal with user cold-start, the authors did not experiment with purely content embeddings to deal with new items.

2.4.2 Metric embedding methods

Traditionally, latent factor models have been optimized with non-metric distance measures such as cosine similarity or inner product. Recently, metric space embedding methods have been shown to significantly improve over non-metric factorizations by better generalization due to metric conditions such as the triangle inequality. In Latent Mahalanobis Transform (LMT) [MTSvdH15] the authors proposed to incorporate image features and co-purchases data to learn a low-rank Mahalanobis transformation of items into a space where compatible items are close. Monomer [HPM16] extended LMT to learn not only compatibility between single items but also compatibility of heterogeneous item bundles. Both works deal with item and user cold start but exclude the sequential signal and are non-personalized. In TransRec [HKM17], the authors proposed to learn users as vector translations applied to item vectors in a metric space. TransRec models user history as a series of vector translations. Experiments showed that TransRec outperforms all factorization and metric embedding baselines. In the same work, TransRec was also extended to an item-based method which deals with item and user cold-start by learning global translation vectors over item vectors which are produced from content features. While the work addresses both user and item cold-start problems as well as sequential modelling and personalization, authors do not provide a unified framework which combines all the benefits.

2.4.3 Recurrent neural networks for recommendations

A lot of work has been done on adapting deep learning methods to the recommendation task, which has been recently surveyed [ZYST19]. Sequential models with recurrent neural networks is the closest set of methods to our work. Hidasi et al. [HKBT15] first explored the problem of session-based recommendations with constant user cold-start via a non-personalized GRU4Rec model, later improved in follow-up works [TXL16], [HK18]. Hierarchical recurrent neural networks [QKHC17] provided additional personalization to GRU4Rec recommendations obtained from session dynamics when users have more than one session. Another work by the same authors [HQKT16] used images and text about items to improve session predictions but observed that the network which used only item identifiers was significantly outperforming pure content-based network. Beutel et al. [BCJ+18] used recurrent neural architecture for session next-item predictions by learning to predict items from a fixed set available during training time, following the underlying assumption that the network is retrained from scratch regularly. Recurrent recommender networks [WAB+17] used recurrent neural networks to model both sequential and temporal dynamics of both users and items. The work did not include experimentation on purely content-based item representation. In general, recurrent neural networks have proved to be a strong method to model sequential and temporal dynamics in recommendation context but did not succeed with pure content-based recommendations.

2.4.4 Hybrid models for item cold-start

Item cold-start problem has been effectively addressed by pure content-based models. Early content-based methods derived a user profile from features of the content consumed previously, e.g. movie genres or clothing categories. Previously unseen items can then be recommended to users with existing history based on content similarity. Such pure content models require estimating a model for each user, and exclusion of sequential and collaborative information makes them less expressive. As stated before, latent factor methods have become a natural choice to model the sparse nature of user-item interaction matrices with methods such as matrix factorization and have been successfully used in dealing with item cold-start. In timeSVD++ [Kor10], matrix factorization rating prediction for the Netflix challenge was appended with modelling user and item rating drifts with time. What makes the work more relevant to us is that IRCD-CCS [WHC^{+}17] extended timeSVD++ to handle cold items by finding content-wise most similar non-cold items and aggregating their estimated ratings. While the temporal rating drifts are handled in the model, sequential information is not incorporated and the model suffers from user cold-start. To address textual content features of items, Ask the GRU [BBM16] proposed to learn item embeddings from text by multitask training of a GRU encoder on item text metadata from scratch. The obtained embeddings are then used in personalized matrix factorization framework. Training text encoders requires large amounts of data which are not always available with recommendation datasets. Instead, we explore transfer learning from large scale language models [PNI^{+}18] to obtain content embeddings from item titles and descriptions.

2.4.5 Methods for Implicit feedback

Our work focuses mainly on the data with implicit feedback as such information is more often recorded by E-Commerce platforms. A seminal work on matrix factorization [RFGST09] introduced Bayesian personalized ranking (BPR) loss function specifically tailored for implicit feedback, we provide the detailed form in Chapter 3. Instead of propagating the predicted rating error as done with explicit feedback learning, BPR optimizes the probability of scoring positive items over negative items. Since no pure negative items are available with implicit feedback, the negatives are usually taken from the set of unobserved items for the user. Another popular loss function choice for implicit feedback is the Weighted approximately ranked pairwise (WARP) loss [WYW13]. However, WARP requires non-standard optimization procedure which complicated the implementation and prevented us from using it. Authors of GRU4Rec [HK18] introduced

Top-1 loss function and in later revision of the same architecture proposed BPR-max and Top-1-max loss functions to combat diminishing gradient and introduce rating value regularization for implicit feedback tasks. Due to its close similarity to BPR, we did not test Top-1 loss separately, but we experimented with BPR-max, which is also discussed in more details in Chapter 3.

3 Methods

3.1 Definitions

We consider a set of users $u \in U$ and a set of items $v \in I$. I can represent for example a set of products in an online store, a set of movies in a streaming platform or a set of travel destinations in a tourist booking platform. U represents users interacting with the catalog, e.g. web shop customers. Content features associated with item v such as text or image are mapped to a tuple $F(v) = (c_1, c_2, ..., c_K), F(v) \in C$, where C is a set of content tuples for the whole catalog and c_i are embedded content features such as item description text embedding vector or item image embedding vector. We define S_u as a sequence of user interactions with the item catalog $S_u = (v_1, v_2, ..., v_T)$ where T is the number of timesteps in the sequence. We also define $S_{u,t}$ as a sequence of user interactions until the timestep t: $S_{u,t} = (v_1, v_2, ..., v_t)$. Each interaction in the sequence refers to an instance of implicit preference from the user such as item click, purchase or review. The interaction can be extended to include any additional interaction information such as explicit rating value or review text with a sentiment.

We consider the task of ranking the set of all items I based on relevance to user u after t timesteps and recommending top k items based on this ranking. The ranking score $\hat{y}_t(u, v)$ is obtained from the underlying recommender model. This is a typical task formulation for implicit feedback datasets.

3.2 Sequential content-based recommender

We propose EmbRec, a novel recommendation method. To solve item cold-start problem, we rely on item content features. The content features are converted into item embeddings. We obtain the embedding vector for item v_t as $e_t = E(F(v_t); \Theta_E)$, where E is an embedding operation, Θ_E is a set of learned parameters and $e_t \in R^n$, n is the dimensionality of the item embedding space. We test different ways of extracting content information (different $F(v_t)$).

As a solution to user cold-start problem, we adopt a recurrent neural architecture to model sequential dynamics. Recurrent cell has the form $h_t = RNN(h_{t-1}; e_t; \Theta_{RNN})$ where h_t is the hidden state at timestep t, e_t is the content embedding of the item in the user sequence at timestep t and RNN is a recurrent cell such as gated recurrent unit (GRU) [CvMBB14]. We use GRU recurrent cells which better incorporate long-term signal from previous interactions on each prediction timestep than simple RNN cells. GRU is a simpler unit than long short-term memory (LSTM) [HS97] but likewise incorporates a circuit to conditionally remember information from previous steps in the hidden state. The hidden state $h_t \in R^m$ can be considered a learned user context embedding, m being the dimensionality of the latent user context space.

The goal of the model is to predict content embedding $\hat{e_t} = G(h_t; \Theta_G)$ of the next item e_{t+1} in the user sequence given content embeddings of items in the user sequence until timestep t. We explore multiple approaches to supervise the model training including pointwise and pairwise ranking loss functions. During inference, the recommendations computation can use approximate nearest neighbour search to efficiently find item embeddings close to the predicted embedding $\hat{e_t}$.

The predicted embedding $\hat{e_t}$ is scored against target item catalog embeddings $e' \in \{v' \in I' \mid E(F(v'); \Theta_E)\}$. During training, the set of target items I' is different depending on the loss function. During inference, regardless of the loss function used in training, target scores are obtained for all items, $I' = I$. The scoring function $\hat{y_t}(u, v)$ during inference is either normalized or non-normalized inner product between $\hat{e_t}$ and e'.

The network is trained with mini-batch stochastic gradient descent on batches of user sequences with Adam optimizer [KB14]. The loss values are computed at each time step of the user sequence, the general form of the loss function is presented in Equation 1, where L_t is defined by the specific loss function in use.

$$L = \frac{1}{|U|} \sum_{u \in U} \frac{1}{|S_u| - 1} \sum_{t=1}^{|S_u|-1} L_t \tag{1}$$

Cosine distance loss This loss function directly minimizes the cosine distance between predicted and positive item embeddings (see Figure 2a). A similar approach was used in [TXL16] but based on identity features. The loss function is presented in Equation 2. During inference, the relevance score for target is obtained as cosine similarity between the target and predicted embedding, presented in Equation 3. During training, the loss value is computed only based on the positive item from the next step, $I' = \{v_{t+1}\}$, which makes it very efficient. However, we have to use content features and not learned embeddings for each item, otherwise the network will exploit the trivial solution of learning to predict all embedding vectors to be exactly the same, thus minimizing the cosine distance between all the pairs in the catalog to 0.

$$L_t = 1 - \cos(e_{t+1}, \hat{e_t}) = 1 - \frac{e_{t+1} \cdot \hat{e_t}}{\|e_{t+1}\| \, \|\hat{e_t}\|} \tag{2}$$

$$\hat{y_t}(u, v') = \cos(e', \hat{e_t}) = \frac{e' \cdot \hat{e_t}}{\|e'\| \, \|\hat{e_t}\|} \tag{3}$$

Next, we introduce loss functions which are defined over a set of target items $I' = \{v_{t+1}\} \cup I_u^-$, where I_u^- is a set of additionally sampled negative items for the user u, $I_u^- \subset I$. We employ popularity-based sampling, where the probability of selecting an item as a negative sample is proportional to the support of the item: $p_{sample}(v) = \frac{N_v}{N}$, where N_v is the number of interactions with item v and N is the total number of interactions. Items which are present in the user sequence are excluded from the negative

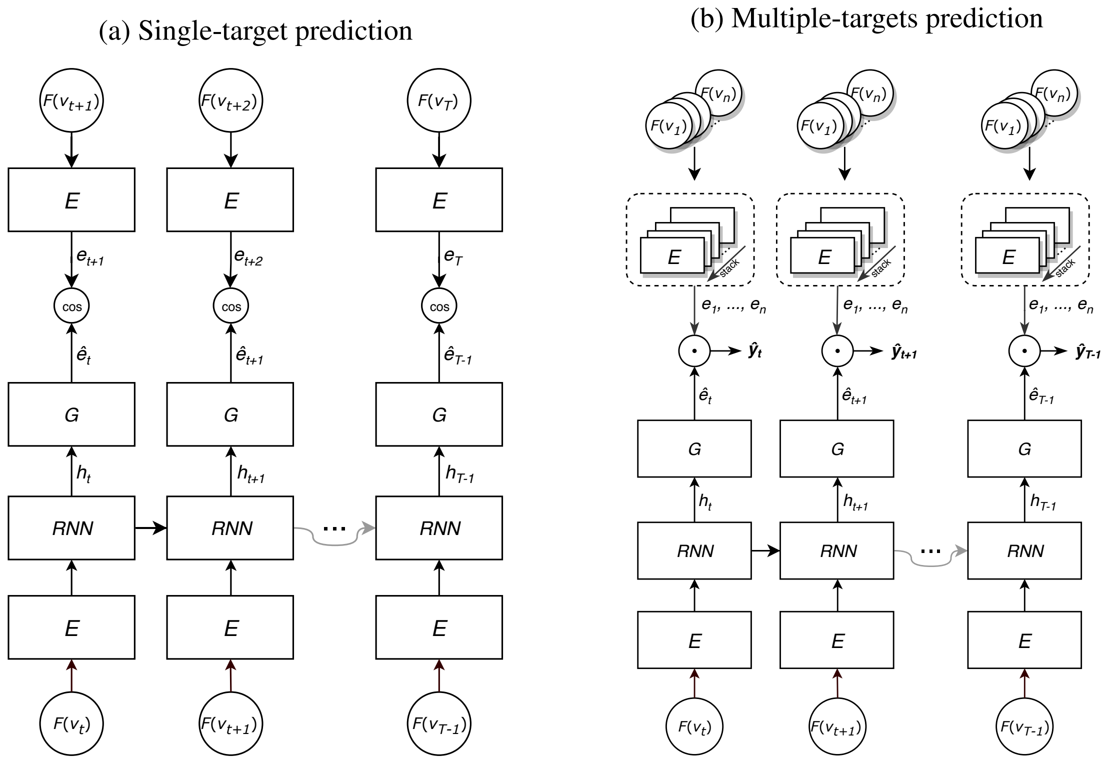


Figure 2. Recommender system neural architecture, E is the embedding operation. (a) Architecture used during training with cosine distance loss function. (b) Architecture used during training with other loss functions as well as during inference for all architectures. Target item scores computation is vectorized using a tensor multiplication operation denoted as •.

samples. This sampling schema was shown to perform well in GRU4Rec [HK18]. For all of the following loss functions, both at training and inference time, the relevance scores are computed as $\hat{y}_t(u, v') = e' \cdot \hat{e}_t$ (see Figure 2b). We denote the vector of target item scores as $\boldsymbol{\hat{y}_t}$.

Cross-entropy loss The task of next item prediction is very closely related to multi-class classification task. Cross-entropy is a widely used loss function for multi-class classification and can be also considered a pointwise ranking loss function. Cross-entropy is defined over conditional discrete probability distribution over target items, $p(v'|S_{u,t}), v' \in I'$. The probability distribution can be approximated using softmax operation over target item scores, presented in Equation 4.

$$\hat{p}(v|S_{u,t}) = \frac{\exp(\hat{y}_t(u,v))}{\sum_{v' \in I'} \exp(\hat{y}_t(u,v'))} \tag{4}$$

Softmax makes cross-entropy a list-aware loss function. Categorical cross-entropy loss is presented in Equation 5. While cross-entropy incorporates training signal from the negative item scores, it does not directly optimize the ranking of items.

$$L_t = -\log \hat{p}(v_{t+1}|S_{u,t}) \tag{5}$$

BPR loss Pairwise ranking losses have been shown to work better on implicit feedback and thus we implemented Bayesian Personalized Ranking [RFGST09] which directly optimizes the probability of scoring positive items higher than negatives. BPR is in Equation 6, where $\sigma(x) = \frac{1}{1+\exp(-x)}$.

$$L_t = \frac{1}{|I_u^-|} \sum_{v' \in I_u^-} -\log \sigma(\hat{y}(v_{t+1}|S_{u,t}) - \hat{y}(v'|S_{u,t})) \tag{6}$$

BPR-max loss Following the experiments on GRU4Rec architecture, we test the regularized variant of BPR-max loss introduced with the method [HK18]. The BPR-max loss is tailored to combat diminishing weight updates in BPR loss when the network has learned to rank the positive item above most negatives. Loss components from negative items that are scored below the positive item are close to 0. Since the original BPR averages together the loss contribution from all negative samples, the loss will be driven to zero even when there are negative examples which are incorrectly ranked above positive. Thus BPR-max weights the loss components from high-scoring negatives higher than from low-scoring ones. The loss contributions are weighted using a softmax over all negative scores. The loss function also includes the rating regularization term $\hat{y}(v'|S_{u,t})^2$ for each negative sample, which is also weighted using softmax over all negative scores. The final loss formulation is presented in Equation 7, where λ is the regularization coefficient and $\hat{p}$ is the softmax operation defined earlier.

$$L_t = \frac{1}{|I_u^-|} - \log \sum_{v' \in I_u^-} \hat{p}(v'|S_{u,t}) \sigma(\hat{y}(v_{t+1}|S_{u,t}) - \hat{y}(v'|S_{u,t})) + \lambda \sum_{v' \in I_u^-} \hat{p}(v'|S_{u,t}) \hat{y}(v'|S_{u,t})^2 \tag{7}$$

Table 1. Dataset statistics after preprocessing

Dataset	#users $\lvert U \rvert$	#items $\lvert I \rvert$	#actions	avg. #actions /user	avg. #actions /item
Auto	29,325	39,364	175,006	5.97	4.45
Clothing	165,848	172,885	1,035,906	6.25	5.99
Electronics	252,878	144,802	2,066,567	8.17	14.27
Office	16,245	22,154	126,340	7.78	5.7
Toys	53,983	68,236	396,493	7.34	5.81
Cellphone	66,583	59,711	423,891	6.37	7.1
Games	30,901	23,558	274,064	8.87	11.63
Total	**615,763**	**530,710**	**4,498,267**	—	—

3.3 Evaluation methodology

3.3.1 Data

We conduct experiments using the public Amazon reviews dataset first used in [MTSvdH15]. The data includes 142.8 million reviews spanning May 1996 - July 2014 and is split into reviews per high-level category. We chose this specific dataset because it contains detailed item features such as text title, price, text description and precomputed AlexNet image embeddings. We focus on 7 categories. We consider sequences of items reviewed by the same customers, thus the problem is technically one of predicting the next reviewed item, which is fully analogous to next purchase prediction. We filter out users and items with fewer than 5 events, reserve the last item in each user sequence for test and the one before last for validation. The validation and test splits are analogous to the ones used in TransRec experiments [HKM17] which makes our experiment results comparable. For scalability reasons, we limit each user sequence to last 50 reviews. The statistics of preprocessed categories are presented in Table 1.

3.3.2 Baselines

Our goal with the experimentation is to show that the model is able to provide accurate personalized recommendations for users with sufficient history, given the added benefit of combatting user and item cold-start.

We include the following baselines:

PopRec Simplest model which always recommends the same set of items which have the highest support in the catalog.

TransRec [HKM17] State-of-the-art personalized next-item prediction algorithm which outperformed all sequential matrix factorization methods on Amazon data. Thus, we do not include factorization models separately.

GRU4Rec [HK18] Sequence-based recommendation system based on recurrent neural networks. This is the closest architecture to our work. While original GRU4Rec was trained to provide non-personalized recommendations on session sequences, we train the network on user reviews sequences, making it personalized. We use shared learned identity embeddings of dimensionality 128 for all items on both input and output since authors reported it to be the best configuration. Like EmbRec, we use 2 shared feed-forward embedding layers applied to the input embeddings (more details in Section 3.4). We use our own implementation for GRU4Rec, given architectural similarity to EmbRec.

3.3.3 Metrics

We evaluate the accuracy of the models in offline fashion. We compute the accuracy metrics of guessing the last item in the user sequence given the preceding sequence. We use the following metrics:

Area under the ROC curve (AUC) AUC shows the probability of recommender to score positive items over negatives and it is directly optimized in the experiments with BPR and BPR-max losses. AUC estimation is presented in Equation 8, where $1_{\{True\}}$ is an indicator function which returns 1 if the argument is True. With AUC, we consider $I_u^- = I \setminus \{v_T\}$, meaning every item in the catalog except the positive on the last step.

$$AUC = \frac{1}{|U|} \sum_{u \in U} \frac{1}{|I_u^-|} \sum_{v` \in I_u^-} 1_{\{True\}}(\hat{y}_{T-1}(u, v_T) > \hat{y}_{T-1}(u, v')) \tag{8}$$

Hitrate@k In our case, Hitrate@k is same as Recall@k given that there is only one positive item at each timestep. Hit@k is 1 if the top k recommendations contain the positive item and 0 otherwise. Hitrate@k is calculated as average Hit@k across all interactions. Hitrate@k estimation is presented in Equation 9, where $r_{T-1}(u, v_T)$ is the rank of item v_T for user u after $T-1$ timesteps.

$$Hitrate@k = \frac{1}{|U|} \sum_{u \in U} 1_{\{True\}}(r_{T-1}(u, v_T) < k) \tag{9}$$

3.4 Feature extraction

3.4.1 Textual features

Different textual features are the primary and most versatile source of contextual information about the item and are often available in recommendation datasets in the form of titles and descriptions. We explore multiple ways of extracting dense high-level embeddings from those features with several techniques. As the simplest bag-of-words approach, we extract Term Frequency Inverse Document Frequency (TF-IDF) vectors from the corpus of concatenated textual information in the catalog. As a more complex bag-of-words approach, we experiment with averaged word vectors from the skip-gram vector model [MSC$^+$13] available pretrained on Tensorflow Hub[2]. Lastly, following the recent success of learning multiple downstream NLP tasks from embeddings of the pretrained large scale language models, we incorporate paragraph embeddings obtained from the ELMo model [PNI$^+$18] pretrained in an unsupervised fashion on the 1 Billion Word dataset [CMS$^+$13]. The benefit of ELMo over skip-gram and TF-IDF embeddings is that ELMo supports unknown tokens in text of newly added items without recalculation of embeddings for the whole item catalog thanks to character-level model. The final paragraph embeddings from the latter are obtained by averaging together context-embedded word vectors from all 3 layers of the ELMo encoder (see Figure 3). We do not use activations from the 3 layers as separate inputs as suggested in ELMo paper. We use ELMo model available from TensorFlow Hub[3]. For scalability reasons, we limit the number of words in textual features input for ELMo to 250 words.

3.4.2 Image features

Image information is also often available for e-commerce catalogs. Image embeddings obtained as activations from higher layers of convolutional neural networks were shown to work very well for recommendations on the data we are working with [MTSvdH15]. We use the same image features as LMT [MTSvdH15], namely the output of the second dense layer of the AlexNet pretrained on ImageNet [KSH12].

3.4.3 Fusing text and image

We experiment with fusing image and text from item content simultaneously to obtain rich item embeddings. We produce embeddings from content features by applying 2 fully connected layers with ReLU nonlinearity separately for each content feature (see Figure 4). This allows the network to convert the input to useful internal representation. The embeddings from text and images are concatenated together before the recurrent

[2]https://tfhub.dev/google/Wiki-words-250/1
[3]https://tfhub.dev/google/elmo/2

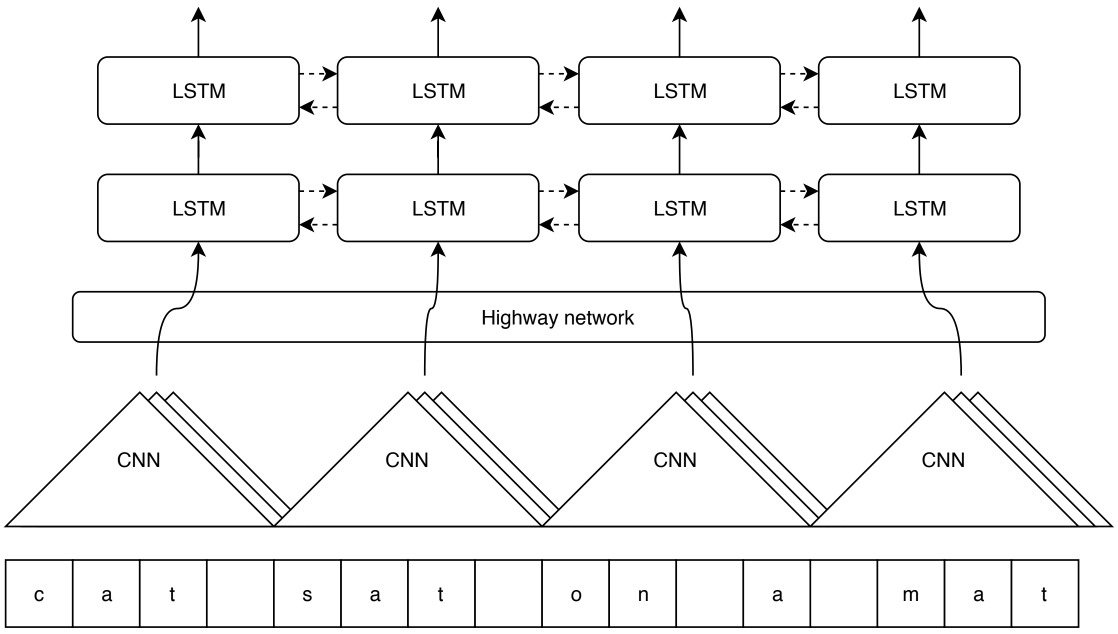


Figure 3. ELMo text encoder architecture over character input. Solid arrows represent the activations that are averaged together to obtain the paragraph embeddings

layer. The embedding part of the computation graph is reused on the input and output layers with the same weights (see Figure 2a and Figure 2b).

3.5 Training process

3.5.1 Model optimization

Since the data contains sequences of very different lengths, we have to apply padding in the end to turn batches into rectangular shape. In order to minimize dummy computations on the paddings, we distribute the sequences into buckets of similar length and then pad them to have the length of the longest sequence in the batch (see Figure 5). The loss, evaluation results and gradients are masked for the padded part of sequences. The network is trained with complete backpropagation through time. In order to prevent overfitting, we apply dropout [SHK$^+$14] on the recurrent layer.

For each of the categories in the dataset, we perform a hyperparameter search to identify optimal configurations. The hyperparameter search is performed using HyperOpt search algorithm [BYC13] and Asynchronous Hyperband trial scheduler [LJR$^+$18]. The resource allocation for parallel experimentation and trials execution is implemented with Ray Tune [LLN$^+$18].

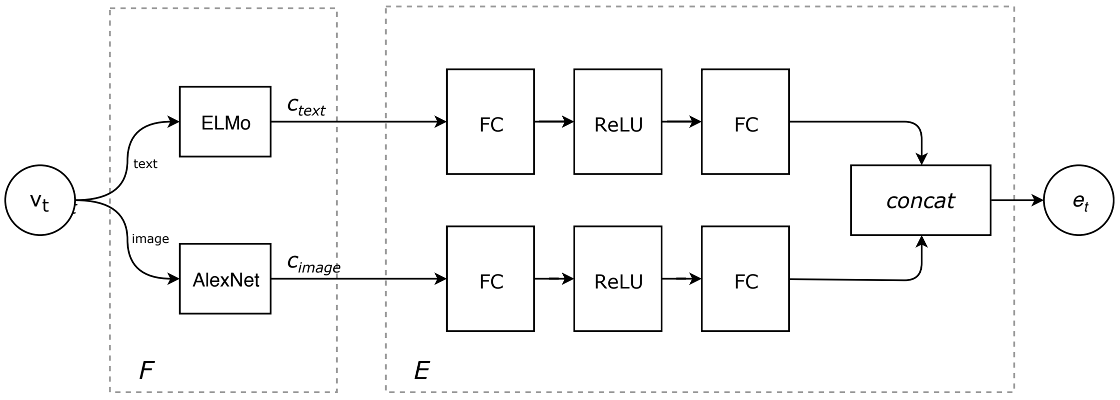


Figure 4. Embedding of the item content features. Text and image embeddings are pre-calculated and saved on disk and are not backpropagated through. FC is an abbreviation for a fully connected layer, ReLU is a Rectified linear unit.

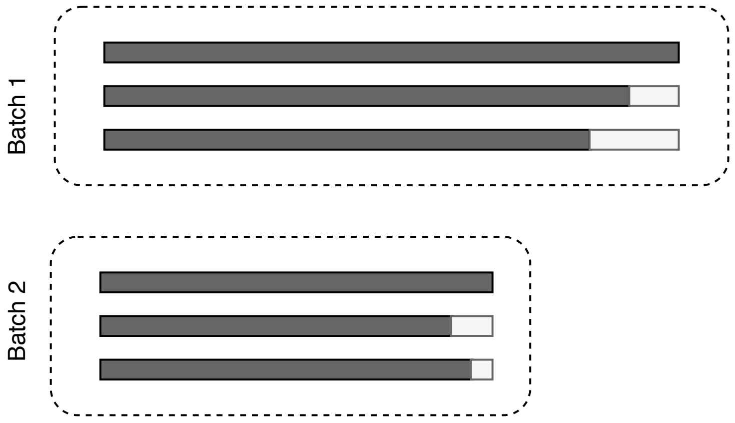


Figure 5. Bucketing and padding by sequence length. The bucket boundaries are specified based on dataset statistics.

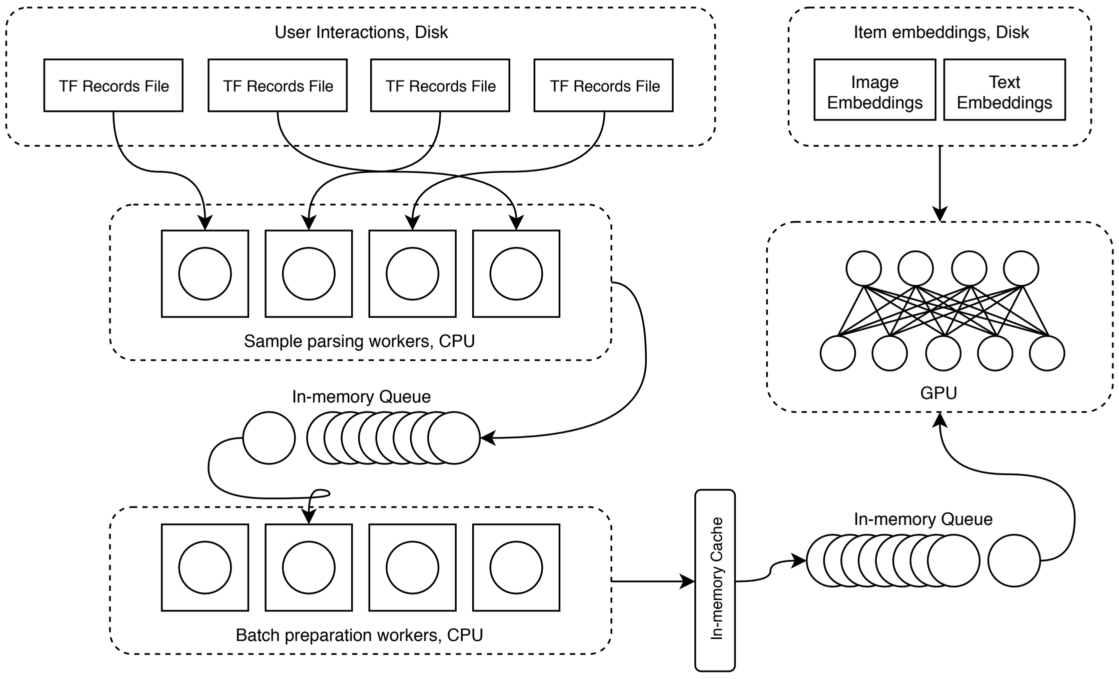


Figure 6. Concurrent data ingestion pipeline

3.5.2 Data ingestion

In order to prevent bottlenecks in batch preparation, we use a queue-based data ingestion pipeline based on Tensorflow Datasets API (see Figure 6). User interaction sequences are stored on disk in shards as TF Record binary Protocol Buffer files. Only item indices are stored in the interaction files. The sequences are parsed and loaded by a pool of workers and are queued up for batch preparation. The batch preparation is also executed in parallel by multiple workers. Batch preparation includes bucketing the sequences by sequence length and padding as well as sampling negative items for each batch. Prepared batches are cached in memory to be consumed during the following epochs. Batches are consumed from another in-memory queue by the GPU accelerator, where the network training happens. A prespecified number of batches are prefetched into the cache in order to avoid the accelerator waiting for batches to be prepared.

Item and image feature embeddings for items are extracted before the training starts and the feature vectors are stored on disk for the entire item catalog. During training and inference, the feature vectors are loaded into memory as lookup tables and are indexed into based on the item indices coming from interaction sequences. Both model training and evaluation metrics are implemented as part of the TensorFlow graph and are executed on the accelerator. The model code including branching into training and inference are implemented using TensorFlow Estimator API [CST+17].

4 Results

4.1 Test set evaluation

After performing extensive hyperparameter search for GRU4Rec and EmbRec on the validation set, we obtained test set results for best performing configurations in each category. For each category, we fixed the loss function to be cross-entropy and for EmbRec used either ELMo text embeddings alone or combined them with image embeddings. We tuned learning rate, dropout rate and, for GRU4Rec, optionally, L2 regularization coefficient due to bigger tendency of GRU4Rec to overfit training data. For EmbRec, we chose ELMo embeddings over TF-IDF to gain support for out-of-vocabulary terms in cold-start items at the cost of slightly worse performance (more details in Section 4.4). We chose cross-entropy for both GRU4Rec and EmbRec because of its superior validation performance over other loss functions in EmbRec experiments (more details in Section 4.5). Similarly, we fixed the number of negatives for both GRU4Rec and EmbRec to include all negatives because of superior validation performance in EmbRec experiments (more details in section 4.6). The results of test set experiments are given in Table 2. We report the results of TransRec from the corresponding paper since the data preprocessing and train-test splits are analogous. The best performing hyperparameter values for each category for EmbRec are presented in Table 3.

We report higher results for our method than all baselines except for the Electronics category. Note, however, that TransRec results were obtained with learning rate and latent space dimensionality fixed for all categories and only regularization coefficient was tuned on all categories separately. Results suggest that our architecture is able to outperform identity-based model only using content features.

4.2 Qualitative evaluation of recommendations

In addition to numerical evaluation, we performed manual inspection of how the recommendation model behaves. In order to do manual assessment, we generated a random sample of 100 users from the Electronics category and looked at history and top recommendations for those users.

In many cases the model was able to identify general personal preference of users, some examples are demonstrated in Figure 7. Notably, however, the model was presenting a very similar set of top recommendations in a sizable proportion of cases. In those situations, the model predicted high scores for a very limited set of items that have high support in the training data, even though the context user history in those cases was substantially diverse. We provide several examples in Figure 8. Therefore, predicting high-support items seems to be a fallback strategy for the model in cases of high uncertainty.

Table 2. Experimental results. Best results for each category are highlighted in bold.

Category	Metric	PopRec	TransRec	GRU4Rec	EmbRec
Automotive	AUC	0.5758	0.6868	0.6530	**0.7462**
	Hitrate@50	3.75%	5.07%	5.22%	**5.80%**
Clothing	AUC	0.6127	0.7243	0.7032	**0.7923**
	Hitrate@50	1.11%	2.12%	1.01%	**2.48%**
Electronics	AUC	0.7825	**0.8484**	0.8310	0.8367
	Hitrate@50	4.55%	5.19%	5.69%	**5.97%**
Office	AUC	0.6388	0.7302	0.7005	**0.7640**
	Hitrate@50	1.62%	6.51%	6.29%	**8.71%**
Toys	AUC	0.6240	0.7590	0.7279	**0.8146**
	Hitrate@50	1.69%	5.44%	3.89%	**6.45%**
Cellphone	AUC	0.6959	0.8104	0.7939	**0.8488**
	Hitrate@50	4.43%	9.54%	9.32%	**9.64%**
Games	AUC	0.7495	0.8815	0.8683	**0.8879**
	Hitrate@50	5.17%	16.44%	15.83%	**17.35%**

Table 3. Hyperparameter settings for EmbRec

Category	Learning rate	Dropout rate	ELMo embeddings	AlexNet embeddings
Automotive	2.34E-04	0.8541	✓	✗
Clothing	3.32E-04	0.8562	✓	✗
Electronics	9.90E-05	0.8156	✓	✓
Office	4.55E-04	0.7142	✓	✓
Toys	4.71E-04	0.7979	✓	✓
Cellphone	3.32E-04	0.7792	✓	✓
Games	2.00E-04	0.8000	✓	✓

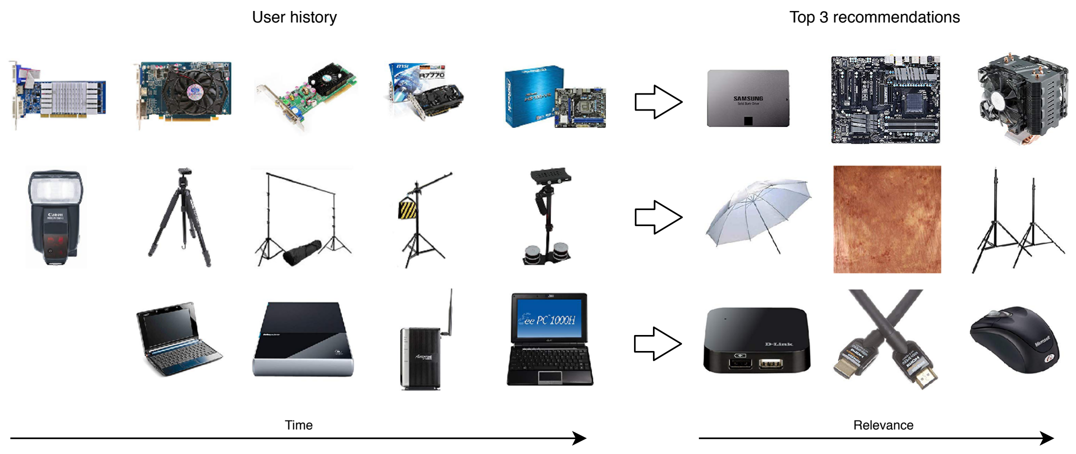


Figure 7. Example users where the model was able to capture the long-term signal

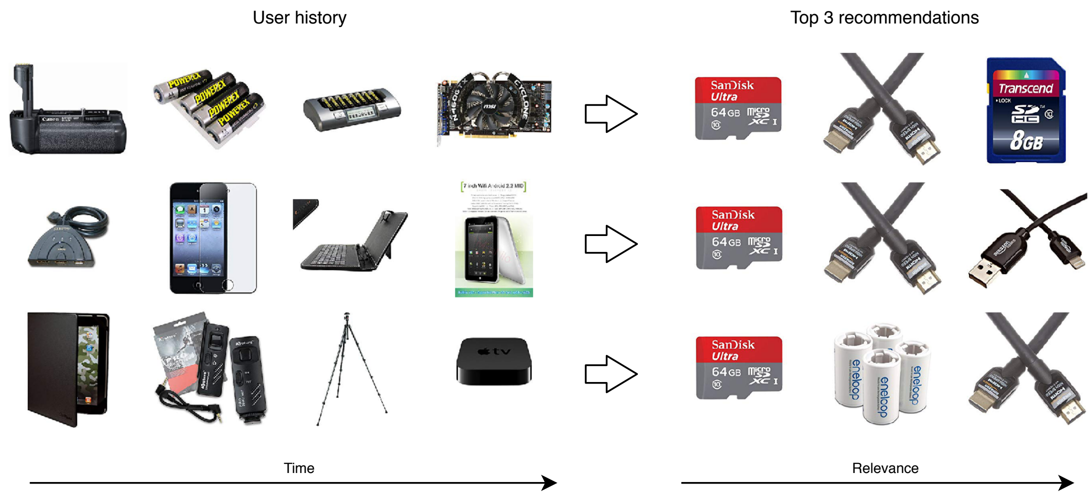


Figure 8. Examples where the model failed to capture user preferences and fell back to popular items

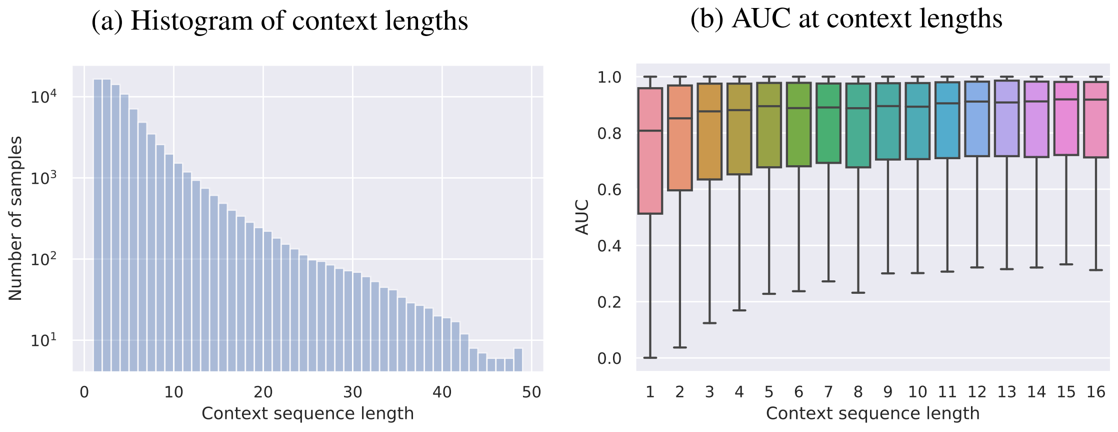


Figure 9. Model performance at different context lengths. (a) The distribution of context lengths. (b) AUC performance at different context lengths.

4.3 Model behavior based on context sequence length

Similar to experiments in GRU4Rec paper [HKBT15], we explored the accuracy of the model when dealing with shorter and longer user sequences. Since we wanted to evaluate recommendations on each step of every holdout user sequence, we could not use the test split from the baseline evaluation, which held out only last interaction from every sequence. Therefore, we held out 10% of all user sequences in Clothing category from training completely and evaluated AUC at each step in every holdout user sequence. Thus we evaluate each sequence at all its prefixes. We consider sequences of up to 16 interactions and ignore the performance on longer sequences due to very limited number of them (see Figure 9a). The AUC results by prefix lengths are given in Figure 9b.

It can be observed that longer sequence context leads to higher ranking accuracy of the model which suggests that the network is able to extract sequential signal from the user behavior and provide better personalization farther in the session.

To check if more personalization happens with longer context, we estimated the catalog coverage of recommendations provided at different context lengths. Catalog coverage is defined as the proportion of items in the catalog that are being recommended. The more personalized the experience, the more fine-grained the set of items the user receives which means that more items from the tail of the catalog are being used, thus increasing the catalog coverage. Since we had more events from shorter sequences than from longer, we could not compare directly the coverages. Coverage at sequence length 1 would be larger than for longer sequences by sheer amount of items being recommended. Therefore, we estimated the distribution of catalog coverages at each sequence length by bootstrap sampling the same number of recommendations for each sequence length 1000 times, results are given in Figure 10a.

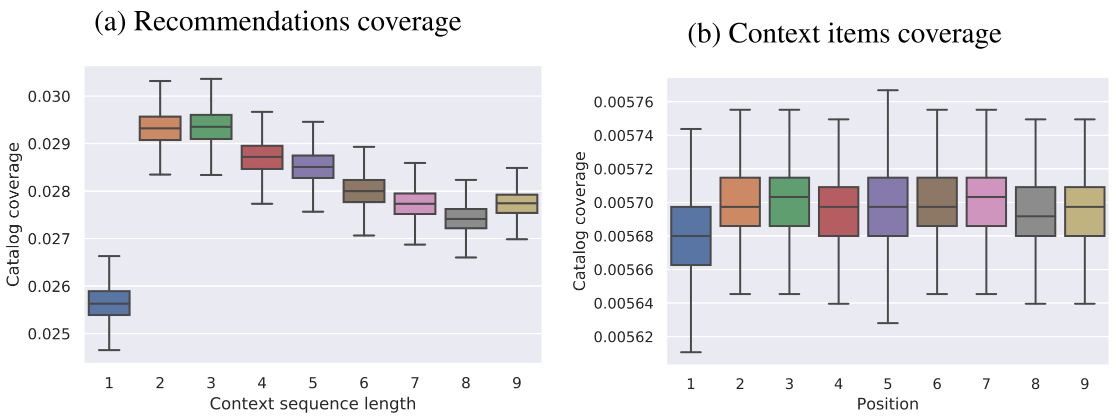


Figure 10. Catalog coverage at different context lengths estimated by bootstrap sampling. (a) Coverage of recommended items at different context lengths. (b) Coverage of items encountered as context at different context lengths.

We also make sure that the catalog coverage of context items at each position in the sequence is similar since diverse context items will enable more diverse recommendations just on their own. The distribution of context item coverages at each context lengths are presented in Figure 10b. It is indeed the case that context items at each of the compared positions have very similar coverage. Context catalog coverage at each position was also estimated using bootstrap downsampling to a fixed number of items 1000 times.

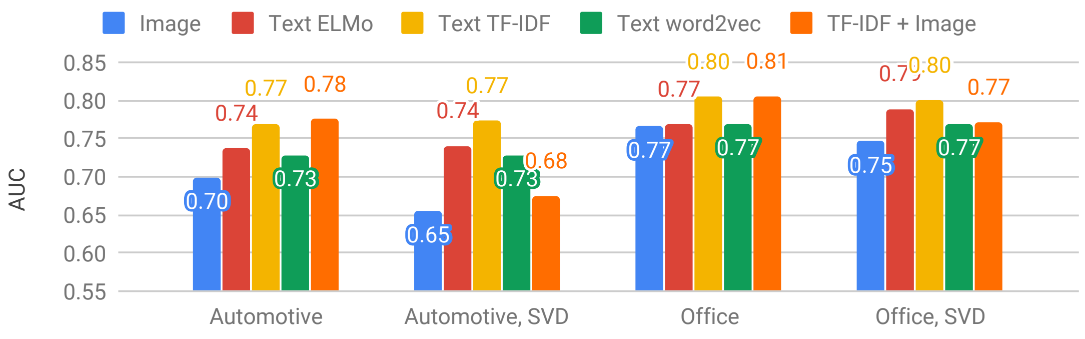
From Figure 10a, it can be seen that the catalog coverage at first timestep is smaller than at the following steps, meaning that the model sticks with a smaller set of items when facing a context sequence of only 1 item. It is notable however, that at context lengths greater than 3, catalog coverage starts dropping, which should be further investigated.

4.4 Evaluation of different content features

We performed experiments comparing the accuracy of models trained on different kinds of input features. Image embedding vectors obtained from AlexNet are of dimensionality 4096. Text embedding vectors are of dimensionality 5000 for TF-IDF, 1024 for ELMo and 250 for word2vec. Since bigger embedding size also increases the number of parameters learned on the feed-forward embedding layer, we performed separate experiments where all embedding vectors were compressed to 250 components using truncated SVD. In case of SVD-reduced TF-IDF + image input, we compress each of the sources to 125 components to sum up to 250 as well. We fix the same random seed and set of hyperparameters for all experiments to make them comparable. The results of the validation set comparisons are given for AUC (Figure 11a) and Hitrate@50 (Figure 11b).

Image embeddings are more useful is some categories than others, for instance images

(a) AUC



(b) Hitrate@50

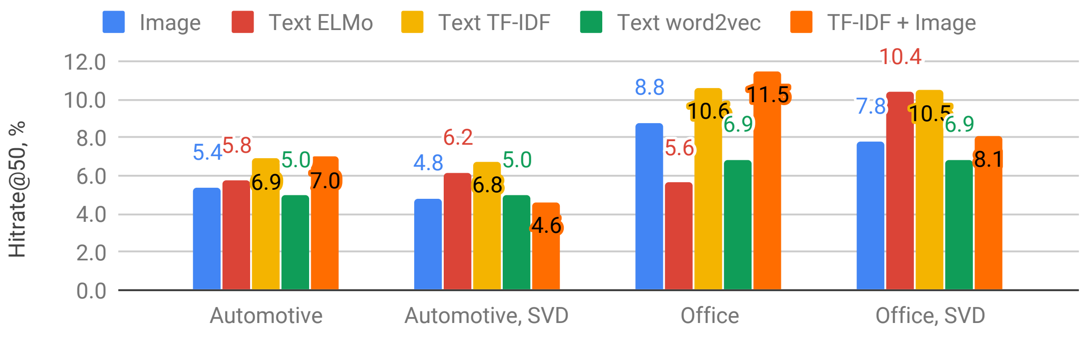


Figure 11. Performance on different input features.

Table 4. Silhouette scores for category clusters based on text embedding strategies

Method	Automotive	Office	Toys
TF-IDF	-0.14	-0.17	-0.23
Word2vec	-0.47	-0.47	-0.41
ELMo	-0.56	-0.58	-0.53

produce high results on Office data, but are quite inferior on Automotive. In general image data on its own is less expressive of the item catalog than text, but both images and text can be used in conjunction which leads to better performance but requires additional model capacity, hence SVD-reduced text and image input performs much worse than pure text.

Our experiments showed that out of all text embedding strategies, simple weighted bag-of-words approach of TF-IDF gives consistently superior results on several categories, both on full embedding and SVD-reduced embedding sizes. While ELMo is capable of processing any unknown tokens and is trained on a huge general text dataset, the descriptions of items inside the category are from a rather narrow vocabulary and TF-IDF can leverage this fact better by making use of statistics in the vocabulary. Like ELMo, shallow paragraph embeddings from the skip-gram model showed inferior results to TF-IDF, which might be explained in part by non-weighted averaging of word vectors.

To check which of the text embedding approaches better matches the inherent structure in the catalog, we estimated how the embeddings of items of the same category are positioned in the embedding space. In order to do that, we computed the Silhouette score [Rou87] of category clusters in the embedding space (Table 4). The Silhouette score measures the tightness of clusters as well as their separation and ranges from -1 to 1, 1 meaning perfect separation, 0 meaning that the clusters overlap and -1 meaning that each sample was assigned to the wrong cluster.

The Silhouette cluster scores show that TF-IDF vectors of items are embedded more consistently with categories of the items compared to neural embedding methods which suggests that TF-IDF embeddings capture the structure in the data better. It should be noted, though, that category grouping does not match the positioning of items in the embedded space very well for any of the methods.

4.5 Choice of loss function

We separately evaluate each of the loss functions on smaller categories using the validation split given the optimal hyperparameter configuration found for each loss function. All models are trained for a fixed number of gradient steps. Input data is the same for all experiments and is the SVD-reduced 250 dimensions of TF-IDF scores. The results are given separately for AUC (see Figure 12a), and Hitrate@50 (see Figure 12b).

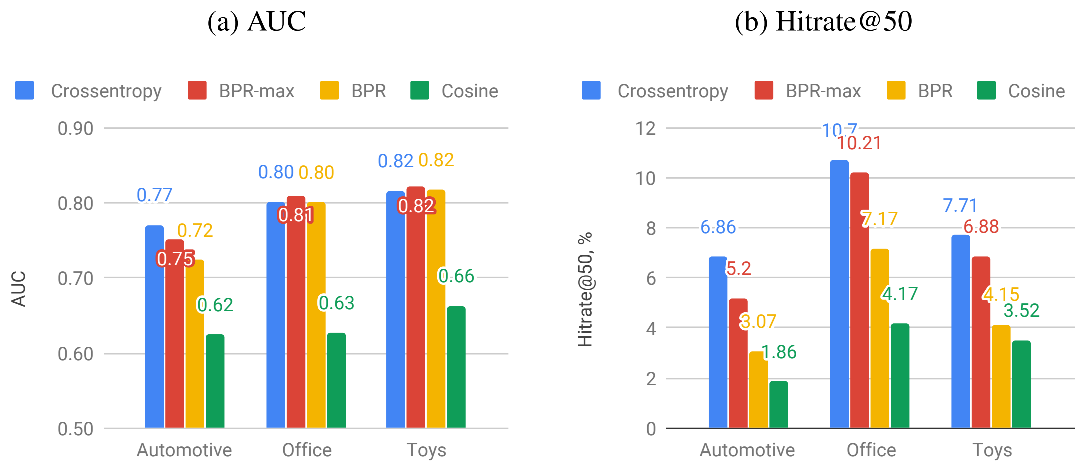


Figure 12. Validation set performance of different loss functions

Experiments suggest that cross-entropy is the prefered loss function for the architecture. Both BPR and improved BPR-max directly optimize the AUC metric during training which explains why they often slightly outperform cross-entropy on AUC evaluation. However, models trained with cross-entropy tend to perform much better according to the top-k Hitrate metric. In our case, Hitrate@50 is a more relevant metric, since we are more interested in the quality of top-k recommendations in applications of the recommendation system.

It is clear, however, that the softmax weighting in BPR-max loss indeed improves the performance compared to regular BPR loss. On the other hand, our experiments suggest that the BPR-max loss is slightly more unstable during training and needs more fine-grained tuning of the learning rate as well as gradient clipping in case of longer user sequences, where we faced exploding gradient issues.

Networks trained with the cosine distance loss underfit the data and we were not able to achieve high results with it. It should be noted as well, that the capacity of the network trained using cosine distance loss is smaller than for other loss functions due to the restriction of fixed embeddings and thus absence of embedding feed-forward layers.

4.6 Importance of number of negatives

In order to understand the importance of negative sampling, we experimented with different numbers of negative samples prepared in each batch. We tested validation set performance based on number of negative samples for each of the loss functions that use negative samples separately. In both AUC and Hitrate@50, we found that using all available products except the positive examples for negatives gives the best

(a) AUC (b) Hitrate@50

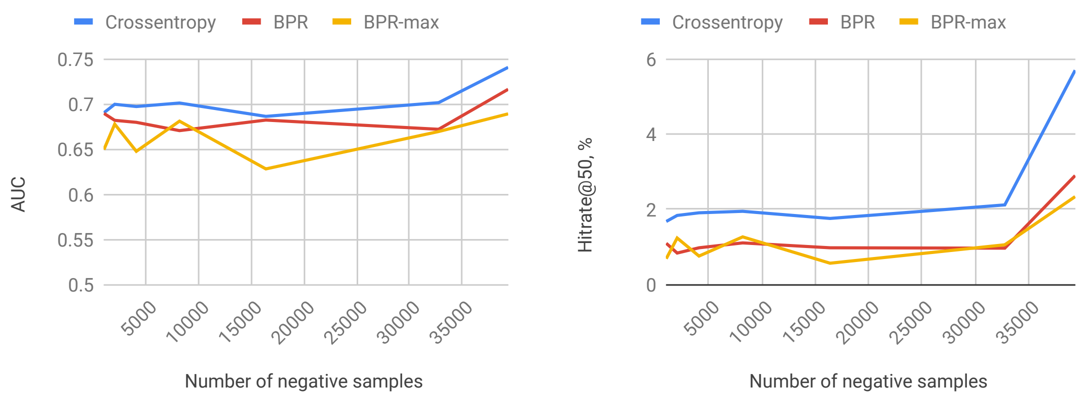


Figure 13. Validation set performance of different number of negatives.

performance. While there is a smaller payoff with AUC (see Figure 13a), the Hitrate@50 performance (see Figure 13b) increases substantially when all negative examples are included. Even though the increase from 32000 to 39000 samples is not very big, the Hitrate@50 performance is several times better with the latter.

5 Conclusions

We proposed, implemented and evaluated a personalized sequential content-based recommender system. The method handles user cold-start problem using sequential modelling with a recurrent neural network and item cold-start problem with image and text embedding strategies. We performed extensive hyperparameter optimization as well as experimentation comparing 4 different content embedding strategies and 4 loss functions commonly used in the recommendation system literature. Numerical experimentation against strong baselines showed high results on a large public dataset of Amazon reviews.

While we explored several options to obtain text and image embeddings, the landscape for possible approaches includes more recent convolutional architectures such as ResNet [HZRS16] and more expressive language models such as BERT [DCLT18]. Recent successes with metric embeddings for factorization methods [HKM17], [MTSvdH15], [HPM16] suggest possible improvements if metric optimization objectives are used instead of inner products. Furthermore, parallel recurrent neural architectures where separate neural networks are focusing on specific input channels [HQKT16], [WAB$^+$17] have shown good results, albeit are much more complicated to train. Similarly, second-order neural architectural interactions such as attention models and memory networks are a very promising future direction for sequential recommendation systems [TATH18], [LLH17].

Acknowledgements

I would like to thank Carlos Bentes for initial research direction and all the consequent technical and organizational support as well as fruitful discussions in the moments of necessity that made this work possible. I am sincerely grateful to Tambet Matiisen for proposing the novelty in the presented approach as well as providing technical and academic guidance that enabled me to shape and develop this work. I would also like to thank my employer STACC OÜ for providing hardware resources necessary for experimentation as well as supporting the development of the project. This research has been supported by European Regional Development Fund under the grant no EU48684. This work was carried out in part in the High Performance Computing Center of University of Tartu. The work was also supported by Dora Plus Sub-Action 2.1 Support for international master‘s students.

European Union
European Regional
Development Fund

Investing
in your future

References

[ABAH14] Charu C Aggarwal, Mansurul A Bhuiyan, and Mohammad Al Hasan. Frequent pattern mining algorithms: A survey, 2014.

[Agg16] Charu C Aggarwal. Knowledge-Based recommender systems, 2016.

[AT11] Gediminas Adomavicius and Alexander Tuzhilin. Context-Aware recommender systems, 2011.

[BBM16] Trapit Bansal, David Belanger, and Andrew McCallum. Ask the GRU, 2016.

[BCJ$^+$18] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent cross, 2018.

[BL$^+$07] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA., 2007.

[Bur02] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Model. User-adapt Interact.*, 12(4):331–370, November 2002.

[BYC13] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[CMS$^+$13] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[CST$^+$17] Heng-Tze Cheng, David Soergel, Yuan Tang, Philipp Tucker, Martin Wicke, Cassandra Xia, Jianwei Xie, Zakaria Haque, Lichan Hong, Mustafa Ispir, Clemens Mewald, Illia Polosukhin, Georgios Roumpos, D Sculley, and Jamie Smith. TensorFlow estimators, 2017.

[CvMBB14] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–Decoder approaches, 2014.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[HK18] Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations, 2018.

[HKBT15] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. November 2015.

[HKM17] Ruining He, Wang-Cheng Kang, and Julian McAuley. Translation-based recommendation, 2017.

[HKTR04] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems, 2004.

[HKV08] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets, 2008.

[HM16] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation, 2016.

[HPM16] Ruining He, Charles Packer, and Julian McAuley. Learning compatibility across categories for heterogeneous item recommendation, 2016.

[HQKT16] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations, 2016.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term memory, 1997.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. December 2014.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems, 2009.

[KNK13] Santosh Kabbur, Xia Ning, and George Karypis. FISM, 2013.

[Kor10] Yehuda Koren. Collaborative filtering with temporal dynamics, 2010.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LJR$^+$18] Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. February 2018.

[LLH17] Pablo Loyola, Chen Liu, and Yu Hirate. Modeling user session and intent with an attention-based encoder-decoder architecture. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 147–151. ACM, 2017.

[LLN$^+$18] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. July 2018.

[MR07] Tariq Mahmood and Francesco Ricci. Learning and adaptivity in interactive recommender systems, 2007.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[MTSvdH15] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-Based recommendations on styles and substitutes, 2015.

[PB07] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[PNI$^+$18] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[QKHC17] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks, 2017.

[Ren12] Steffen Rendle. Factorization machines with libFM, 2012.

[RFGST09] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[RFST10] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation, 2010.

[Rou87] Peter J Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, 1987.

[RRS11] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook, 2011.

[Sch04] Barry Schwartz. The paradox of choice: Why more is less. Ecco New York, 2004.

[SHK$^+$14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SKKR01] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms, 2001.

[TATH18] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 729–739. International World Wide Web Conferences Steering Committee, 2018.

[TXL16] Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations, 2016.

[WAB$^+$17] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks, 2017.

[WHC$^+$17] Jian Wei, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. Collaborative filtering and deep learning based recommendation system for cold start items, 2017.

[WYW13] Jason Weston, Hector Yee, and Ron J Weiss. Learning to rank recommendations with the k-order statistic loss, 2013.

[ZCM01] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. Using temporal data for making recommendations. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 580–588. Morgan Kaufmann Publishers Inc., August 2001.

[ZYST19] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):5, 2019.

Appendix

I. Licence

Non-exclusive licence to reproduce thesis and make thesis public