Temporal-Contextual Recommendation in Real-Time

Yifei Ma*
yifeim@amazon.com
AWS AI Labs

Balakrishnan (Murali) Narayanaswamy*
muralibn@amazon.com
AWS AI Labs

Haibin Lin
haibilin@amazon.com
AWS Deep Engine

Hao Ding
haodin@amazon.com
AWS AI Labs

ABSTRACT

Personalized real-time recommendation has had a profound impact on retail, media, entertainment and other industries. However, developing recommender systems for every use case is costly, time consuming and resource-intensive. To fill this gap, we present a black-box recommender system that can adapt to a diverse set of scenarios without the need for manual tuning. We build on techniques that go beyond simple matrix factorization to incorporate important new sources of information: the temporal order of events [Hidasi et al., 2016], contextual information to bootstrap cold-start users, metadata information about items [Rendle 2012] and the additional information surrounding each event. Additionally, we address two fundamental challenges when putting recommender systems in the real-world: how to efficiently train them with even millions of unique items and how to cope with changing item popularity trends [Wu et al., 2017]. We introduce a compact model, which we call hierarchical recurrent network with meta data (HRNN-meta) to address the real-time and diverse metadata needs; we further provide efficient training techniques via importance sampling that can scale to millions of items with little loss in performance. We report significant improvements on a wide range of real-world datasets and provide intuition into model capabilities with synthetic experiments. Parts of HRNN-meta have been deployed in production at scale for customers to use at Amazon Web Services and serves as the underlying recommender engine for thousands of websites.

CCS CONCEPTS

• **Information systems** → *Enterprise applications*; **Personalization**; **Recommender systems**; *Learning to rank*; • **Computing methodologies** → *Sequential decision making*.

KEYWORDS

recommender systems, recurrent neural networks, real-time, collaborative filtering, content filtering, hybrid model, negative sampling

ACM Reference Format:
Yifei Ma, Balakrishnan (Murali) Narayanaswamy, Haibin Lin, and Hao Ding. 2020. Temporal-Contextual Recommendation in Real-Time. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403278

*Both authors contributed equally to the paper.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs International 4.0 License.
KDD '20, August 23–27, 2020, Virtual Event, CA, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403278

1 INTRODUCTION

Personalized recommendation systems aim to present the right items to a user, at the time that it is most useful to her. They typically use two sources of information: user-item interaction histories and user/item features [21]. User histories aid the discovery of personalized items by observing patterns about what "similar people typically do". On the other hand, the user/item features help users discover new items from other users who have had similar contexts and information needs. Both sources of information add fundamental value by helping users find useful information efficiently; information that is otherwise buried in the "long-tail" of a large catalog or website. Machine learning provides a way forward toward building a system that could address the following distinct challenges - the system should adapt to user requests in real-time; learn from long user histories including hundreds of events; be effective for "cold-start" users and items; debias historical trends to "predict the future"; and scale to large item catalogs.

Many recommendation models are implicitly or explicitly built on user and item latent representations, whose inner-products represent the relevance of the item to the user, e.g. by decomposing a matrix of user-item interaction counts [10, 22, 26]. Traditionally, these approaches are limited in their ability to use information in the timing and order of user events, and as a result are less responsiveness to changes in user behavior, e.g. new interests. This need has motivated novel user-state models ranging from multi-hot auto-encoders [24] to n-gram factorization machines [21] and recurrent recommender networks [11, 34]. Extending recurrent language models [4, 13], these models preserve order information and can adapt to new user events in real-time.

In addition to the order of events, the timing of interactions is key to building models that can deal with long user histories. One typical assumption is that user behaviors can be grouped into sessions, within which the users tend to have similar intents, e.g., watching different episodes of the same series. To model long user sequences with temporal patterns, Quadrana et al. [20] proposed a two-layer hierarchical GRU (HGRU) model: one layer models intra-session dynamics and another layer, which performs updates only at the end of each session, captures inter-session dynamics. HGRU can learn from longer sequences because the hierarchical structure allows for shorter gradient paths to distant pasts. However, the specific implementation of HGRU can be inefficient since it requires direct manipulation of the RNN hidden states. In this paper, we show that we can encode time information at the input, through a learned embedding, and achieve comparable or even better results. This not only achieves computational efficiency but also allows us to potentially extend these hierarchical structures to other RNNs (e.g., LSTMs) and beyond (e.g., TCNs, Transformers). We call our model HRNNs. Through evaluation on a number of real world datasets, we show that HRNNs achieve most of the performance of models like HGRU, with a 4x-10x reduction in computational costs, due to a 4x smaller network and utilization of GPU local cache.

Time is only one example of contextual meta-data that a recommendation model may learn from. We extend HRNN to include user and item features such as location, language preference, item location, item pricing, and other sources of information, by stacking the contextual information on top of the respective user/item latent vectors through "field-aware" multi-layer perceptrons [14], an extension we call HRNN-meta. We see that it is able to generalize based on features and improve recommendation coverage in "cold-start" scenarios. We show close-to-SOTA recalls on CIKM 2016 semantic search challenge (Figure 4), with bag-of-words representations of query tokens and product descriptions as contexts.

To further improve cold-start item recommendation over time, we model item state evolution, inspired by [34]. We decompose item states into two parts: a trend-following bias term and a time-invariant user-affinity score. We contribute theoretical insights for the decomposition. Without such a decomposition, RNN models learn item selection policies that maximize the likelihood of user interaction. These policies may suffer from exposure bias, where popular or often recommended items tend to get higher scores over time, even after they are no longer as popular. When we explicitly impute a per item popularity bias term, the residual scores become consistent with the log-click-rate of an item when a user is actually exposed to the item - that is we can account for some exposure bias. In Figure 5, we show that we can combine user-affinity scores with updated item trend biases in real-time to achieve better performance, even with models that are updated less frequently.

Practical recommender systems must be able to serve large item catalogues, while being trained in reasonable amounts of time. Large item sizes often cause throughput and memory challenges in vanilla RNN models. Consider a standard setting with 32 length BPTT, 32 mini-batch size and 256-dimensional hidden layers on a GPU with 16GB of memory. The "dense" decoder layer of a RNN becomes a computational bottleneck when the number of unique items m, exceeds $m > 100k$ and a storage bottleneck for BPTT gradient propagation when $m > 1M$. Inspired by recent progress in language models [12, 13] and large-scale recommender systems [36], we use importance sampling (IS) to approximate the expected gradients in back-propagation. Table 8 shows that IS achieves similar speed and accuracy performance when "dense" decoders are available, but 10x speedup and better accuracy when the "dense" decoders are at the limit of some modern hardware.

2 HIERARCHICAL RECURRENT NETWORKS WITH META DATA (HRNN-META)

We start with the problem of making personalized recommendations based on an interaction dataset, where each row contains a historical record of {(time, user id, item id, value)} = $\{(t_k, u_k, a_k, v_k) : k = 1, \ldots, n\}$. Value can be used to represent e.g. click vs. purchase. We will further extend the representation to include other contexts, e.g. time of interaction, rating, purchase value, user profiles, product features, etc. In the next sections we describe each of the components of the HRNN model, and demonstrate some of their theoretical and empirical properties.

2.1 Sequence Models of User Interactions

Sequence models gain statistical power by aggregating "people who watched X also watched" from individual user histories, with the added ability to capture recency and trends in customer behaviour. We group interactions into ordered user histories, $\mathcal{X}_i = [x_{i1}, x_{i2}, \ldots, x_{ik_i}]$, with temporal re-indexing, where $x_{ik} = (a_{ik}, t_{ik}, v_{ik})$ and $t_{ik} > t_{ij}$ if $k > j$ (most prior work ignores the timestamp t_{ik}, and only uses the order of events). We omit user indices i below, noting that we learn a global model of items-to-item dynamics across users by stochastic gradient descent on mini-batches of user histories. We use RNNs such as GRUs to predict the probability that a user may interact with a specific item given their history, similar to how language models learn to predict the next word in a sentence [28]. Let A_{k+1} be a random variable representing the next item a_{k+1}, $x_{1:k} = (x_1, \ldots, x_k)$ be the history, and $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_m)^\top$ be the scoring vector function for each of the m items; we model:

$$a_{k+1} \sim p(A_{k+1} \mid x_{1:k}) = \text{softmax}(\boldsymbol{\phi}(\mathbf{h}(x_1, \ldots x_k))), \qquad (1)$$

where $\mathbf{h}_k = \mathbf{h}(x_{1:k})$ is a learned (hidden state) representation of the corresponding user history. A naive state can be the multi-hot encoding of past items. If we use $\boldsymbol{a}_k = (0, \ldots, 0, 1, 0, \ldots, 0)^\top$ to denote the indicator vector corresponding to the item index a_k, the hidden state is then $\mathbf{h}(x_{1:k}) = \boldsymbol{a}_1 + \cdots + \boldsymbol{a}_k$. RNNs can be viewed as a weighted sum of representations of past events, where the weights are learned to 'gate' or 'weigh' representation updates by the learned salience of every item interaction in the context of the items that the customer has interacted with so far.

Figure 1 sketches RNN sequence models and our hierarchical extensions. While (1) describes the prediction of a single item given an entire user history, in practice, we apply Back-Propagation Through Time (BPTT) for all user events concurrently, up to a fixed length of rolling histories and in a mini-batch of multiple users.

2.2 Hierarchical Models and Contextual Inputs

RNNs may have a limited ability to model long sequences due to vanishing gradients. In our model, **hierarchical sessions** are used to divide long user histories into short activity intervals within which the sequence has some consistency, e.g. a single user intent. Session breaks can be inferred from periods of inactivity, or provided. HGRU [20] uses separate RNNs for inter- and intra-session dynamics which are combined at the end of each session. We reproduce this separation logic in Figure 2 and we introduce one modification to rerun the intra-session GRU after the inter-session HGRU updates, which leads to better prediction for the first item in each session. However, when the mini-batch size is longer than the average session length, multiple GRU layers must be sequentially executed before moving to the next time-stamp, because the higher layers are used by more than one user in expectation. This leads to slow training speed. As an alternative, we simply concatenate a

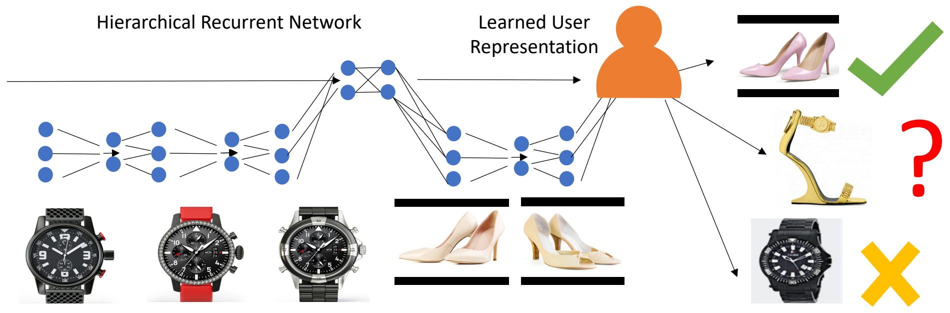


Figure 1: HRNN sequence model to predict the next item in a recurrent fashion. The time-deltas since the last clicks often positively correlate with the possibilities of changes in attention and intent.

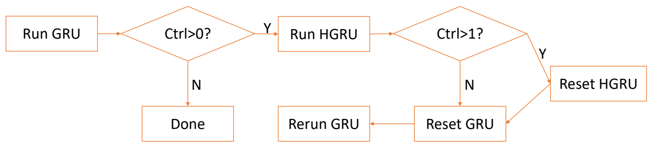


Figure 2: Example two-level HGRU with time-delta controls.

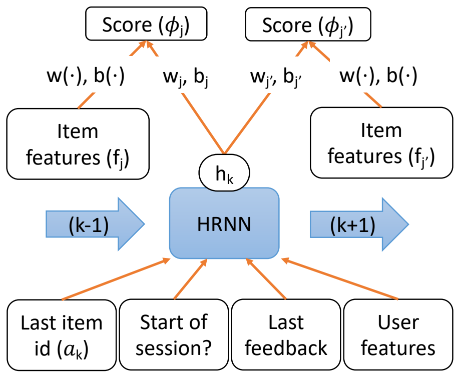


Figure 3: HRNN-meta cell model.

session start "control signal" with the input, shown as the second input bit in Figure 2. Our experiments empirically verify that allocating such special "**control**" inputs allows the model to learn to reset states as appropriate, in properly trained hierarchical models.

While the control input captures the time-delta context between the current recommendation and the one before, we may incorporate other contextual features in a similar manner. We concatenate **user features** (e.g., age) and **user dynamic features** (e.g., device or language of the request) with the one-hot encoded item id inputs and control inputs. Additionally, we concatenate encoded **feedback events** (e.g., ratings, purchases, costs, view duration etc.) of the previous item as input context to the next recommendation. For example, HRNN can learn from previous rating values to predict future rating values as part of the scoring basis. HRNN can also learn to be language/device aware while matching semantically similar contents. Besides concatenation, we also explored more complex models such as multiplying the RNN states with scalar feedback values or deep-cross models [32]. However, they yielded similar performance in our preliminary experiments, likely due to the representation power of the internal gates in GRU cells.

Our novel contribution of feedback encoding together with the session control inputs allows better **response to implicit negatives**, e.g., a refresh without any clicks, or a short dwell time. This is possible when the model is trained with larger (smaller) weights on losses associated with positive (negative) feedback. Due to connections to explicit feedback (Section 3), the model learns both the relative user-affinity of these items and their temporal associations.

2.3 Item Features and Cold Start

Item features can naturally be included in HRNNs, once we notice the connection between RNN decoders and factorization models. Standard RNN decoders compute the score of the jth item with function $\phi_j(\mathbf{h}_k) = (\mathbf{w}_j^\top \mathbf{h}_k + b_j)$, where $\mathbf{w}_j \in \mathbb{R}^r$ represents the coefficients and $b_j \in \mathbb{R}$ is the intercept. Alternatively, let $\mathbf{f}_j$ be the feature vector for item j; we view $\mathbf{w}_j = \mathbf{w}_j(\mathbf{f}_j)$ as the embedding vector of the jth item and $b_j = b_j(\mathbf{f}_j)$ as the feature-related bias. We reconsider the decoder as a second-order interaction model between $\mathbf{w}_j$ and $\mathbf{h}_k$ and combine both views as:

$$\tilde{\phi}_j(\mathbf{h}_k) = (1-\lambda)\phi_j(\mathbf{h}_k) + \lambda\phi'(\mathbf{h}_k, \mathbf{f}_j), \text{ where}$$
$$\phi_j(\mathbf{h}_k) = (\mathbf{w}_j^\top \mathbf{h}_k + b_j) \text{ and } \phi'(\mathbf{h}_k, \mathbf{f}_j) = (\mathbf{w}(\mathbf{f}_j)^\top \mathbf{h}_k + b(\mathbf{f}_j)), \quad (2)$$

where $\mathbf{w}(\cdot), b(\cdot)$ are learnable functions that embed item features and $0 \le \lambda \le 1$ is a mixing parameter.

Item features are particularly useful when recommending **cold-start** items which have no interaction data. In this scenario, we found that vanilla training resulted in learned item feature embeddings that were complements of the embeddings learned from the stronger signal in user-item interactions and were not useful for cold-start. We randomized λ during training, a form of dropout that prevents co-training, [27] and were able to learn complementary embeddings. Setting $\lambda = 1$ at inference time results in cold-start inference based on metadata. To understand the statistical implications and alternative solutions, we will dive deeper into the loss function in Section 3.

3 IMPLICIT AND EXPLICIT FEEDBACK

HRNNs are implicit-feedback models, which learn to optimize the log-likelihood of the next item interacted with, out of the set of all items. This is equivalent to policy optimization in single-step environments [15], which are designed to maximize on-policy expected rewards with reduced variance and provable eventual convergence [18]. However, it becomes useful to model explicit feedback in scenarios of negative sampling, item cold-start, and non-uniform rewards. In this section, we discuss some connections in this line.

Following (1), we use $\boldsymbol{\phi}(\mathbf{h}_k) \in \mathbb{R}^m$ to denote the vector of all item scores for a user at time t_{k+1}. Let $\mathbf{r}(t_{k+1}) \in [0,1]^m$ be the true reward vector, e.g. the expected click counts, after integrating out the internal stages of candidate retrieval and item exposure. Let $\mathbf{q}(t_{k+1})$ be an imputed exposure bias vector. We focus on model consistency in infinite data case, under the influence of exposure bias, and omit from the notation dependencies that are clear from context. The original softmax cross-entropy loss solves for:

$$\arg\max_{\boldsymbol{\phi}} \mathbf{r}^\top \Big[\boldsymbol{\phi} - \log \textstyle\sum_j (e^{\phi_j}) \mathbf{1}\Big] \quad \Rightarrow \quad \boldsymbol{\phi}^* = \log \mathbf{r} + Const. \quad (3)$$

3.1 Negative Sampling

Negative Sampling (NS) is a process by which only a small subset of weights are updated with each example, to speed up training. NS with a softmax objective is a special case in noise-contrastive estimation [8]. When the item space exceeds $m \geq 100k$, it becomes inefficient to compute every entry of $\boldsymbol{\phi} \in \mathbb{R}^m$ for every user-item pair. Instead, between any pair of random items $a_j, a_{j'}$, the probability that a_j is the true item is given by $\frac{r_j}{r_j + r_{j'}} = \text{sigmoid}(\phi^*_j - \phi^*_{j'})$ (3). As a consequence, we may sub-sample negative items uniformly to replace the set of full items. Sub-sampling converts an implicit-feedback model on all items to an explicit-feedback model of pairwise comparisons. While not "negatively" sampled, the true item can be considered "positively" sampled by nature from the same distribution, when the estimated model converges to the true model.

When the true item coverage is low, uniform sampling may have low hit-rates, which leads to insufficient sampling over potentially relevant items and ineffective training. [8] proposed to sample from a known distribution $\mathbf{q}$, which is fast to sample from, yet close to the true distribution $\mathbf{r}$. Since

$$\textstyle\sum_J \big[r_J 1_{\{J=j\}}\big] = \sum_J q_J \big[\frac{r_J}{q_J} 1_{\{J=j\}}\big], \quad (4)$$

the probability of a successful draw with the new distribution is proportional to (r_j/q_j). We therefore revise the loss to $\frac{r_j/q_j}{r_j/q_j + r_{j'}/q_{j'}} = \text{sigmoid}\big((\phi^*_j - \log q_j) - (\phi^*_{j'} - \log q_{j'})\big)$. I.e., we **subtract** $(\log \mathbf{q})$ from the estimated scores and evaluate loss only on the sampled subset. We follow [13] to extend sigmoid to softmax with multiple negative draws. On the other hand, different from [13], we use item frequencies to construct $\mathbf{q}$ to be more adaptive to varying item probabilities, and use the Alias Method [30] to improve sampling speed. Negative sampling imputes exposure bias, it does not change the recommendation scores - the resulting models will always be consistent regardless of the sampler choice, as shown in Table 8.

3.2 Item Trend Decomposition

Item trend decomposition is our attempt to separate out the impact of item popularity changes and exposure bias. It changes how we impute negative feedback and thus changes the recommendation scores. It is particularly important in cold-start scenarios. Suppose the imputation is correct, i.e., q_j truly reflects the propensity score that the user might have access to item a_j. If we further sub-sample the items from $\mathbf{q}$ after the data is collected, the chance that an item is both sampled and clicked becomes r_j/q_j, due to (4). From (3), we obtain that the HRNN solution on the sub-sampled data becomes $\phi^*_j = \log(r_j/q_j)$. This is desirable, because we effectively solved for the item Click-Through Rate (CTR), conditioned on the user being exposed to the item. Instead of sub-sampling, we may **add** the bias term $(\log \mathbf{q})$ to the full score vector to reduce variance for small item spaces. To summarize, we mathematically decompose the next-item click/interaction probability with multiplicative factors of item CTR and item exposure rate,

$$a_{k+1} \sim p(A_{k+1}|\mathbf{h}_k, t_{k+1}) \propto p(A_{k+1}|\mathbf{h}_k) q(A_{k+1}|t_{k+1}). \quad (5)$$

A practical estimator for item exposure rate may come from its global popularity in the last time period. In Figure 5, we use item trend decomposition to successfully improve performance on news recommendation, where the news trends change quickly.

3.3 Non-Uniform Rewards

Using the connections to policy optimization [15], we assign non-negative weight values $r_{a_{k+1}}$ to achieve non-uniform rewards. Suppose all items are presented uniformly at random, the reward-weighted cross-entropy objective leads to asymptotic convergence to the expected reward of every item in logarithmic terms (3), up to a global constant shared by all items. This allows our models to pick the highest-reward items. When the items are presented according to a different distribution $\mathbf{q}$, the objective combines both effects from item exposure and user feedback, to improve upon previous recommendation policies. If the exposure distributions are logged or can be imputed, we may use (5) to further decompose the scores to reflect the expected reward of every exposed item [3, 29].

An alternative to reward-weighting is direct reward estimation. Comparatively, reward-weighting provides "safe" improvements when the item exposure distributions are positively biased, with data collected from previous recommendation results. Reward estimation yields smaller variance on well-balanced data. Our future work considers both approaches for bias-variance trade-off [33].

4 EXPERIMENTS ON REAL DATASETS

We conduct studies to show the effects of different aspects of HRNN-meta, including session information, user and item features, and interaction feedback in different scenarios such as cold start and with large numbers of items. Clearly separating out the effects of some features required simulated data, which we describe in the extended version. The primary test dataset we use is **MovieLens** [9], a dataset of movie ratings, though we also report results on other datasets including Yoochoose, Reddit, Outbrain and Taobao. The conclusions were largely insensitive to specific training hyper-parameters. We typically used hyper-parameters that have been successful in language models [1]. We use **perplexity (PPL, [2])** as a measure of model inaccuracy. A recommendation system with a PPL of p is equivalent to one that recommends a uniform random selection of p items, one of which is the true next item. We also measure Precision, Recall, Mean-Reciprocal Recall (MRR), and Normalized Discounted Cumulative Gain (NDCG) [25].

[1]https://github.com/dmlc/gluon-nlp/

4.1 Meta-Data Models

Table 1 evaluates the full meta-data model from Figure 3. The results show that each modeling aspect was useful. We use **movielens data (ml-20m)** as a public, real-world dataset for movie recommendations. It contains 20 million interactions, 131 263 items,[2] and 138 493 unique users. We split the data by user into 80% train and validation set and 20% test set, and hold out interactions from the last few time steps (a temporal hold out) of the validation set. Table 1 shows that our vanilla RNN implementation achieved 3x the performance of popularity baseline, i.e., PPL 447 versus 2228, similar to [5].[3] The meta data model also improved the prediction accuracy, decreasing PPL to 429 and 410. Here, item features were the movie genre vectors, interaction feedback were the standardized rating values. Notice, using just the item features of the last item the user interacted with (row 2), improved over the popularity baseline to PPL 1342, showing that the model can personalize recommendations based on just the genre of the user's last interaction.

Table 1: Using meta-data lowers the PPL, which indicates a higher likelihood of recommending relevant items on ml-20m. For details on the toy dataset see the extended version.

Seq	User	Item	Feedback	Toy	ml-20m
×	×	×	×	100	2228
×	×	✓	×	54	1342
✓	×	×	×	3	447
✓	✓	✓	✓	1	410

4.2 Hierarchies, Sessions and Time

Hierarchical models are based on the hypothesis that users tend to have similar intents within-session and may change intents between sessions. During training, hierarchical models provide short gradient paths across sessions avoiding vanishing gradients and thus allowing models to have longer memory. We investigate if HRNNs trained on MovieLens do indeed learn such a temporal pattern. As a qualitative example, we consider a user who watched the following titles: Secret of Roan Inish (Children|Drama|Fantasy|Mystery) , Postman (Comedy|Drama|Romance), Thin Blue Line (Documentary), Say Anything... (Comedy|Drama|Romance) and Babe: Pig in the City (Children|Comedy). In Table 2 we see that for the same item watch history HRNN-meta generates different recommendations for different time-delta gaps between last interaction and recommendation time, indicating that the model has learned to place some weight on the temporal inputs. When the recommendation is made by HRNN within session, i.e. when a short time delta is used at query time, the theme of the recommended item stays relatively similar to the last watched item. When the input session control indicates a large time delta or hierarchy, the recommended genres tend to be more diverse, i.e., diminishing personalization effects, as expected. Note, the popularity is not monotonic for shorter time deltas, making this statement subjective.

[2]We treat the size of the index space as the number of items despite only 26 744 unique items being included in the interaction data.
[3]There are no standard temporal train-test splits on ml-20m, so we cannot make a more direct comparison.

Table 2: Top-1 recommendation to the same user changes as the inference time changes. As time between last click and inference increases, the recommendations change but stay in the genres the user has interacted with.

Δt	title	genres	popularity
0	Purple Rose of Cairo	Comedy|Drama	0.000236
60	Unbearable Lightness	Drama	0.000209
3600	Local Hero	Comedy	0.000195
86400	Big	Comedy|Drama	0.001130

Table 3: Hit@5 on the Reddit dataset. We pick one-hour browsing gaps as sessions break signals. We observe performance improvements as the session-based model gains complexity. The eventual performance also depends on other factors such as the size of the hidden states. Hierarchical models improve performance for small model sizes.

	GRU	HRNN	HGRU	Pop	BPR[5]
50 hidden	0.26	0.42	0.47	0.11	0.39
200 hidden	0.55	0.55	0.55		

4.3 RNNs vs. HRNNs

To see if hierarchies are useful, we compare the three models – a vanilla GRU, HRNN with input encoding and a Hierarchical GRU on a **Reddit** dataset[4], where some prior work has indicated the value of hierarchies [23]. The dataset contains users, subreddits they interacted with and the timestamps of those interactions. The dataset has 18 271 users and 27 452 items, results are in Table 3. We use the same train and test splits as in [23]. Similar to [23], we see that for small hidden sizes (50), HGRU and HRNN substantially outperform vanilla RNN. *HRNN is 10 times faster than HGRU, about as fast as an RNN.* A preliminary conclusion is that allowing the model to learn to encode the input timestamps provides much of the benefit of HGRU at a substantially lower computational cost.

We then experimented with larger hidden layer sizes and alternative algorithms. We see that BPR performs much better than initially reported GRU numbers in [23]. With hidden layer size of 200, our GRU substantially improves over HGRU with small hidden dimension and BPR. Incremental improvements with HRNN and HGRU on this dataset were small. We conclude that, given sufficient data, a large RNN performs well even for datasets which seem to have substantial session level effects. In Section 4.6 we explore improving the scaling and throughput of large RNN models.

We next test feedback encoding - the ability to encode and use the type or quality of interaction - within the context of sequential rating predictions, with results in Table 4. In this task we predict the rating of each item the user interacts with, and use **Root Mean Square Error (RMSE)** between the true rating the user assigned to the item and the predicted rating, to measure performance (lower is better). Standard RNNs without feedback encoding are unaware of the user rating patterns and do not outperform a rolling average

[4]https://www.kaggle.com/colemaclean/subreddit-interactions

baseline which predicts the next item rating as the average of the item average rating and user average rating. This reflects the fact that each user has a distinct rating pattern on this dataset, and highlights the challenges we face when directly using RNNs and sequential models like HGRU for rating prediction. RNNs with feedback encoding not only outperformed the baselines but also our implementation of Factorization machines and I-AutoRec. The gating mechanism in RNN learns to automatically trade-off and appropriately weight short-term attention and long-term memories. We see similar performance in the **Netflix** dataset[5] in Table 5.

Table 4: ml-20m rating prediction. Feedback encoding (one of our meta data fields) is the key to improve rating RMSE beyond the rolling average baseline.

	RMSE
Rolling average baseline	0.933
Factorization machine [21]	0.916
I-AutoRec [24]	0.871
RNN wo. feedback encoding	0.941
RNN w/ feedback encoding	0.857
HRNN w/ feedback encoding	0.846

Table 5: Netflix-full rating prediction. Feedback encoding distinguishes HRNN models from previous approaches.

	RMSE
Rolling average baseline	0.954
TimeSVD++ [16]	0.928
PMF [19]	0.925
RRN [34]	0.910
HRNN w/ feedback encoding (ours)	0.856

In Table 6 we compare RNNs and HRNNs on the Recsys2015 **Yoochoose** dataset[6], following the evaluation strategy in [11]. Again we see that well trained RNNs perform very well when compared with baselines, on most of the metrics we measure. Since an RNN with a large hidden dimension can capture some hierarchical effects if sufficient data is available it may hard to obtain substantial improvements over well trained RNNs with hierarchical models.

4.4 Cold Start with Item Metadata and IPS

One important application of item meta data models is recommending items without prior interaction histories, e.g. new items, in a personalized manner. Offline evaluation of cold-start recommendation quality on public datasets is difficult since, almost by definition, cold start items rarely occur. We simulate cold-start by holding out items on two real-world datasets.

[5]https://www.netflixprize.com/. We used the full version of the dataset, with a test set from the full month of December 2005 and train set containing all previous 98.1M interactions, for a total of 17.7k unique items, following [34].
[6]https://2015.recsyschallenge.com/

Table 6: Yoochoose ranking loss (MRR@20, Hit@20, and NDCG@20 in percentiles) and perplexity. HRNN with time-delta contexts outperforms our GRU implementation, which further outperforms the original GRU4Rec.

	MRR	Hit	nDCG	PPL
Popular items	0.1	0.6	0.2	17683
Hidasi et al. [11]	26.9	63.2	-	-
GRU (ours)	30.1	69.4	45.1	225
HRNN (ours)	31.4	70.8	46.4	204

MovieLens Cold Start with Genre Information. To test the HRNN-meta model with cold start, we held out half of all items (selected uniformly at random from the entire movielens (**ml-1m**) interactions dataset) and evaluated the recommendation performance only on the held-out items. Notice, a baseline strategy that does not use meta-data can only recommend items uniformly at random, since without any history even item popularity is unknown. Table 7 shows that HRNN-meta model significantly improved cold-start recommendation over the uniform-random baseline. We used (2) with $\lambda = U[0,1]$ during training and $\lambda = 1$ during inference to recommend cold-start items only using their meta data.

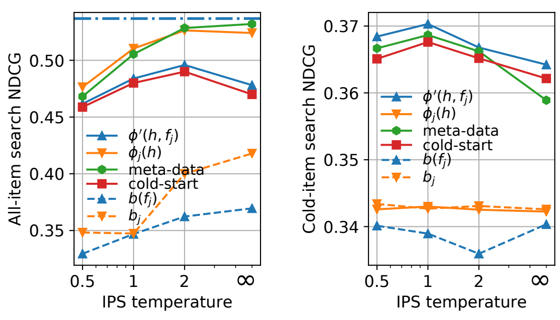
Table 7: ML-1M item cold start. Cold-start items have no interaction histories so won't be recommended by a collaborative filtering algorithm. We use the HRNN-meta model to find relevant cold start movies using the genre meta-data.

	Random choice	HRNN-meta
MRR	0.109	0.323
Precision@5	0.039	0.162
Precision@25	0.041	0.133
NDCG@5	0.075	0.208
NDCG@25	0.187	0.388

CIKM Cup 2016 Semantic Search. [7] To measure meta data performance, we use bag-of-words (BoW) features and a product search dataset. Following Zhang et al. [35], we consider 37.7k train queries and 16k test queries as users, and all 184k unique products as candidates. Query/product features come from their bag-of-words (BoW) representation, where we used only the top 3000 frequent words and performed a square-root transform on the BoW features. To study the effects of IPS off-policy correction, we imputed a global impression bias by counting item frequency in the training set.

Figure 4(a) shows that the NDCG with our meta-data model is comparable with SOTA from Zhang et al. [35] (green versus dot-dash blue line at the top). Using cold-start models hurt performance because there is no significant distribution shift between the train and test splits, i.e., total-variation loss is lower between train/test (15%) than two random folds splitting the test set (18%). We compare 6 models : referencing (2), $\phi'(\mathbf{h}, \mathbf{f}_j)$ a model based only on meta-data, $\phi_j(\mathbf{h})$ a model based only on interaction data, a full meta-data model, a cold start model ((2) with $\lambda = 1$ during evaluation)

[7]https://cikm2016.cs.iupui.edu/cikm-cup/



(a) Original all-item search **(b) Cold-start item search**

Figure 4: IPS off-policy correction on CIKM Cup 2016 search challenge. Lower temperature implies more weight on the IPS term. (a) The original all-item search does not have significant distribution drifts; static $\phi(\mathbf{h}_k)$ is the primary signal and IPS may hurt overall accuracy. (b) Cold-start simulation by splitting items between train/test sets. Meta-data $\phi'(\mathbf{h}_k, \mathbf{f}_j)$ is the primary signal and IPS is beneficial.

and two non-personalized models. We see that models that include interaction data consistently outperform models that do not. We see that priors from meta-data embedding are washed out by evidence from interactions data, when there is no distribution shift.

To simulate distribution shift, we split items into train and test so that the test set only contains cold-start items. Figure 4(b) shows that the meta-data embedding is useful in the cold start scenario and that IPS helps address the distribution shift. Cold-start with $\lambda = 1$ also alleviates distribution shifts when IPS temperature is ∞.

4.5 Retraining with Item Trend Corrections

In addition to the one-time item cold-start case, many application domains, such as news and live events require the model to identify trends when new items are continuously introduced and old items removed. We show how to use a combination of retraining and item frequency correction to solve the problem.

To demonstrate that modeling item dynamics and user-item correlations separately could benefit the learning process and improve performance, we tested our model on the Outbrain news recommendation dataset. We chose this dataset because it has the nice property that the item popularity changes often as news articles expire quickly. Outbrain covers 2 weeks user-item interaction history. We filtered the dataset by selecting interactions based on items with a specific publisher ID. We performed a temporal train-test split and used the first 70% data for training and the rest 30% for testing.

We further divided the test data into 1 hour frames. Offline HRNN was tested iteratively on the 1 hour frames while the (batch) online HRNN was retrained after each frame. Online HRNN also leverages item popularity information to debias the fast-changing item trends and stabilize the learning process, so it can be retrained on a longer history without worrying that the model may capture the outdated item trends. We use recent popularity as our baseline, a model which predicts the next item solely based on item popularity over a recent window.

In the experiment, both offline and online HRNN are trained for 5 epochs. During retraining, the online HRNN is trained on most recent 10 hours and tested on the next 1 hour slot, and we calculate item popularity on an hourly basis and feed it into the model. Similarly, the Recent Popular model calculates item popularity for each hour and uses it to predict over the next hour. As shown in Figure 5, the performance of offline HRNN drops drastically in the first 20 steps (20 hours) and then remains stable. The online HRNN and Recent Popular, show significantly better performance when compared with offline HRNN, albeit with some fluctuations. The dataset is strongly popularity biased, with dynamic item trends which makes Recent Popular a competitive baseline. Online HRNN outperforms Recent Popular most of the time illustrating that it learns personalized recommendation on top of item popularity.

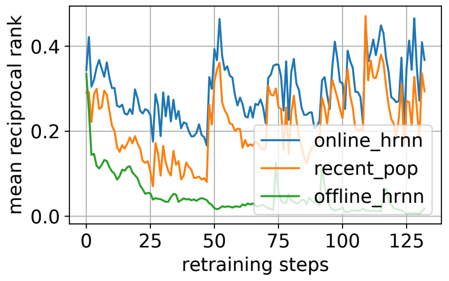


Figure 5: Comparing offline/online learning with/without IPS corrections. News articles are time-sensitive. The offline models quickly become outdated. Performance can be much improved from IPS corrections and batch online retraining.

4.6 Large Catalogs and Importance Sampling

Table 8 shows **throughput and accuracy comparisons** between dense output layers and IS layers. The movielens datasets used a GRU model with 43-dimensional hidden states. All models were trained for 4 epochs. The PPL and NDCG were measured on predictions of held out last-items for test users, using their previous histories as inputs.

IS-based training yields models with quality consistent with the original dense training on movielens datasets - differences seem to be primarily due to other modeling factors, and were not significant even given the sizes of the datasets. We also conducted experiments on a large-scale dataset from **Taobao**[8] described in [36]. We followed the train/test splits as in [36] - i.e., we held out $10k$ users for testing where we evaluated on the second half of items they interacted with, using the first half of the items as input - maintaining a temporal split. We used an additional filtering step, only retaining users and items with at least 10 historical interactions in the training data. The filtering step resulted in a less than 5% data drop, but significant decrease in item catalogue size from around $4M$ to around $1M$. Even with this smaller catalog, the dense model requires >16GB memory which is only available on new and expensive hardware. On the other hand, IS reduces the memory footprint by 2.5x (from 30GB to 14GB) and yields a 11x speed-up.

[8]https://tianchi.aliyun.com/dataset/dataDetail?dataId=649

Table 8: HRNN throughput and accuracy. Boldfonts/underlines show significant/insignificant differences, respectively. IS significantly improved model training for up to 1M unique items (Taobao dataset). Additionally, the dense model for the Taobao dataset requires >16GB memory for BPTT, which is infeasible on many GPUs.

	Output	ml-1m	ml-10m	ml-20m	Taobao
Output size	IS	62	255	362	1087
(m)	Dense	1683	65 134	131 263	1 183 451
Throughput	IS	23k	20k	**20k**	**7.8k**
(#items/sec)	Dense	23k	23k	17k	631
PPL	IS	377	405	455	**17.6k**
	Dense	409	439	494	119k
NDCG	IS	0.128	0.123	0.12	**0.15**
	Dense	0.123	0.119	0.115	0.08

We limited training time to 6 hours. Due to its higher throughput, the IS-approx model yielded significantly better performance on the test set- a 6x improvement on the perplexity metric and 2x increase in the NDCG@25 value.

From the throughput and accuracy experiments, we conclude that when number of items $m < 100k$, recurrent layer computations dominate the training time and IS-approximation is not useful. When $m \geq 1M$, we see significant improvements in throughput, leading to much better model performance given the same amount of training time.

Table 9: Comparing IS-approximated HRNN with other large-scale methods. The values are comparable, suggesting good recommendation quality, with the exception of HRNN dense on Taobao dataset (>$1M$ output size), which is so inefficient that it failed to train in limited time (<6 hours).

	ml-20m		Taobao	
	p@10	r@10	p@200	r@200
YouTube product-DNN	11.87%	8.71%	1.48%	7.58%
TDM attention-DNN	14.06%	10.55%	**2.00%**	**10.81%**
HRNN dense	**24.81%**	**11.16%**	0.85%	3.97%
HRNN IS-approx	23.23%	10.87%	1.54%	7.97%

Our IS-approximated HRNN model performed about as well as external benchmarks, suggesting good recommendation quality. Exact comparisons are subject to other factors such as training time and model complexity. For example, we trained for only 6 hours with a generic IS sampler by raising the item frequencies to the 0.75-th power, whereas for the better performing state-of-the-art TDM attention-DNN model [36] the original authors did not report training time. We speculate that some data errors are possible: e.g. we discovered negative time stamps in the dataset, which may add some uncertainty to the benchmarks.

We conclude that IS approximations lead to large throughput improvements and near-state-of-the-art results with short training time (<6 hours). Two additional epochs in three hours improved our results to **p@200=1.66%** and **r@200=8.57%**, respectively. We believe the remaining performance gaps can be closed by additional training time and/or better IS negative samplers. Our goal is to achieve good results while reducing training time, so we do not explore this further here. Besides [36], other possible improvements include self-contrastive estimators [6] and IRGAN [31].

5 RELATED WORK

Our work builds on seminal work on applying RNNs to recommendation systems. GRU4Rec [11], and follow up work that demonstrated the value of choosing appropriate loss funcitons and the idea of using hierarchical models to capture long term inter-session intent dynamics using HGRUs [20]. Ruocco et al. [23] extend the study of hierarchical models to new datasets, and focus on specific evaluations e.g. recommendations at the start of a new session. While much of this prior work used GRUs, Donkers et al. [5] show that a modified recurrent unit can achieve better performance.

In this context our work further solidifies the case that RNNs are good models of user behaviour for next item recommendation. We show that HRNNs where time is an input to the model, rather than an enforced hierarchical structure achieve a good trade-off of efficiency and accuracy. In addition, our focus is on dynamic systems with meta-data and cold start, and we demonstrate the scaling value on the meta-data and item space. Most related to our work is by Li et al. [17], who model time-deltas in recurrent models, but without making the connection to hierarchical sessions and do not consider meta-data, cold start and IS.

For dynamic recommendation, Wu et al. [34] show the value of capturing item popularity dynamics using item specific RNNs. We show that much of this value can be captured using item popularity as a bias term, and further that we can improve item cold-start performance with item trend decomposition.

For meta-data modeling, we took inspiration from a few important papers on Factorization Machines [21], Deep&Cross models [7], and TimeSVD++ [16]. These models are able to regress click/rating values on the meta-data features from all user/item pairs, which naturally extends to feature-based cold-start item recommendation. Additionally, while the models are proposed as non-sequence models, it is natural and often important to represent user activity histories as part of their features by "multi-hot"-encoding their past items with exponential moving weights. However, these models are not easy to extend to large item catalogs with millions of items. We instead built on the formulation of Sedhain et al. [24], which jointly outputs multiple rating predictions on multiple items in the same minibatch. This allows for fast scoring on multiple items at inference time, and allows us to use ranking losses as opposed to regression or classification losses.

Part of the reason for our choice of ranking loss over value regression is summarized by [1] and the related work therein. The main argument we rely on is that the desired goal should be to contrast positive events against explicit negatives that are shown at the same time, to cancel out the effects from confounding variables, such as un-modeled seasonality and time varying item popularity. When explicit negatives are unavailable, Bottou et al. [1] suggest propensity fitting of the impression data from observed variables. We extend the idea to solve the item cold-start problem in dynamic environments with time varying item propensities and show convincing empirical evidence that propensity fitting improves the recommendation of both regular and cold-start items.

Negative sampling in large-catalog problems follows similar arguments, but it is considered beneficial bias to bootstrap model training. Jozefowicz et al. [13] use the idea of negative sampling to solve large-scale language training up to 800k unique items. The idea is to generate good-quality candidate items and learn to discriminate the true items from the fake. Other similar formulations include [31, 36], that extend the discriminator and generator to more complex forms. We adapt the solution to recommendation problems at scale and bring connections between the dynamic item popularity drifts and generative-discriminative modeling.

6 CONCLUSION

Our work sought to investigate five important directions in modern recommendation systems: how to represent long interaction sequences, how to incorporate contextual information when systems collect varied rich meta data, how to handle large item catalogs, how to provide recommendations for cold start items and how to provide useful recommendations in non-stationary domains. HRNNs achieve SOTA results on numerous datasets and have been deployed in production at scale for customers to use at Amazon Web Services. Our work also provides support to the importance of sequential models that can also capture temporal context during training and inference, something that we will continue to explore in future work. Practical recommender systems are often simultaneously influenced by multiple sources of information, where failures in any model/data component can lead to catastrophic consequences. It is therefore important future work to identify or suggest root-causes to accuracy issues. We also found limitations in open-source datasets to study the complex interplays between multiple factors that are important in dynamic environments.

REFERENCES

[1] Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X Charles, D Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. 2013. Counterfactual reasoning and learning systems: The example of computational advertising. *The Journal of Machine Learning Research* 14, 1 (2013), 3207–3260.

[2] Peter F Brown, Vincent J Della Pietra, Robert L Mercer, Stephen A Della Pietra, and Jennifer C Lai. 1992. An estimate of an upper bound for the entropy of English. *Computational Linguistics* 18, 1 (1992), 31–40.

[3] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-K Off-Policy Correction for a REINFORCE Recommender System. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 456–464.

[4] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[5] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. 2017. Sequential user-based recurrent neural network recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 152–160.

[6] Ian J Goodfellow. 2014. On distinguishability criteria for estimating generative models. *arXiv preprint arXiv:1412.6515* (2014).

[7] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 55–64.

[8] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*.

[9] F Maxwell Harper and Joseph A Konstan. 2016. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)* (2016).

[10] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web*. Perth Australia, 173–182.

[11] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and D Tikk. 2016. Session-based recommendations with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016*.

[12] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On Using Very Large Target Vocabulary for Neural Machine Translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*.

[13] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410* (2016).

[14] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware Factorization Machines for CTR Prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, Boston Massachusetts USA, 43–50.

[15] Jens Kober and Jan R Peters. 2009. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*. 849–856.

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

[17] Yang Li, Nan Du, and Samy Bengio. 2017. Time-dependent representation for neural event sequence prediction. *arXiv preprint arXiv:1708.00065* (2017).

[18] Yifei Ma, Yu-Xiang Wang, and Balakrishnan Narayanaswamy. 2019. Imitation-Regularized Offline Learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*. 2956–2965.

[19] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *Advances in neural information processing systems*. 1257–1264.

[20] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 130–137.

[21] Steffen Rendle. 2010. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 995–1000.

[22] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. 452–461.

[23] Massimiliano Ruocco, Ole Steinar Lillestøl Skrede, and Helge Langseth. 2017. Inter-session modeling for session-based recommendation. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*. ACM, 24–31.

[24] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. AutoRec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 111–112.

[25] Guy Shani and Asela Gunawardana. 2011. Evaluating recommendation systems. In *Recommender systems handbook*. Springer, 257–297.

[26] Yue Shi, Martha Larson, and Alan Hanjalic. 2014. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)* 47, 1 (2014), 3.

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

[28] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*.

[29] Adith Swaminathan and Thorsten Joachims. 2015. Counterfactual risk minimization: Learning from logged bandit feedback. In *International Conference on Machine Learning*. 814–823.

[30] Alastair J Walker. 1974. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters* 10, 8 (1974), 127–128.

[31] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. IRGAN: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 515–524.

[32] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. ACM, 12.

[33] Yu-Xiang Wang, Alekh Agarwal, and Miroslav Dudık. 2017. Optimal and Adaptive Off-policy Evaluation in Contextual Bandits. In *International Conference on Machine Learning*. 3589–3597.

[34] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. ACM, 495–503.

[35] Yuan Zhang, Dong Wang, and Yan Zhang. 2019. Neural IR Meets Graph Embedding: A Ranking Model for Product Search. In *The World Wide Web Conference*. ACM, 2390–2400.

[36] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning Tree-based Deep Model for Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1079–1088.