Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations

Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, Ed Chi
Google, Inc.
{xinyang,jiyangjy,lichan,zcheng,heldt,aditeek,zhezhao,liwei,edchi}@google.com

ABSTRACT

Many recommendation systems retrieve and score items from a very large corpus. A common recipe to handle data sparsity and power-law item distribution is to learn item representations from its content features. Apart from many content-aware systems based on matrix factorization, we consider a modeling framework using two-tower neural net, with one of the towers (item tower) encoding a wide variety of item content features. A general recipe of training such two-tower models is to optimize loss functions calculated from in-batch negatives, which are items sampled from a random mini-batch. However, in-batch loss is subject to sampling biases, potentially hurting model performance, particularly in the case of highly skewed distribution. In this paper, we present a novel algorithm for estimating item frequency from streaming data. Through theoretical analysis and simulation, we show that the proposed algorithm can work without requiring fixed item vocabulary, and is capable of producing unbiased estimation and being adaptive to item distribution change. We then apply the sampling-bias-corrected modeling approach to build a large scale neural retrieval system for YouTube recommendations. The system is deployed to retrieve personalized suggestions from a corpus with tens of millions of videos. We demonstrate the effectiveness of sampling-bias correction through offline experiments on two real-world datasets. We also conduct live A/B testings to show that the neural retrieval system leads to improved recommendation quality for YouTube.

CCS CONCEPTS

• **Information systems** → **Information retrieval**.

KEYWORDS

Recommender systems; Information Retrieval; Neural Networks

ACM Reference Format:
Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, Ed Chi. 2019. Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations. In *Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3298689.3346996

RecSys '19, September 16–20, 2019, Copenhagen, Denmark

ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3346996

1 INTRODUCTION

Recommendation systems help users discover content of interest across many internet services, including video recommendations [12, 18], app suggestions [9], and online advertisement targeting [38]. In many cases, these systems connect billions of users to items from an extremely large corpus of content, often in the scale of millions to billions, under stringent latency requirements. A common practice is to treat the recommendation as a retrieval-and-ranking problem, and design a two-phase system [9, 12]. That is, a scalable retrieval model first retrieves a small fraction of related items from a large corpus, and a fully-blown ranking model re-ranks the retrieved items based on one or multiple objectives such as clicks or user-ratings. In this work, we focus on building a real-world learned retrieval system for personalized recommendation that scales up to millions of items.

Given a triplet of $\{user, context, item\}$, a common solution to build a scalable retrieval model is: 1) learn query and item representations for $\{user, context\}$ and $\{item\}$ respectively; and 2) use a simple scoring function (e.g., dot product) between query and item representations to get recommendations tailored for the query. Context often represents variables with dynamic nature, such as time of day, and devices users are using. The representation learning problem is typically challenging in two ways: 1) The corpus of items could be extremely large for many industrial-scale applications; 2) Training data collected from users' feedback is very sparse for most items, and thus causes model predictions to have large variance for long-tail content. Facing the well-reported cold-start problem, real-world systems need to be adaptive to data distribution change to better surface fresh content.

Inspired by the Netflix prize [32], matrix factorization (MF) based modeling has been widely adopted for learning query and item latent factors in building retrieval systems. Under the MF framework, a body of recommendation research (e.g., [21, 34]) addresses the aforementioned challenges in learning from a large corpus. The common idea is to leverage the content features of query and item. Content features can be roughly defined as a wide variety of features describing items beyond item id. For example, content features of a video can be the visual and audio features extracted from video frames. MF-based models are usually only capable of capturing second-order interactions of features, and thus have limited power in representing a collection of features with various formats.

In recent years, motivated by the success of deep learning in computer vision and natural language processing, there is a large amount of work applying deep neural networks (DNNs) to recommendations. Deep representations are well suited for encoding complicated user states and item content features in low-dimensional embedding space. In this paper, we explore the applications of

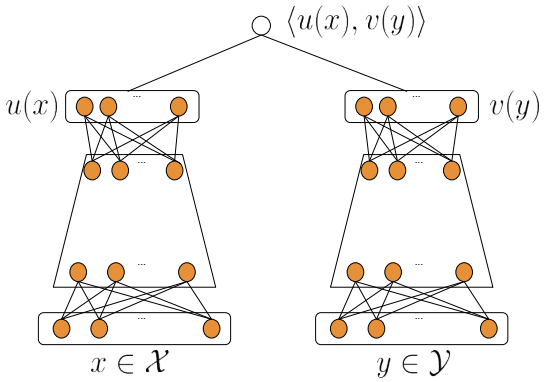

Figure 1: A two-tower DNN model for learning query and candidate representations.

two-tower DNNs in building retrieval models. Figure 1 provides an illustration of the two-tower model architecture where left and right towers encode $\{user, context\}$ and $\{item\}$ respectively. Two-tower DNN is generalized from the multi-class classification neural network [19], a multi-layer perceptron (MLP) model, where the right tower of Figure 1 is simplified to a single layer with item embeddings. As a result, the two-tower model architecture is capable of modeling the situation where label has structures or content features. MLP model is commonly trained with many sampled negatives from a fixed vocabulary of items. In contrast, with deep item tower, it is typically inefficient to sample and train on many negatives due to item content features and shared network parameters for computing all item embeddings.

We consider batch softmax optimization, where item probability is calculated over all items in a random batch, as a general recipe of training two-tower DNNs. However, as shown in our experiments, batch softmax is subject to sampling bias and could severely restrict the model performance without any correction. Importance sampling and the corresponding bias reduction have been studied in MLP model [4, 5]. Inspired by these works, we propose to correct sampling bias of batch softmax using estimated item frequency. In contrast to MLP model where the output item vocabulary is stationary, we target the streaming data situation with vocabulary and distribution changes over time. We propose a novel algorithm to sketch and estimate item frequency via gradient descent. In addition, we apply the bias-corrected modeling and scale it to build a personalized retrieval system for YouTube recommendations. We also introduce a sequential training strategy, designed to incorporate streaming data, along with the indexing and serving components of the system.

The major contributions of this paper include:

- **Streaming Frequency Estimation**. We propose a novel algorithm to estimate item frequency from a data stream, subject to vocabulary and distribution shifts. We offer analytical results to show the variance and bias of the estimation. We also provide simulation that demonstrates the efficacy of our approach in capturing data dynamics.
- **Modeling Framework**. We provide a generic modeling framework for building large-scale retrieval systems. In particular, we incorporate the estimated item frequency in a cross entropy loss for the batch softmax to reduce the sampling bias of in-batch items.
- **YouTube Recommendation**. We describe how we apply the modeling framework to build a large-scale retrieval system for YouTube recommendations. We introduce the end-to-end system including the training, indexing, and serving components.
- **Offline and Live Experiments**. We perform offline experiments on two real-world datasets and demonstrate the effectiveness of sampling bias correction. We also show that our retrieval system built for YouTube leads to improved engagement metrics in live experiments.

2 RELATED WORK

In this section, we give an overview of the related work, and highlight the connections to our contributions.

2.1 Content-Aware and Neural Recommenders

Utilizing content features of users and items is critical for improving generalization and mitigating cold-start problems. There is a line of research focusing on incorporating content features in the classic matrix factorization framework [23]. For instance, the generalized matrix factorization models, e.g., SVDFeature [8] and Factorization Machine [33], can be applied to incorporate item content features. These models are able to capture up to bi-linear, i.e., second-order, interactions between features. In recent years, deep neural networks (DNNs) have been shown effective in improving recommendation accuracy. Due to the nature of being highly nonlinear, DNNs offer a larger capacity for capturing complicated feature interactions [6, 35], compared to traditional factorization approaches. He et al. [21] directly applies the intuition of collaborative filtering (CF), and offers a neural CF (NCF) architecture for modeling user-item interactions. In the NCF framework, users and items embeddings are concatenated and passed through a multi-layer neural network to get the final prediction. Our work differs from NCF in two aspects: 1) we leverage a two-tower neural network for modeling user-item interactions so that the inference can be conducted over a large corpus of items in sub-linear time; 2) learning NCF relies on point-wise loss (such as squared or log loss), while we introduce a multi-class softmax loss and explicitly model item frequency.

On a separate line of work, deep and recurrent models such as LSTM are applied to incorporate temporal information and historical events in recommendations, e.g., [12, 14]. Besides the learning of separate user and item representations, there is another set of works focusing on designing neural networks particularly for learned to rank systems. Notably, multi-task learning has been a central technique in optimizing multiple objectives for complicated recommenders [27, 28]. Cheng et al. [9] introduces a wide-n-deep framework with jointly trained wide linear models and deep neural networks.

2.2 Extreme Classification

Softmax is one of the most commonly used functions in designing models for the prediction of a large output space up to millions of labels. Lots of research has been focusing on training softmax classification models for a large number of classes, ranging from language tasks [5, 29] to recommenders [12]. When the number of classes is extremely large, a widely used technique to speed up training is to sample a subset of classes. Bengio et al. [5] shows that a good sampling distribution should be adaptive to the model's output distribution. To avoid the complication of computing the sampling distribution, many real-world models apply a simple distribution such as unigram or uniform as a proxy. Recently, Blanc et al. [7] designs an efficient and adaptive kernel based sampling method. Despite the success of sampled softmax in various domains, it is not applicable to the case where label has content features. Adaptive sampling in this case also remains an open problem. Various works have shown that tree-based label structures, e.g., hierarchical softmax [30], are useful for building large-scale classification models while significantly reducing inference time. These approaches typically require a predefined tree structure based on certain categorical attributes. As a result, they are not suitable for incorporating a wide variety of input features.

2.3 Two-tower Models

Building neural networks with two towers has recently become a popular approach in several natural language tasks including modeling sentence similarities [31], response suggestions [24], and text-based information retrieval [17, 37]. Our work contributes to this line of research, particularly demonstrating the effectiveness of two-tower models in building large-scale recommenders. Compared to many language tasks in the aforementioned literature, it is worth noting that we focus on the problem with a much larger corpus size, which is common in our target applications such as YouTube. Through live experiments, we find that explicitly modeling item frequency is critical for improving retrieval accuracy in this setting. Yet, this problem is not well addressed in existing works.

3 MODELING FRAMEWORK

We consider a common setup for recommendation problems where we have a set of queries and items. Queries and items are representsed by feature vectors $\{x_i\}_{i=1}^N$ and $\{y_j\}_{j=1}^M$ respectively. Here $x_i \in \mathcal{X}, y_j \in \mathcal{Y}$ are both mixtures of a wide variety of features (e.g., sparse IDs and dense features) and could be in a very high dimensional space. The goal is to retrieve a subset of items given a query. In personalization scenario, we assume user and context are fully captured in x_i. Note that we begin with a finite number of queries and items to explain the intuition. Our modeling framework works without such an assumption.

We aim to build a model with two parameterized embedding functions $u : \mathcal{X} \times \mathbb{R}^d \to \mathbb{R}^k$, $v : \mathcal{Y} \times \mathbb{R}^d \to \mathbb{R}^k$ that map model parameter $\theta \in \mathbb{R}^d$ and features of query and candidates to a k-dimensional embedding space. We focus on the case where u, v are represented by two deep neural networks as illustrated Figure 1. The output of the model is the inner product of two embeddings, namely,

$$s(x, y) = \langle u(x, \theta),\ v(y, \theta) \rangle.$$

The goal is to learn model parameter θ from a training dataset of T examples, denoted by

$$\mathcal{T} := \{(x_i, y_i, r_i)\}_{i=1}^T,$$

where (x_i, y_i) denotes the pair of query x_i and item y_i, and $r_i \in \mathbb{R}$ is the associated reward for each pair.

Intuitively, the retrieval problem can be treated as a multi-class classification problem with continuous rewards. In classification tasks where each label is equally important, $r_i = 1$ for all positive pairs. In recommenders, r_i can be extended to capture various degrees of user engagement with a certain candidate. For example, in news recommendations, r_i can be the time a user spent on a certain article. Given a query x, a common choice for the probability distribution of picking candidate y from M items $\{y_j\}_{j=1}^M$ is based on the softmax function, i.e.,

$$\mathcal{P}(y|x; \theta) = \frac{e^{s(x,y)}}{\sum_{j \in [M]} e^{s(x, y_j)}}. \tag{1}$$

By further incorporating rewards r_i, we consider the following weighted log-likelihood as the loss function

$$L_T(\theta) := -\frac{1}{T} \sum_{i \in [T]} r_i \cdot \log(\mathcal{P}(y_i|x_i; \theta)). \tag{2}$$

When M is very large, it is not feasible to include all candidate examples in computing the partition function, i.e., the denominator in Equation (1). A common idea is to use a subset of items in constructing the partition function. We focus on dealing with streaming data. As a result, unlike training MLP models where negatives are sampled from a fixed corpus, we consider only using in-batch items [22] as negatives for all queries from the same batch. Precisely, given a mini-batch of B pairs $\{(x_i, y_i, r_i)\}_{i=1}^B$, for each $i \in [B]$, the batch softmax is

$$\mathcal{P}_B(y_i|x_i; \theta) = \frac{e^{s(x_i, y_i)}}{\sum_{j \in [B]} e^{s(x_i, y_j)}}. \tag{3}$$

In-batch items are normally sampled from a power-law distribution in our target applications. As a result, Equation (3) introduces a large bias towards full softmax: popular items are overly penalized as negatives due to the high probability of being included in a batch. Inspired by the logQ correction used in sampled softmax model [5], we correct each logit $s(x_i, y_j)$ by the following equation

$$s^c(x_i, y_j) = s(x_i, y_j) - \log(p_j).$$

Here p_j denotes the sampling probability of item j in a random batch. We defer the estimation of p_j to the next section.

With the correction, we have

$$\mathcal{P}_B^c(y_i|x_i; \theta) = \frac{e^{s^c(x_i, y_i)}}{e^{s^c(x_i, y_i)} + \sum_{j \in [B], j \neq i} e^{s^c(x_i, y_j)}}.$$

Then plugging the above term into Equation (2) leads to

$$L_B(\theta) := -\frac{1}{B} \sum_{i \in [B]} r_i \cdot \log(\mathcal{P}_B^c(y_i|x_i; \theta)), \tag{4}$$

which is the batch loss function. Running SGD with learning rate γ yields the model parameter update as

$$\theta \leftarrow \theta - \gamma \cdot \nabla L_B(\theta). \tag{5}$$

Algorithm 1 Training Algorithm

1: **Input:** Two parameterized embedding functions $u(\cdot, \theta), v(\cdot, \theta)$ where each one maps input features to an embedding space through a neural network. Learning rate γ (fixed or adaptive).
2: **repeat**
3: Sample or receive a batch of training data $\{(x_i, y_i, r_i)\}_{i=1}^{B}$ from a stream.
4: Obtain the estimated sampling probability p_i of each y_i from Algorithm 2.
5: Construct loss $L_B(\theta)$ according to (4).
6: $\theta \leftarrow \theta - \gamma \cdot \nabla L_B(\theta)$.
7: **until** stopping criterion

Algorithm 2 Streaming Frequency Estimation

1: **Input:** Learning rate α. Arrays A and B with size H. Hash function h with output space $[H]$.
2: *(Training)*
3: **For steps** $t = 1, 2, \ldots$.
4: Sample a batch of items $\mathcal{B}$. For each $y \in \mathcal{B}$, do

$$B[h(y)] \leftarrow (1-\alpha) \cdot B[h(y)] + \alpha \cdot (t - A[h(y)]).$$
$$A[h(y)] \leftarrow t.$$

5: **Until** stopping criterion
6: *(Inference)*
7: For any item y, sampling probability $\hat{p} = 1/B[h(y)]$.

Note that L_B does not require a fixed set of queries or candidates. Accordingly, Equation (5) can be applied to streaming training data whose distribution changes over time. See Algorithm 1 for a full description of our proposed approach.

Nearest Neighbor Search: Once the embedding functions u, v are learned, inference consists of two steps: 1) computing query embedding $u(x, \theta)$; 2) performing nearest neighbor search over a set of item embeddings that are pre-computed from embedding function v. Moreover, our modeling framework offers the option to choose an arbitrary set of items to serve at inference time. Instead of computing the dot product over all items to surface top items, low-latency retrieval is commonly based on a highly efficient similarity search system built on hashing techniques, e.g., [2, 10, 25], for approximate maximum inner product search (MIPS) problems. Specifically, compact representations of high dimensional embeddings are built through quantization [20] and end-to-end learning of coarse and product quantizers [36].

Normalization and Temperature. Empirically, we find that adding embedding normalization, i.e., $u(x, \theta) \leftarrow u(x, \theta)/\|u(x, \theta)\|_2$, $v(y, \theta) \leftarrow v(y, \theta)/\|v(y, \theta)\|_2$, improves model trainability and thus leads to better retrieval quality. In addition, a temperature τ is added to each logit to sharpen the predictions, namely,

$$s(x, y) = \langle u(x, \theta), v(y, \theta) \rangle / \tau.$$

In practice τ is a hyper-parameter tuned to maximize retrieval metrics such as recall or precision.

4 STREAMING FREQUENCY ESTIMATION

In this section, we elaborate the streaming frequency estimation used in Algorithm 1.

Consider a stream of random batches, where each batch contains a set of items. The problem is to estimate the probability of hitting each item y in a batch. A critical design criterion is to have a fully distributed estimation to support distributed training when there are multiple training jobs (i.e., workers).

In either single-machine or distributed training, a unique global step, which represents the number of data batches consumed by the trainer, is associated with each sampled batch. In a distributed setting, the global step is typically synchronized among multiple workers through parameter servers. We can leverage the global step and convert the estimation of frequency p of an item to the estimation of δ, which denotes the average number of steps between two consecutive hits of the item. For example, if one item gets sampled every 50 steps, then we have $p = 0.02$. The use of global step offer us two advantages: 1) Multiple workers are implicitly synchronized in frequency estimation via reading and modifying the global step; 2) Estimating δ can be achieved by a simple moving average update, which is adaptive to distribution change.

As using fixed item vocabulary is not practical, we apply hash arrays to record the sampling information of streaming IDs. Note that introducing hashing could cause potential hash collision. We will revisit this issue and propose an improved algorithm at the end of this section. As shown in Algorithm 2, we keep two arrays A and B with size H. Suppose h is a hash function that maps any item to an integer in $[H]$, the mapping can be based on the ID or any other thumbnail feature values. Then for a given item y, $A[h(y)]$ records the latest step when y is sampled, and $B[h(y)]$ contains the estimated δ of y. We use array A to help updating array B. Once item y occurs in step t, we update array B by

$$B[h(y)] \leftarrow (1-\alpha) \cdot B[h(y)] + \alpha \cdot (t - A[h(y)]). \tag{6}$$

After B is updated, we assign t to $A[h(y)]$.

For each item, suppose the number of steps between two consecutive hits follows a distribution represented by random variable Δ with mean $\delta = \mathbb{E}(\Delta)$. Here our goal is to estimate δ from a stream of samples. Whenever an item occurs in a batch at step t, $t - A[v(y)]$ is a new sample of Δ. Accordingly, the above update can be understood as running SGD with fixed learning rate α to learn the mean of this random variable. Formally, in the case of no collision, the next result shows the bias and variance of this online estimation.

Proposition 4.1. *Suppose $\{\Delta_1, \Delta_2, ..., \Delta_t\}$ is a sequence of i.i.d. samples of random variable Δ. Let $\delta = \mathbb{E}[\Delta]$. Consider an online estimation where for $i \in [t]$ and $\alpha \in (0, 1)$,*

$$\delta_i = (1-\alpha) \cdot \delta_{i-1} + \alpha \cdot \Delta_i.$$

The estimation bias is given by

$$\mathbb{E}(\delta_t) - \delta = (1-\alpha)^t \delta_0 - (1-\alpha)^{t-1} \delta. \tag{7}$$

For the variance, we have

$$\mathbb{E}\left[(\delta_t - \mathbb{E}[\delta_t])^2\right] \le (1-\alpha)^{2t}(\delta_0 - \delta)^2 + \alpha \cdot \mathbb{E}\left[(\Delta_1 - \alpha)^2\right]. \tag{8}$$

Proof. By taking expectation, we have

$$\mathbb{E}[\delta_i] = (1-\alpha) \cdot \mathbb{E}(\delta_{i-1}) + \alpha \cdot \delta.$$

By induction on t, we have (7). For the variance, we have

$$\begin{aligned}&\mathbb{E}\left[(\delta_t - \mathbb{E}[\delta_t])^2\right]\\&= \mathbb{E}\left[(\delta_t-\delta)^2\right] + 2\cdot\mathbb{E}\left[(\delta_t-\delta)(\delta-\mathbb{E}[\delta_t])\right] + (\mathbb{E}[\delta_t]-\delta)^2\\&= \mathbb{E}\left[(\delta_t-\delta)^2\right] - (\mathbb{E}[\delta_t]-\delta)^2 \le \mathbb{E}\left[(\delta_t-\delta)^2\right].\end{aligned}$$

For the last term, we have

$$\begin{aligned}\mathbb{E}\left[(\delta_i-\delta)^2\right] &= (1-\alpha)^2\mathbb{E}\left[(\delta_{i-1}-\delta)^2\right] + \alpha^2\mathbb{E}(\Delta_t-\delta)^2\\&\quad + 2(1-\alpha)\alpha\mathbb{E}\left[(\delta_{i-1}-\delta)(\Delta_i-\delta)\right].\end{aligned}$$

As δ_{i-1} and Δ_i are independent, the last term is zero. Then by induction on i, we have

$$\begin{aligned}&\mathbb{E}\left[(\delta_t-\delta)^2\right] = (1-\alpha)^{2t}(\delta_0-\delta)^2 + \alpha^2\frac{1-(1-\alpha)^{2t-2}}{1-(1-\alpha)^2}\mathbb{E}\left[(\Delta_1-\delta)^2\right]\\&\le (1-\alpha)^{2t}(\delta_0-\delta)^2 + \alpha\mathbb{E}\left[(\Delta_1-\delta)^2\right].\end{aligned}$$

□

Equation (7) indicates that the bias $|\mathbb{E}[\delta_t]-\delta| \to 0$ as $t\to\infty$. Moreover, an ideal initialization $\delta_0 = \delta/(1-\alpha)$ can lead to an unbiased estimation in every step. Equation (8) gives an upper bound on the estimation variance. The learning rate α affects the variance in two folds: 1) higher rate causes a faster decrease of the first term that depends on initialization error; 2) lower rate reduces the second term which depends on the variance of Δ and does not decrease over time.

To get the estimated sampling probability $\hat{p}$ of y, we can simply perform $\hat{p} = 1/B[h(y)]$.

Distributed Updates. We consider the distributed training framework presented in [13], where model parameters are distributed over a set of servers called parameter servers, and multiple workers process training data independently and communicate with parameter servers to fetch and update model parameters. Algorithm 2 can be extended to this setting. Arrays A, B and the global step parameter are distributed over parameter servers. Each worker executes line 4 via sampling a mini-batch of items. In detail, at step t, $A[h(y)]$, $B[h(y)]$ are fetched from parameter servers. Then $A[h(y)]$, $B[h(y)]$ are updated as shown and sent back. Therefore, the updates in Algorithm 2 can be executed along with the asynchronous stochastic gradient descent training of neural networks.

Multiple Hashings. Inspired by a similar idea in count-min sketch [11], we extend Algorithm 2 to leverage multiple hash functions to mitigate over-estimation of item frequency due to collisions. The improved estimation is presented in Algorithm 3. Updating each array A_i, B_i follows the corresponding steps in Algorithm 2. Each bucket in B could be an under-prediction of the true step gap as it could represent the union of multiple items because of collisions. As a result, for inference, we take the maximum of our m estimations representing the number of steps between two consecutive hits.

Algorithm 3 Improved Frequency Estimation with Multiple Arrays

1: **Input:** Learning rate α. A set of m arrays $\{A\}_{i=1}^m$ and $\{B\}_{i=1}^m$ with size H. A set of independent hash functions $\{h\}_{i=1}^m$ with output space $[H]$.
2: *(Training)*
3: **For steps** $t = 1, 2, \ldots$.
4: Sample a batch of items $\mathcal{B}$. For each $y \in \mathcal{B}$, and $i \in [m]$,
$B_i[h(y)] \leftarrow (1-\alpha)\cdot B_i[h(y)] + \alpha\cdot(t - A_i[h(y)])$.
$A_i[h(y)] \leftarrow t$.
5: **Until** stopping criterion
6: *(Inference)*
7: For any item y, estimated probability $\hat{p} = 1/\max_i\{B_i[h(y)]\}$.

5 NEURAL RETRIEVAL SYSTEM FOR YOUTUBE

We apply the proposed modeling framework and scale it to build a large scale neural retrieval system for one particular product in YouTube. This product generates video recommendations conditioned on a video (called seed video) being watched by a user. The recommendation system consists of two stages: nomination (a.k.a., retrieval) and ranking. At nomination stage, we have multiple nominators that each generates hundreds of video recommendations given constraints of a user and a seed video. These videos are subsequently scored and re-ranked by a fully-blown neural network ranking model. In this section, we focus on building an additional nominator in the retrieval stage, especially on the perspectives of data, model architecture, training, and serving.

5.1 Modeling Overview

The YouTube neural retrieval model we built consists of query and candidate networks. Figure 2 illustrates the general model architecture. At any point of time, the video which a user is watching, i.e., the seed video, provides a strong signal about the user's current interest. As a result, we make use of a large set of seed video features along with the user's watch history. The candidate tower is built to learn from candidate video features.

Training Label. Video clicks are used as positive labels. In addition, for each click, we construct a reward r_i to reflect different degrees of user engagement with the video. For example, $r_i = 0$ for clicked videos with little watch time. On the other hand, $r_i = 1$ indicates the whole video got watched. The reward is used as example weight as shown in Equation (4).

Video Features. The video features we use include both categorical and dense features. Examples of categorical features include *Video Id*, and *Channel Id*. For each of these entities, an embedding layer is created to map each categorical feature to a dense vector. Normally we are dealing with two kinds of categorical features. Some features (e.g., *Video id*) have strictly one categorical value per video, so we have one embedding vector representing that. Alternatively, one feature (e.g., *Video topics*) might be a sparse vector of categorical values, and the final embedding representing that feature would be a weighted sum of the embeddings for each of the values in the sparse vector. To handle out-of-vocabulary entities, we randomly assign them to a fixed set of hash buckets, and learn an embedding for each one. Hash buckets are important for model to capture new entities available in YouTube, particularly when sequential training described in Section 5.2 is used.

User Features. We use a user's watch history to capture the user's interest besides the seed video. One example is a sequence of k video ids the user recently watched. We treat the watch history as a bag of words (BOW), and represent it by the average of video

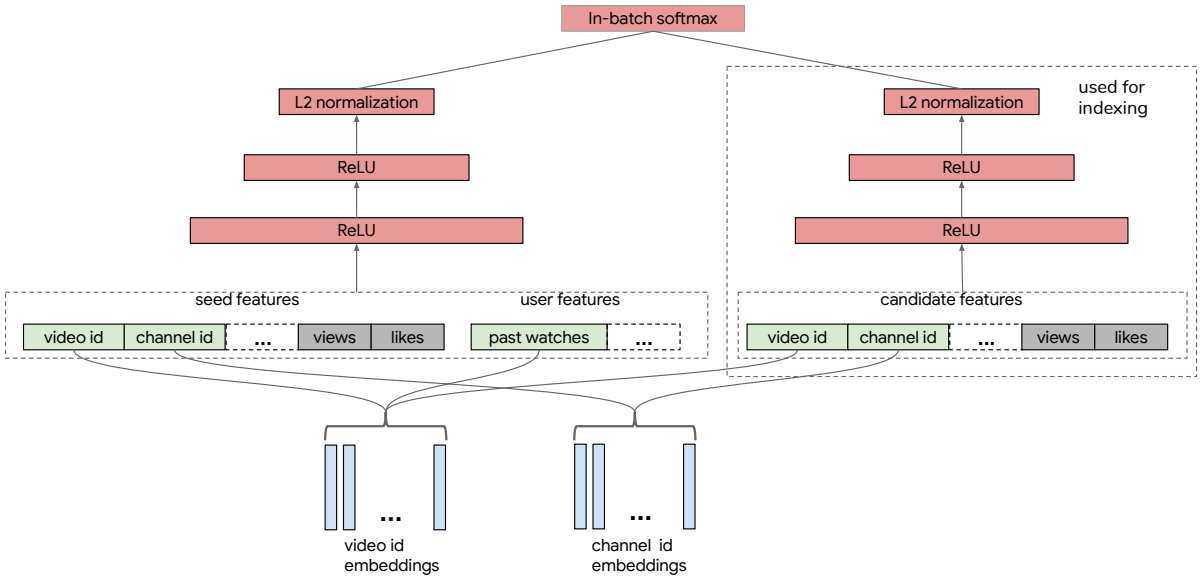

Figure 2: Illustration of the Neural Retrieval Model for YouTube.

id embeddings. In the query tower, user and seed video features are fused in the input layer, which is then passed through a feed forward neural network.

For the same type of IDs, embeddings are shared among the related features. For example, the same set of video id embeddings is used for seed, candidate and users' past watches. We did experiment with non-shared embeddings, but did not observe significant model quality improvement.

5.2 Sequential Training

Our model is implemented in Tensorflow [1], and trained with distributed gradient descent over many workers and parameter servers. In YouTube, new training data is generated every day, and training datasets are accordingly organized by days. The model training makes use of this sequential structure in the following way. Trainer consumes the data sequentially from the oldest training examples to the most recent training examples. Once the trainer has caught up to the latest day of training data, it waits for the next day's training data to arrive. In this way, the model is able to keep up with latest data distribution shift. Training data is essentially consumed by trainer in a streaming manner. We apply Algorithm 2 (or Algorithm 3 if multiple hashings are used) to estimate item frequency. The online update of Equation (6) enables the model to adapt to new frequency distribution.

5.3 Indexing and Model Serving

The index pipeline in the retrieval system periodically creates a Tensorflow SavedModel for online serving. The index pipeline was built in three stages: candidate example generation, embedding inference, and embedding indexing, as shown in Figure 3. In the first stage, a set of videos are selected from YouTube corpus based on certain criterion. Their features are fetched and added to the candidate examples. In the second stage, right tower of Figure 2 is applied to compute embeddings from candidate examples. In the third stage, we train a Tensorflow-based embedding index model based on tree and quantizied hashing techniques. We gloss over the details as they are not the focus of this paper. Finally, the SavedModel used in serving is created by stitching the query tower of Figure 2 and the indexing model together.

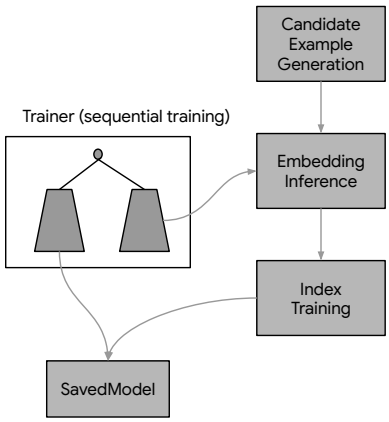

Figure 3: Overview of the index pipeline for YouTube neural retrieval system.

6 EXPERIMENTS

In this section, we show experimental results to demonstrate the effectiveness of the proposed item frequency estimation and modeling framework.

6.1 Simulation on Frequency Estimation

To evaluate effectiveness of Algorithms 2 & 3, we begin with a simulation study where we first apply each proposed algorithm

to fit a fixed item distribution, and then change the distribution after a certain step. To be more precise, in our setting, we use a fixed set of M items and each item is sampled independently according to probabilities $q_i \propto i^2$ for $i \in [M]$ where $\sum_i q_i = 1$. We conduct the simulation on an input stream where a batch of items $\mathcal{B}$ are sampled in each step. Here each item in $\mathcal{B}$ is sampled without replacement from q_i. Therefore, the item sampling probability we aim to fit is $p_i = |\mathcal{B}| \times q_i$. We keep the sampling distribution static for the first t steps. We then switch it to $q_i \propto (M-1-i)^2$ for the remaining steps. To evaluate the estimation accuracy, we use a rescaled L_1 distance between the estimated probability set $\{\hat{p}_i\}_{i\in[M]}$ and $\{p_i\}_{i\in[M]}$, precisely $\frac{1}{2|\mathcal{B}|}\sum_i |\hat{p}_i - p_i|$, as the estimation error. It can be also understood as the total variance between the estimates $\{\hat{q}_i\}_{i\in[M]}$ and $\{q_i\}_{i\in[M]}$.

Specifically, we report: 1) effect of learning rate α, and 2) effect of multiple hashings.

Effect of Learning Rate α. We set $M = 1000, B = 128$, and use array size $H = 5000$ for both A and B. In addition, we initialize array A with all zeros and every entry of B with value 100. Distribution is switched at step $t = 10000$. We use one hash function and run Algorithm 2. Figure 4 shows the estimation errors over the number of global steps for a set of learning rates α. We observe that all three curves converge to a level of error which comes from hashing collisions and estimation variance. With a higher learning rate, the algorithm is more adaptive to distribution change, but the final variance is higher as shown in Proposition 4.1.

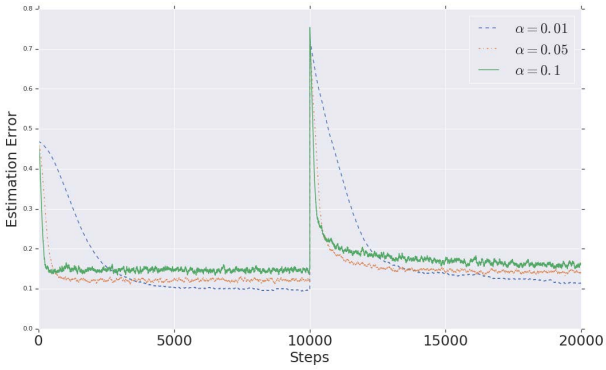

Figure 4: Frequency estimation errors for various learning rates α. Item distribution is switched at step 10000.

Effect of Multiple Hashings. For the second simulation, we run Algorithm 3 and experiment with various number of hash functions m. Figure 5 shows the curves of estimation error for $m = 1, 2, 4$. We choose different array sizes H for A, B so that the total number of hash buckets remain the same across these three settings. Figure 5 demonstrates the effectiveness of using multiple hash functions to reduce estimation error, even under same number of parameters.

6.2 Wikipedia Page Retrieval

In this section, we conduct page-retrieval experiments on the Wikipedia dataset [16] to show the efficacy of the sampling-bias-corrected batch loss 4.

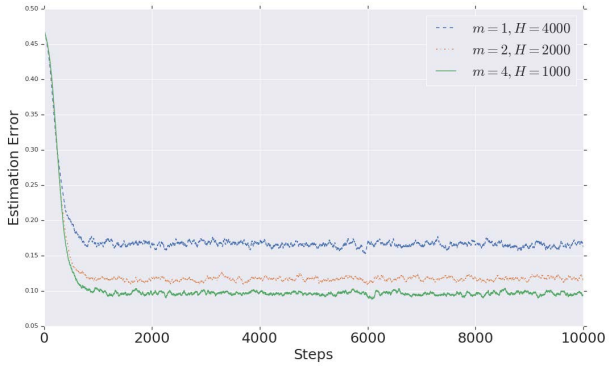

Figure 5: Frequency estimation errors for various number of hash functions m.

Dataset. We consider the task of predicting intra-site links between Wikipedia pages. For a given pair of source and destination pages (x, y), label is 1 if there is a link from x to y, and 0 otherwise. Each page is represented by a set of content features including page URL, a bag-of-words representation of the set of n-grams in the title, and a bag-of-words representation of the page categories. We experiment with the English graph, which consists of $5.3M$ pages, $430M$ links, $510K$ title n-grams, and $403.4K$ unique categories.

Model. We treat the link prediction as a retrieval problem, where given a source page, the task is to retrieve the destination pages from the page corpus. As a result, we train a two-tower neural network where the left and right towers map the features of source and destination features respectively. The input feature embeddings are shared among the two towers. Each tower has two fully connected ReLU layers with dimensions [512, 128].

Baselines. We compare the proposed sampling-bias-corrected batch softmax (*correct-sfx*) with batch softmax without any correction (*plain-sfx*) as shown in Equation (3), to demonstrate the efficacy of bias correction. In addition, we consider the the mean squared loss which is widely adopted in modeling implicit feedback in recommendations. The loss is a combination of MSE on observed pairs and a regularization term pushing all unseen pairs to a constant prior commonly chosen as 0. Under the framework introduced in Section 3, the loss is

$$\frac{1}{|\Omega|}\sum_{(x_i,y_i)\in\Omega}(\langle u(x_i),\, v(y_i)\rangle - r_i)^2 + \lambda\cdot\frac{1}{|\Omega^c|}\sum_{(x_i,y_i)\in\Omega^c}\langle u(x_i),\, v(y_i)\rangle^2,$$

where Ω and Ω^c denote the set of observed pairs and its complement, and λ is a positive hyperparameter. In matrix factorization, such a loss is typically trained using alternating least squares [23] or coordinate descent methods [3] by writing the regularization term as a matrix-inner-product of two Gramians, which can be computed efficiently for linear embeddings. Very recently, Krichene et al. [26] extends the Gramian computation to non-linear scenarios by SGD estimation. We refer to this method as *mse-gramian*.

Training and Evaluation. For all methods, we use batch size 1024 and the model is trained with Adagrad [15] and learning rate 0.01 for $10M$ steps. For frequency estimation, we use $m = 1$, $H = 40M$ and $\alpha = 0.01$. We holdout 10% links for evaluation. We evaluate

the model performance by Recall@K, which is essentially the average probability of including true label in top k candidates against the full page corpus. Parameter λ in *mse-gramian* is tuned via line search and we report the best results here. We find that normalizing the output layers always improves model performance and training stability. We only show the results with normalization. We experiment with multiple temperature values τ for *plain-sfx* and *correct-sfx*.

Results are summarized in Table 1. For each temperature value, *correct-sfx* outperforms the corresponding *plain-sfx* by a large margin. It is interesting to see the effect of temperature on performance, suggesting the necessity to carefully tune this parameter when normalization is applied. We also note that batch softmax based methods lead to better performance than *mse-gramian*.

Method	Recall@10	Recall@50	Recall@100	Recall@300
mse-gramian [26]	0.0432	0.1326	0.2027	0.3530
plain-sfx $\tau = 0.05$	0.0579	0.2259	0.3573	0.5931
plain-sfx $\tau = 0.07$	0.0643	0.2423	0.3746	0.5991
plain-sfx $\tau = 0.14$	0.0614	0.2216	0.3341	0.5200
correct-sfx $\tau = 0.05$	0.0987	**0.3202**	**0.4835**	**0.7413**
correct-sfx $\tau = 0.07$	**0.1065**	0.3079	0.4664	0.7234
correct-sfx $\tau = 0.14$	0.0807	0.2411	0.3519	0.5529

Table 1: Recall@K for retrieving destination pages from the 5.3M page corpus in Wikipedia.

6.3 YouTube Experiments

We carry out offline and live experiments on YouTube based on the neural retrieval system introduced in Section 5. The YouTube training data we use includes billions of clicked videos on a daily basis, across many days.

Setup. Recall that the model structure we use is shown in Figure 2. As aforementioned, input feature embeddings are shared between query and candidate towers if they are available to both. We use three-layer DNNs with hidden layer sizes [1024, 512, 128] for both towers. We train the model using Adagrad, learning rate 0.2, and batch size 8192. For frequency estimation, we set $H = 50M, m = 1, \alpha = 0.01$. Recall that we apply sequential training as introduced in Section 5.2. For experiments in this section, index of approximately $10M$ videos chosen from the YouTube corpus is built periodically every few hours. The index corpus is subject to change over time due to, for example, uploads of fresh videos. But it typically covers more than 90% of training examples.

Offline Experiments. We assign $r_i = 1$ for all clicked videos, and evaluate the model performance by the recall when retrieving clicked videos. We simplify the reward function for offline experiments because it is not obvious to define an appropriate offline metric for a continuous reward. To incorporate the sequential training, we evaluate the model performance after day d_0 when the trainer completes the catch-up phase, which is set to 15 days, and starts to wait for new data. For each new day after d_0, we holdout 10% data for evaluation. To account for a weekly pattern, we report averaged offline results over 7 days, i.e., from $d_0 + 1$ to $d_0 + 7$. The results are presented in Table 2. Again, we see significant improvements from using batch softmax compared to *mse-gramian*. Also, among the settings with different τ, item-frequency-corrected softmax performs significantly better than the plain softmax.

Method	Recall@5	Recall@10	Recall@30	Recall@50
mse-gramian [26]	0.0554	0.0768	0.1149	0.1338
plain-sfx $\tau = 0.1$	0.1916	0.2512	0.3658	0.4246
plain-sfx $\tau = 0.025$	0.1958	0.2609	0.3839	0.4456
plain-sfx $\tau = 0.05$	0.2069	0.2728	0.3964	0.4586
correct-sfx $\tau = 0.1$	0.1957	0.2689	0.4125	0.4796
correct-sfx $\tau = 0.025$	0.2014	0.2790	0.4314	0.5082
correct-sfx $\tau = 0.05$	**0.2150**	**0.2960**	**0.4537**	**0.5322**

Table 2: Recall@K for retrieving clicked videos pages from a 10M video corpus in YouTube. Reward r_i is set to be 1 for all clicked videos in training.

Live Experiments. We also conduct live experiments in an A/B testing framework in YouTube. For users in the control group, videos are suggested from the production system. For the treatment group, users are presented with recommendations after adding candidates from the neural retrieval system shown in Figure 2 to the nomination stage. As recommending videos which users like to click is not ideal, for live experiments we train the model with the reward in a way to reflect the users' real engagement with clicked videos. We report the engagement metric aligned with this label. Results are shown in Table 3. As can be seen, adding the neural retrieval system improves the previous production system by a significant margin. Moreover, the model with *correct-sfx* performs better than the baseline of using *plain-sfx* by a significant margin, demonstrating the effectiveness of sampling bias correction.

Method	Engagement Metric Improvement
plain-sfx $\tau = 0.05$	+0.20%
correct-sfx $\tau = 0.05$	+0.37%

Table 3: YouTube live experiment results. Reward r_i of clicked video is set to be some user feedback related to the engagement metric reported.

7 CONCLUSION

In this paper, we presented a generic modeling framework for building large scale content-aware retrieval models for industrial-scale applications. We proposed a novel algorithm for estimating item frequency. Theoretical analyses and simulation demonstrated its correctness and effectiveness. We applied the proposed modeling framework to build a neural retrieval system for YouTube recommendations. In particular, to capture the data dynamics of YouTube, we presented a sequential training strategy to which the streaming frequency estimation algorithm could be easily integrated. Offline experiments on both Wikipedia link prediction and YouTube video retrieval showed significant improvements using sampling bias correction. Live experiments in YouTube also showed improvements in user engagement with candidates retrieved from our neural retrieval system.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM* 51, 1 (Jan. 2008), 117–122. https://doi.org/10.1145/1327452.1327494

[3] Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. 2017. A Generic Coordinate Descent Framework for Learning from Implicit Feedback. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. 1341–1350.

[4] Yoshua Bengio and Jean-Sébastien Sénécal. 2003. Quick Training of Probabilistic Neural Nets by Importance Sampling. In *Proceedings of the conference on Artificial Intelligence and Statistics (AISTATS)*.

[5] Y. Bengio and J. S. Senecal. 2008. Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model. *Trans. Neur. Netw.* 19, 4 (April 2008), 713–722. https://doi.org/10.1109/TNN.2007.912312

[6] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H. Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. ACM, New York, NY, USA, 46–54. https://doi.org/10.1145/3159652.3159727

[7] Guy Blanc and Steffen Rendle. 2018. Adaptive Sampled Softmax with Kernel Based Sampling. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 589–598. http://proceedings.mlr.press/v80/blanc18a.html

[8] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. 2012. SVDFeature: A Toolkit for Feature-based Collaborative Filtering. *J. Mach. Learn. Res.* 13, 1 (Dec. 2012), 3619–3622. http://dl.acm.org/citation.cfm?id=2503308.2503357

[9] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide Deep Learning for Recommender Systems. *arXiv:1606.07792* (2016). http://arxiv.org/abs/1606.07792

[10] Edith Cohen and David D. Lewis. 1997. Approximating Matrix Multiplication for Pattern Recognition Tasks. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 682–691. http://dl.acm.org/citation.cfm?id=314161.314415

[11] Graham Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *J. Algorithms* 55, 1 (April 2005), 58–75. https://doi.org/10.1016/j.jalgor.2003.12.001

[12] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA.

[13] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *NIPS*.

[14] 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 (RecSys '17)*. ACM, New York, NY, USA, 152–160. https://doi.org/10.1145/3109859.3109877

[15] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 12 (July 2011), 2121–2159. http://dl.acm.org/citation.cfm?id=1953048.2021068

[16] Wikimedia Foundation. [n.d.]. Wikimedia Downloads. https://dumps.wikimedia.org/

[17] Daniel Gillick, Alessandro Presta, and Gaurav Singh Tomar. 2018. End-to-End Retrieval in Continuous Space. *CoRR* abs/1811.08008 (2018). arXiv:1811.08008 http://arxiv.org/abs/1811.08008

[18] Carlos A. Gomez-Uribe and Neil Hunt. 2015. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article 13 (Dec. 2015), 19 pages. https://doi.org/10.1145/2843948

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[20] Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. 2016. Quantization based Fast Inner Product Search. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Arthur Gretton and Christian C. Robert (Eds.), Vol. 51. PMLR, Cadiz, Spain, 482–490. http://proceedings.mlr.press/v51/guo16a.html

[21] 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 (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 173–182. https://doi.org/10.1145/3038912.3052569

[22] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *The International Conference on Learning Representations (ICLR 2016)*.

[23] Y. Hu, Y. Koren, and C. Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *2008 Eighth IEEE International Conference on Data Mining*. 263–272. https://doi.org/10.1109/ICDM.2008.22

[24] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufman, Balint Miklos, Greg Corrado, Andrew Tomkins, Laszlo Lukacs, Marina Ganea, Peter Young, and Vivek Ramavajjala. 2016. Smart Reply: Automated Response Suggestion for Email. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) (2016)*. https://arxiv.org/pdf/1606.04870v1.pdf

[25] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient Retrieval of Recommendations in a Matrix Factorization Framework. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12)*. ACM, New York, NY, USA, 535–544. https://doi.org/10.1145/2396761.2396831

[26] Walid Krichene, Nicolas Mayoraz, Steffen Rendle, Li Zhang, Xinyang Yi, Lichan Hong, Ed Chi, and John Anderson. 2019. Efficient Training on Very Large Corpora via Gramian Estimation. In *7th International Conference on Learning Representations*.

[27] Jiaqi Ma, Zhe Zhao, Jilin Chen, Ang Li, Lichan Hong, and Ed H. Chi. 2019. SNR: Sub-Network Routing for Flexible Parameter Sharing in Multi-task Learning. In *AAAI 2019*. http://www.jiaqima.com/papers/SNR.pdf

[28] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. 2018. Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. ACM, New York, NY, USA, 1930–1939. https://doi.org/10.1145/3219819.3220007

[29] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'13)*. Curran Associates Inc., USA, 3111–3119. http://dl.acm.org/citation.cfm?id=2999792.2999959

[30] Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *AISTATS'05*. 246–252.

[31] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016. Learning Text Similarity with Siamese Recurrent Networks. In *Rep4NLP@ACL*.

[32] The Netflix Prize. 2012. The Netflix Prize. http://www.netflixprize.com/.

[33] S. Rendle. 2010. Factorization Machines. In *2010 IEEE International Conference on Data Mining*. 995–1000. https://doi.org/10.1109/ICDM.2010.127

[34] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. 2017. DropoutNet: Addressing Cold Start in Recommender Systems. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4957–4966. http://papers.nips.cc/paper/7081-dropoutnet-addressing-cold-start-in-recommender-systems.pdf

[35] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17 (ADKDD'17)*. ACM, New York, NY, USA, Article 12, 7 pages. https://doi.org/10.1145/3124749.3124754

[36] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N Holtmann-Rice, David Simcha, and Felix X Yu. 2017. Multiscale Quantization for Fast Similarity Search. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5749–5757. http://papers.nips.cc/paper/7157-multiscale-quantization-for-fast-similarity-search.pdf

[37] Yinfei Yang, Steve Yuan, Daniel Cer, Sheng-Yi Kong, Noah Constant, Petr Pilar, Heming Ge, Yun-hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Learning Semantic Textual Similarity from Conversations. In *Proceedings of The Third Workshop on Representation Learning for NLP*. Association for Computational Linguistics, Melbourne, Australia, 164–174. https://www.aclweb.org/anthology/W18-3022

[38] 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 (KDD '18)*. ACM, New York, NY, USA, 1079–1088. https://doi.org/10.1145/3219819.3219826