Mixed Negative Sampling for Learning Two-tower Neural Networks in Recommendations

Ji Yang
Google
jiyangjy@google.com

Xinyang Yi
Google
xinyang@google.com

Derek Zhiyuan Cheng
Google
zcheng@google.com

Lichan Hong
Google
lichan@google.com

Yang Li
Google
ngli@google.com

Simon Xiaoming Wang
Google
wxm@google.com

Taibai Xu
Google
taibaixu@google.com

Ed H. Chi
Google
edchi@google.com

ABSTRACT

Learning query and item representations is important for building large scale recommendation systems. In many real applications where there is a huge catalog of items to recommend, the problem of efficiently retrieving top k items given user's query from deep corpus leads to a family of factorized modeling approaches where queries and items are jointly embedded into a low-dimensional space. In this paper, we first showcase how to apply a two-tower neural network framework, which is also known as dual encoder in the natural language community, to improve a large-scale, production app recommendation system. Furthermore, we offer a novel negative sampling approach called Mixed Negative Sampling (MNS). In particular, different from commonly used batch or unigram sampling methods, MNS uses a mixture of batch and uniformly sampled negatives to tackle the selection bias of implicit user feedback. We conduct extensive offline experiments using large-scale production dataset and show that MNS outperforms other baseline sampling methods. We also conduct online A/B testing and demonstrate that the two-tower retrieval model based on MNS significantly improves retrieval quality by encouraging more high-quality app installs.

KEYWORDS

Information Retrieval, Neural Networks, Context-aware Recommender Systems, Extreme Classification

ACM Reference Format:
Ji Yang, Xinyang Yi, Derek Zhiyuan Cheng, Lichan Hong, Yang Li, Simon Xiaoming Wang, Taibai Xu, and Ed H. Chi. 2020. Mixed Negative Sampling for Learning Two-tower Neural Networks in Recommendations. In *Companion Proceedings of the Web Conference 2020 (WWW '20 Companion), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3366424.3386195

WWW '20 Companion, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7024-0/20/04.
https://doi.org/10.1145/3366424.3386195

1 INTRODUCTION

Recommendation systems are important in connecting users to a large number of relevant items and content. One of the most critical challenges in building real-world recommenders is to accurately score millions to billions of items in real time. Many industry-scale systems [7, 9] have a two-stage architecture where a retrieval model first retrieves a small fraction of relevant items from item corpus, and a ranking model is applied to re-ranks the retrieved items based on users' feedback such as clicks or ratings on impressions. In this paper, we focus on the retrieval problem and showcase how we improve the app retrieval system of Google Play, one of the largest commercial mobile app stores, by jointly learning query and app representations via deep neural networks.

Recently, lots of research has been developed on embedding-based retrieval models. Matrix factorization (MF) (e.g., [14]) is one of most popular approaches for learning query and item latent factors in building retrieval systems. One challenge of MF is cold-start, i.e., it's hard for this method to generalize to items that have no user interaction. A body of recommendation research (e.g., [4, 13, 21]) addresses this challenge by further leveraging item's content features, which can be loosely defined as a wide variety of features describing items beyond their ids. For instance, content features of an app could be text descriptions, creators, categories, etc. As deep learning has demonstrated tremendous successes in computer vision and natural language processing, there are many works [3, 7] that apply extreme multi-class classification model to learn query embedding via multi-layer perceptions. Despite the non-linearity on the query side, each item is still represented by a single embedding. Hence, similar to MF, this method fails to represent a collection of items features with various formats.

Most recently, two-tower neural networks, with towers referring to encoders based on deep neural network (DNN), attains growing interests [24], and are applied to tackle the challenge of cold-start issue of MF and multi-class extreme classification models. The basic idea is to further incorporate items' content features through a multi-layer neural network that would generalize to fresh or tail items with no training data. See Figure 1 for an illustration of the model architecture. This model framework is also closely connected to the dual encoder framework [8, 23] in language models. This paper lies in this line of work. We focus on applying the two-tower

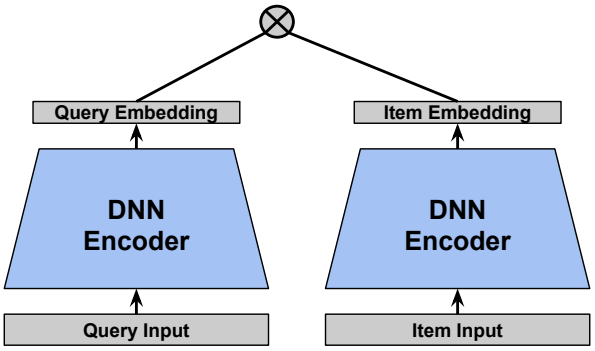

Figure 1: Two-tower Neural Network.

framework to improve the retrieval system of Google Play, which is one of the largest commercial mobile app stores, connecting millions of apps to billions of users across the world.

Similar to language tasks, negative sampling plays a critical role in training two-tower neural networks in recommendations. Especially, in recommendation, users' positive feedback are often collected, and counterfactuals on items not shown are very hard to obtain. A popular sampling approach [1, 7] for fitting a softmax output distribution is to sample according to the unigram distribution of items. The work in [24] extends unigram sampling to the two-tower setting by using batch negatives, i.e., using the positive items in a mini batch as shared negatives for all queries in the same batch. We note that unigram-sampled or batch negatives have the limit of selection bias in training data. This is because the training data is derived from user feedback logs, and users often interact with a small set of popular items suggested by existing recommender system. Items that are not favored by the existing system are less likely to get user feedback. Accordingly, sampling batch negatives only from training data will end up with a model lacking resolution for long-tail apps, which seldom appear in the training data.

Inspired by the aforementioned constraint of batch negatives, we propose a novel sampling approach called Mixed Negative Sampling (MNS), where the idea is to use a mixture of unigram and uniform distributions. In particular, in addition to the negatives sampled from batch training data, we uniformly sample negatives from the candidate corpus to serve as additional negatives. This two-stream negative sampling enables us to: (1) reduce selection bias by bringing in samples from the entire candidate corpus; (2) adjust the sampling distribution by changing the number of additional negative samples from the corpus. We further demonstrate the effectiveness of our retrieval system with both offline and online experiments on Google Play. Offline studies showed that MNS significantly improves retrieval quality. In addition, online A/B testing shows that the two-tower model trained with MNS leads to more high-quality app installs from for Google Play. The lessons from this case study sheds light for other large-scale recommendation systems dealing with huge item catalogs.

In summary, our contributions are:

- **Real-world application.** We showcase how to apply the dual-encoder framework to improve a large scale, production app recommendation system. In particular, we show how to leverage the item tower to mitigate the well-known cold-start problem of embedding-based approaches.
- **Mixed negative sampling.** We present the problem of selection bias in the commonly used unigram and batch negative sampling methods, and and propose a novel negative sampling approach called mixed negative sampling.
- **Offline and online experiments.** We conduct extensive offline and online experiments in Google Play to demonstrate the effectiveness of MNS, and report that the new system we build can encourage significantly more high-quality app installs from users.

2 RELATED WORK

In the decade, deep learning has demonstrated tremendous successes in recommender systems, ranging from video recommendation [7], news recommendation [20], to visual discovery in social networks [15, 25]. Cheng et al. [5] introduces a wide-n-deep framework for reranking task in app recommendations. For the retrieval task, there has also been growing interests in applying DNN-based representation learning approaches. Covington et al. [7] treats the retrieval task as an extreme multi-class classification trained with multi-layer perceptron (MLP) [10] model using sampled softmax as its loss function. Despite achieving lots of success, such a model architecture relies on a predetermined item vocabulary and does not generalize well to new items.

A big challenge in training multi-class classification model with softmax is the training cost when the number of classes is huge (e.g., millions). Hierarchical softmax [11, 18] and sampled softmax [1, 2] are two most common approaches used to improve training speed. Hierarchical softmax defines a tree for categories based on their attributes and makes hierarchical decisions traversing the tree to get the final category. On the other hand, sampled softmax specifies a sampling distribution Q from which a subset of the label space is drawn to approximate the gradient. Researchers usually use simple distributions, e.g., unigram or bigram distributions based on sample frequency [1, 2], or a power-raised distribution of the unigram [17]. However, neither hierarchical softmax nor sampled softmax is applicable to our two-tower architecture where label is associated with a rich set of content features.

People started to adopt two-tower DNNs to learn representation from content features in language models [6, 16, 19]. Two-tower DNNs have also been introduced for retrieval task to leverage rich content feature on item side in recommender systems with application in video recommendations [24], where batch negative sampling based on item frequency estimation is adopted to correct sampling bias. In contrast, our work found it important to reduce the selection bias brought by batch negative sampling in the application for app recommendations, which has not been considered in existing works to the best of our knowledge.

3 MODELING FRAMEWORK

In this section, we first provide a mathematical formulation for the retrieval task in large-corpus recommender systems. We then present the modeling approach based on a two-tower deep neural network and describe how we train the model using in-batch

negative sampling. Finally, we introduce the Mixed Negative Sampling (MNS) technique to address the selection bias of the batch negatives.

3.1 Problem Formulation

The retrieval task in recommendation systems aims to quickly select hundreds to thousands of candidate items from the entire item corpus given a certain query. In particular, a query could be a piece of text, an item (e.g., an app), a user, or a mixture of these. Here both queries and items can be represented as feature vectors capturing wide variety of information. We treat the retrieval problem as a multi-class classification problem, and the likelihood of suggesting an item from a large corpus (classes) C is formulated as a softmax probability:

$$P(y|x) = \frac{e^{\varepsilon(x,y)}}{\sum_{j\in C} e^{\varepsilon(x,y_j)}}, \tag{1}$$

where $\varepsilon(x, y)$ denotes the logits provided by the retrieval model, with feature vectors x and y representing the query and the item respectively.

3.2 Modeling Approach

We adopt a two-tower DNN model architecture for computing logits $\varepsilon(x, y)$. As shown in Figure 1, the left tower and right tower learn latent representations of given query and item separately. Formally, we denote the two DNN towers by functions $u(x; \theta)$ and $v(y; \theta)$, which map query and item features x and y to a shared embedding space. Here θ denotes all the model parameters. The model outputs the inner product of query and item embeddings as logits in Equation (1), i.e.,

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

To simplify notations, we denote u as the embedding for a given query x and v_j as the embedding for item j from the corpus C. The cross-entropy loss for a {query (x), item (y_l, positive label)} pair becomes:

$$L = -\log(P(y_l|x)) = -\log\left(\frac{e^{\langle u,v_l\rangle}}{\sum_{j\in C} e^{\langle u,v_j\rangle}}\right). \tag{2}$$

Taking gradient of Equation (2) with respect to parameter θ gives

$$\begin{aligned} &\nabla_\theta(-\log P(y_l|x)) \qquad (3)\\ &= -\nabla_\theta(\langle u, v_l\rangle) + \sum_{j\in C} \frac{e^{\langle u,v_j\rangle}}{\sum_{j\in C} e^{\langle u,v_j\rangle}} \nabla_\theta(\langle u, v_j\rangle)\\ &= -\nabla_\theta(\langle u, v_l\rangle) + \sum_{j\in C} P(y_j|x)\nabla_\theta(\langle u, v_j\rangle). \end{aligned}$$

The second term represents the expectation of $\nabla_\theta(\langle u, v_j\rangle)$ with respect to $P(\cdot|x)$ (referred to as target distribution). It is generally impractical to compute the second term over all items in a huge corpus. As a result, we approximate this expectation by sampling a small number of items using importance sampling [2].

Specifically, we sample a subset of items C' from the corpus with a predefined distribution Q with Q_j being the sampling probability of item j and estimate the second term in Equation (3) as:

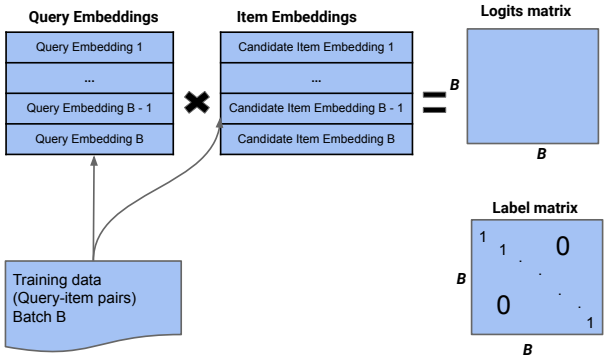

Figure 2: Batch negative sampling for training two-tower DNN model.

$$\mathbb{E}_P[\nabla_\theta(\langle u, v_j\rangle)] \approx \sum_{j\in C'} \frac{\omega_j}{\sum_{j'\in C'} \omega_{j'}} \nabla_\theta(\langle u, v_j\rangle), \tag{4}$$

where $\omega_j = e^{\langle u,v_j\rangle - log(Q_j)}$ incorporates the logQ correction utilized in sampled softmax [1, 2].

A commonly-used sampling strategy for two-tower DNN model is the batch negative sampling. Specifically, batch negative sampling treats other items in the same training batch as sampled negatives and therefore the sampling distribution Q follows the unigram distribution based on item frequency. It avoids feeding additional negative samples to the right tower and thus saves computation cost. Figure 2 shows the computation process in one training batch. Given B pairs of {query, item} in a batch, features of B queries and B items go through the left and right towers of the model, respectively, producing $B \times K$ (K is the embedding dimension) embedding matrices U and V. Then the logits matrix can be calculated as $L = UV^T$. While the batch negative sampling significantly improves training speed, we discuss its problems in the next sub-section and propose an alternative sampling strategy accordingly.

3.3 Mixed Negative Sampling

Controlling bias and variance of the gradient estimator is critical to model quality. There are two ways to reduce bias and variance [2]: (1) increasing the sample size; (2) reducing the discrepancy between proposed Q distribution and target distribution P.

In case of training two-tower DNN models, batch negative sampling implicitly sets sampling distribution Q to be unigram item frequency distribution. It has 2 problems in recommedation scenarios:

- (1) **Selection bias**: for example, an item with no user feedback will not be included as a candidate app in the training data. Thus it will never be sampled as a negative with batch negative sampling. Consequently, the model lacks capability to differentiate items with sparse feedback w.r.t other items.
- (2) **Lack of flexibility to adjust sampling distribution**: The implicit sampling distribution Q is determined by the training data and cannot be further adjusted. Deviation of

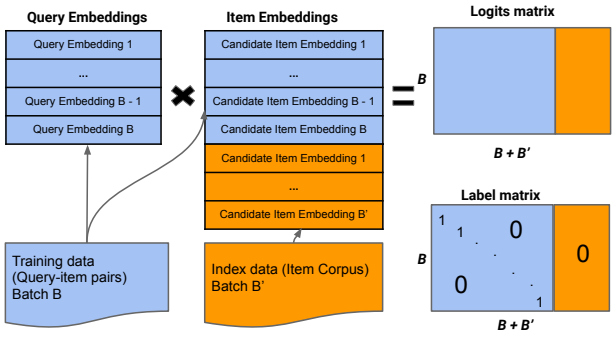

Figure 3: Illustration of MNS for training two-tower DNN model.

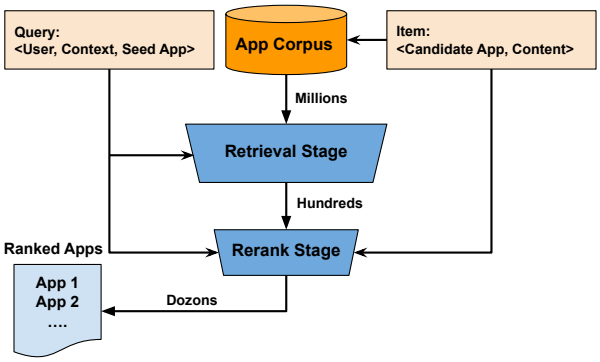

Figure 4: Overview of Google Play app recommendations.

Q from target distribution P might result in a significant amount of bias.

We propose Mixed Negative Sampling (MNS) to tackle these problems. It uniformly samples B' items from another data stream. We refer the additional data stream as index data, which is a set composed of items from the entire corpus. These items serve as additional negatives for every single batch. Figure 3 shows the computation process for a training batch. In addition to the $B \times K$ query embedding matrix U_B, and the $B \times K$ candidate item embedding matrix V_B, B' candidate items uniformly sampled from the index data are fed into the right tower to produce a $B' \times K$ candidate item embedding matrix V'_B. We concatenate V_B and V'_B to get a $(B + B') \times K$ candidate item embedding matrix V. Eventually we have the $B \times (B + B')$ logits matrix $L = UV^T$. The label matrix therefore becomes $B \times (B + B')$ with an all 0 $B \times B'$ matrix appended to the original $B \times B$ diagonal matrix. Accordingly, the loss function for a training batch becomes

$$L_B \approx -\frac{1}{B} \sum_{i \in [B]} \log\left(\frac{e^{\langle u_i, v_i \rangle}}{e^{\langle u_i, v_i \rangle} + \sum_{j \in [B+B'], j \neq i} w_{ij}}\right), \quad (5)$$

where $w_{ij} = e^{\langle u_i, v_j \rangle - log(Q^*_j)}$ with u_i being the ith row of U and v_j denoting the jth row of V. Here the sampling distribution Q^* becomes a mixture of item frequency based unigram sampling and uniform sampling, characterized by a ratio between the batch size B and B'.

MNS tackles the two problems associated with batch softmax described above: (1) Reducing selection bias: all items in the corpus have a chance to serve as negatives so that the retrieval model has better resolution towards fresh and long-tail items; (2) Enabling more flexibility in controlling sampling distribution: the effective sampling distribution Q is a mixture of item frequency based unigram distribution from training data and uniform distribution from index data. With a fixed batch size B, we are able to experiment with different Q by adjusting B'. B' here can be tuned as a hyper-parameter.

4 CASE STUDY: GOOGLE PLAY APP RECOMMENDATION

This section introduces how we apply the two-tower DNN modeling approach for the app recommendation in Google Play, as a real-world case study to experiment with our proposed method.

4.1 System Overview

The app recommendation system for the app landing page in Google Play provides a slate of recommended apps to users when they are browsing a particular app (referred to as seed app). Figure 4 illustrates the high-level architecture of the system. Given a query represented as a concatenation of user, context, and seed app features, the system serves personalized app recommendations in a two-phase manner. The system contains multiple candidate generation modules for retrieval, including both human-crafted rules and machine-learned models. These modules retrieve hundreds of apps from the app corpus. Afterwards, all the apps retrieved are ranked by a reranking model.

4.2 Google Play's Two-tower DNN Model

We apply two-tower DNN model architecture for the retrieval problem in Google Play app recommendation. As shown in Figure 5, the left tower and right tower learn latent representations of given query and candidate app, respectively. Here query features can be a mixture of features representing user, context and seed app, while candidate app features can be both sparse app IDs and content features (e.g., the category of an app). Our training data is constructed from logged user implicit feedback in the form of {query, candidate app} pairs, where candidate app is the next app which user installed from recommended apps. To apply MNS, we prepare an index data including all of the candidate apps and corresponding content features with values from the most recent snapshot.

4.3 Indexing and Model Serving

In order to serve recommendations specially tailored based users' real-time updates, we compute query embedding on the fly via the left tower in the model, while the candidate app embeddings are computed and indexed offline. We build a fast nearest neighbor search service to retrieve top K most relevant candidate apps in

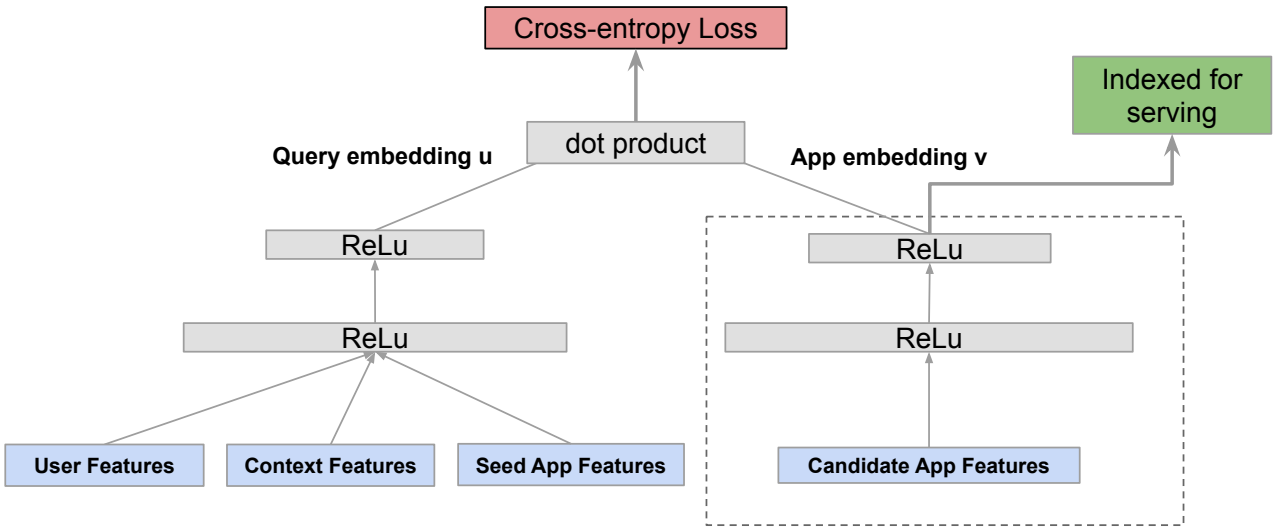

Figure 5: Two-tower model architecture for Google Play app recommendation.

the embedding space. The indexer of the nearest neighbor search is built off-line by applying quantizied hashing and tree search algorithms [12, 22], enabling us to efficiently retrieve hundreds of candidate apps in real-time from a large corpus of millions of apps.

5 EXPERIMENT RESULTS

We demonstrate the effectiveness of our proposed technique via extensive offline experiments on real-world Google Play data. In addition, live experiments on Google Play also show that the two-tower model with MNS delivered significant top-line metric wins.

5.1 Offline Studies

We train the model on 30 days' rolling window of Google Play's logged data and evaluate on the 31st day. To account for weekly pattern, we repeat the evaluation for 7 times and each time we move the rolling window by 1 day. We report the average metric across the 7 train-eval datasets. For a given {query, candidate app} pair in the eval-set, we find the top k nearest neighbors in the embedding space for the query. Recall is the main optimization objective in the retrieval phase of our app recommendation system. We thus report Recall@K, i.e., the average probability for the candidate apps to appear among the top k nearest neighbors retrieved for the query, as our offline metric.

Effectiveness of MNS. To understand the effectiveness of MNS, we experiment with the following 3 algorithms: (1) MLP with sampled softmax without app content features. This is the latest production retrieval system and thus serves as our baseline; (2) Two-tower DNN with batch negative sampling; (3) Two-tower DNN with MNS. Table 1 reports the results.

For MLP with sampled softmax model, we use a two-layer DNN with hidden layer sizes [1024, 128] to encode query. For the two-tower model, we use feed forward layers with hidden layer sizes [512, 128] for both left and right towers. It is worth noting that the two-tower and MLP models have roughly the same number of model parameters, since each of the two towers has half the size of the MLP model and the same embedding sizes are used for the input features across all models. We ensure that the effective sample size in negative sampling is 2048 across all the 3 models as shown in Table 1. We train both types of models using Adagrad as optimizer with learning rate at 0.01. *ReLU* is used as activation function for all hidden layers except that no activation is used for top hidden layers.

From Table 1, we observe that batch negative sampling performs worse than sampled softmax even it incorporates extra app content features. We observe quite a few irrelevant tail apps in its retrieval results. This observation supports our hypothesis that batch softmax suffers from selection bias. Low-quality tail apps do not appear as negatives frequent enough. As such, they are not demoted as they should have been during training. Another observation from Table 1 is that Mixed Negative Sampling performs the best as expected. Compared with MLP with sampled softmax, MNS includes extra app content features. Compared with batch negative sampling, it reduces the selection bias.

Hyper-parameter B' for MNS. We further explore the MNS strategy by looking at how Recall@K changes under different index data batch size B' with a fixed training batch size B. Table 2 reports the results. Our model performs the best when B' is 8192, according to offline tuning of B' as a hyper-parameter. We conjecture that although further increasing B' results in a larger sample size, it leads to a sampling distribution too close to the uniform distribution. It deviates from the desired serving time distribution and thus hurts the model quality. Besides, large index batch size B' significantly increases the training cost and therefore a reasonable size B' is preferred.

Table 1: Recall@K of several models in the Google Play dataset.

Model	Training Batch Size	Index Data Batch Size	Recall@10	Recall@50	Recall@100
MLP with Sampled Softmax	2048	N.A.	0.4283	0.7226	0.8351
Two-tower with Batch Negatives	2048	N.A.	0.3987	0.6439	0.7287
Two-tower with MNS	**1024**	**1024**	**0.4473**	**0.7474**	**0.8590**

Table 2: Recall@K of two-tower models trained with different index data batch size.

Training Batch Size (B)	Index Data Batch Size (B')	Recall@10	Recall@50	Recall@100
1024	1024	0.4473	0.7474	0.8590
1024	2048	0.4501	0.7480	0.8577
1024	4096	0.4549	0.7576	0.8684
1024	**8192**	**0.4780**	**0.7877**	**0.8939**
1024	12000	0.4589	0.7567	0.8635
1024	16000	0.4443	0.7447	0.8575

Table 3: Live experiment results of 3 models in Google Play.

Model	High-quality App Install Gain
MLP with Sampled Softmax	+0.00%
Two-tower with **Batch Negative Sampling**	-1.46%
Two-tower with **Mixed Negative Sampling**	**1.54%**

5.2 Online A/B testing

Beyond offline studies, we conduct live experiments (A/B testing) for 2 weeks. The control group consists of 1% randomly selected users, who are presented with app suggestions made by the production recommendation system using a sampled softmax model. Similarly, we set up two 1% experiment groups to present app suggestions from two-tower models with batch negative sampling, and with MNS, respectively. We report High-quality App Install Gain as our top-line metric here. We define High-quality App Install as the number of apps that users actually used after installing, as opposed to uninstalling within 1 day or having no usage. As shown in Table 3, compared with the production sampled softmax model, two-tower using batch softmax is significantly worse, while using MNS improves High-quality App Installs by +1.54% with statistical significance. In our side-by-side comparisons, we also observe more relevant app recommendations from two-tower model with MNS.

6 CONCLUSION

This paper introduced novel sampling approach called Mixed Negative Sampling for training two-tower neural network for large corpus item retrieval in recommendations. We discussed the selection bias of the commonly used batch negative sampling method, and showed that MNS is effective in reducing such bias by additionally sampling uniform negatives from the item corpus. We applied the MNS-based two-tower modeling approach to build a retrieval model for Google Play. This new model is able to incorporate rich content features of apps, and thus resolves the cold-start problem of the old system. Lastly, we demonstrated the effectiveness of our approach by showing both offline and online improvements.

REFERENCES

[1] 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)*.

[2] 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

[3] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed Chi. 2018. Top-K Off-Policy Correction for a REINFORCE Recommender System. arXiv:cs.LG/1812.02353

[4] 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

[5] 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

[6] Muthuraman Chidambaram, Yinfei Yang, Daniel Cer, Steve Yuan, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Learning Cross-Lingual Sentence Representations via a Multi-task Dual-Encoder Model. *CoRR* abs/1810.12836 (2018).

[7] 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.

[8] Daniel Gillick, Alessandro Presta, and Gaurav Singh Tomar. 2018. End-to-End Retrieval in Continuous Space. arXiv:cs.IR/1811.08008

[9] 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

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

[11] Joshua Goodman. 2001. Classes for Fast Maximum Entropy Training. In *ICASSP*.

[12] 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*, Vol. 51. PMLR.

[13] 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) *(WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva,

Switzerland, 173–182. https://doi.org/10.1145/3038912.3052569
[14] 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
[15] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related Pins at Pinterest: The Evolution of a Real-World Recommender System. In *WWW*.
[16] Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJvJXZb0W
[17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. abs/1301.3781 (2013).
[18] Frederic Morin and Yoshua Bengio. 2005. Hierarchical Probabilistic Neural Network Language Model. In *AISTATS*.
[19] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016. Learning Text Similarity with Siamese Recurrent Networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*. Association for Computational Linguistics, Berlin, Germany, 148–157. https://doi.org/10.18653/v1/W16-1617
[20] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based News Recommendation for Millions of Users. In *KDD*.
[21] S. Rendle. 2010. Factorization Machines. In *2010 IEEE International Conference on Data Mining*. 995–1000. https://doi.org/10.1109/ICDM.2010.127
[22] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N Holtmann-Rice, David Simcha, and Felix Yu. 2017. Multiscale Quantization for Fast Similarity Search. In *Advances in Neural Information Processing Systems 30*.
[23] 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
[24] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Ajit Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations. *13th ACM Conference on Recommender Systems. Copenhagen, Denmark* (2019).
[25] Andrew Zhai, Dmitry Kislyuk, Yushi Jing, Michael Feng, Eric Tzeng, Jeff Donahue, Yue Li Du, and Trevor Darrell. 2017. Visual Discovery at Pinterest. (02 2017).