B. Tech Project Work Report

On

Enhancing the Performance of Two Tower Models using Negative Mining and Sampling

Mathew Aju Thomas

(201CS231)

Prathamesh Irappa Dongritot

(201CS242)

Rahul Syam

(201CS245)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,

SURATHKAL, MANGALORE - 575025

April, 2024

DECLARATION

We hereby declare that the B. Tech Project Work Report entitled **Enhancing the Performance of Two Tower Models using Negative Mining and Sampling** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfillment of the requirements for the award of the Degree of **Bachelor of Technology** in **Computer Science and Engineering** is a *bonafide report of the work carried out by me.* The material contained in this report has not been submitted to any University or Institution for the award of any degree.

1. Mathew Aju Thomas (201CS231)

2. Prathamesh Irappa Dongritot (201CS242)

3. Rahul Syam (201CS245)

Department of Computer Science and Engineering

Place: NITK, Surathkal.

Date: Friday 12th April, 2024

CERTIFICATE

This is to *certify* that the B. Tech Project Work Report entitled **Enhancing the Performance of Two Tower Models using Negative Mining and Sampling** submitted by:

(1) **Mathew Aju Thomas** (201CS231)

(2) **Prathamesh Irappa Dongritot** (201CS242)

(3) **Rahul Syam** (201CS245)

as the record of the work carried out by them, is *accepted as the B. Tech Project Work Report submission* in partial fulfillment of the requirements for the award of degree of **Bachelor of Technology** in Computer Science and Engineering.

Guide

Mrs. Vani M

Department of Computer Science and Engineering

NITK, Surathkal

Chairman - DUGC

Dr. Manu Basavaraju

Department of Computer Science and Engineering

NITK, Surtahkal

ACKNOWLEDGEMENT

We take this opportunity to express our deepest gratitude and appreciation to all those who have helped us directly or indirectly towards the successful completion of this project. We would like to express our gratitude to our guide Mrs.Vani M Ma'am, Associate Professor, Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal for her continuous encouragement and support in carrying out our project work and for continuously mentoring our work and correcting our mistakes, without which it would have been difficult to carry out this project. We would like to thank her for all the help and necessary suggestions that she has always provided us with. We would also like to thank all the supporting staff members of the Department of Computer Science and Engineering. Without their timely help and support, it would not have been possible to carry out our project.

Abstract

Recommender systems are vital in online platforms for suggesting relevant items to users. Two-tower recommender architectures are powerful but rely heavily on efficient negative sampling techniques during training. Traditional in-batch negative sampling faces scalability and efficiency issues, limiting its effectiveness. This project introduces a novel approach integrating Cross-Batch Negative Sampling (CBNS) and NGAME (Negative Mining-aware Mini-batching) to enhance two-tower recommender systems' accuracy. CBNS stores item embeddings from previous mini-batches, reducing redundancy and offering a wider distribution of negative samples. NGAME optimizes mini-batch creation by prioritizing informative negative samples, improving the training signal. Evaluation on a benchmark dataset is expected to show significant improvements in accuracy metrics, targeting a minimum of 10% increase in Recall@k, for various values of k. Successful implementation can benefit recommender systems across diverse platforms, enhancing personalization and relevance.

Keywords: Two-Tower DNN, CBNS, MNS, NGAME, Negative Sampling, Bias Correction, Memory Bank, Feature Drift, Recommendation Metrics

Contents

List of Figures

List of Abbreviations

CBNS : **C**ross-**B**atch **N**egative **S**ampling

CNN : **C**onvolutional **N**eural **N**etwork

DNN : **D**eep **N**eural **N**etwork

FIFO : **F**irst-**I**n **F**irst-**O**ut

IBNS : **I**n-**B**atch **N**egative **S**ampling

MF : **M**atrix **F**actorization

MLP : **M**ulti **L**ayered **P**erceptron

MNS : **M**ixed **N**egative **S**ampling

NDR : **N**etwork **D**etection and **R**esponse

NGAME : **Neg**ative Mining **A**ware **M**ini-Batching for **E**xtreme Classification

List of Tables

Chapter 1

Introduction

1.1 Overview

In today's digital world, recommendation systems play a vital role in suggesting relevant items to users across many different platforms. The recommendation systems recommend items of personal interest to users from a vast candidate pool, where learning high-quality representations for users and items is the main objective. The ever-expanding world of online recommendations demands ever-more sophisticated recommendation systems. One of the most crucial challenges when building real-world recommendation systems is to accurately score millions to billions of items in real time.

Many industrial systems have a two-stage architecture, consisting of retrieval model which retrieves a small fraction of relevant items from the item corpus, followed by a ranking model which returns the retrieved items, ordered based on criteria such as clicks, ratings or interaction time. Two-tower neural network architectures have emerged as a powerful approach for the task of personalized recommendations. These systems leverage separate neural network architectures to learn latent representations of users and items within a shared embedding space. This approach allows them to capture complex user-item interaction patterns beyond simple co-occurrence statistics.

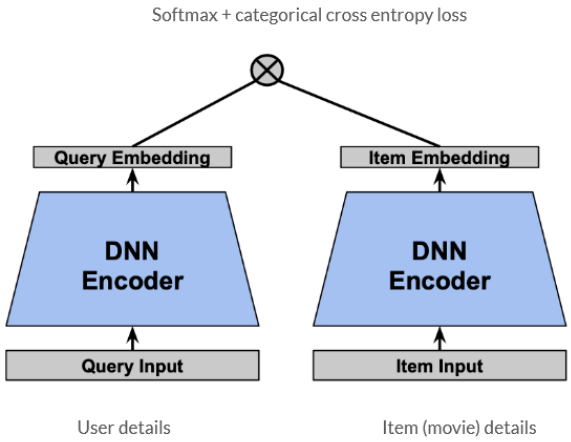

Figure 1.1: Two-tower deep neural network

1.2 Motivation

Training a two-tower recommender over the large-scale item corpus is typically formulated as an extreme multi-class classification task, using sampled softmax. An important aspect influencing the effectiveness of two-tower architectures is the selection of informative negative examples during training. These two-tower models rely heavily on the techniques used to sample negative examples during training. In the traditional **In-Batch Negative Sampling(IBNS)** strategy, negatives are drawn at random from the same mini-batch. The in-batch strategies are inherently limited by the size of mini-batches. It is reasonable that increasing the size of mini-batches benefits negative sampling schemes and can usually boost the performance immediately.

A naive solution is that, in each training iteration, we traverse the entire item corpus and collect their encoded dense representations, after which we conduct negative sampling and optimize the loss. However, such a solution is too time-consuming for training two-tower recommender models, and suffers from limitations in scalability, efficiency, and the exploration of the negative space. As datasets grow larger, in-batch sampling becomes computationally expensive and struggles to handle massive amounts of data. Encoding all items within each training mini-batch becomes memory-intensive, hindering the application of two-tower recommendation systems to truly large-scale data. Additionally, in-batch sampling suffers from inefficiency by restricting the model to a limited pool of negative examples, from the current mini-batch only. As negatives are restricted to the current mini-batch, the model has

difficulty learning a comprehensive representation of the negative space, potentially hindering its ability to make accurate recommendations. It repeatedly encodes the same items across mini-batches, wasting valuable computational resources and slowing down the model's learning process. These limitations can significantly impact the overall accuracy of two-tower recommendation systems.

This project tackles the aforementioned challenges head-on, aiming to address these limitations and enhance the performance of two-tower recommendation systems. We survey some existing negative sampling techniques, and propose an approach that makes use of two innovative techniques: **Cross-Batch Negative Sampling (CBNS)** and **Negative Mining-aware Mini-batching (NGAME)**. CBNS offers a wider distribution of negative samples compared to in-batch methods, by storing encoded item representations from previous mini-batches in a "memory bank". This allows the model to draw additional negative samples from this bank during training, reducing redundancy of negative examples. This can lead to the model having a more comprehensive understanding of the negative space. NGAME complements CBNS by focusing on optimizing mini-batch creation within the training process. It prioritizes informative negative examples that are most likely to challenge the model, improving the training signal and guiding the learning process more effectively. By integrating these techniques, significant improvements can be made in the performance of two-tower recommendation systems, paving the way for a future of highly personalized recommendations across online platforms.

1.3 Organization of Thesis

The remainder of this report is organized as follows. In chapter 2, we survey some existing work regarding the two-tower recommendation systems, negative sampling techniques, sampling bias corrections, and negative mining. We also describe the problem statement, as well as the primary objectives of our project. In chapter 3, we describe key algorithms such as CBNS, sampling probability estimation. and NGAME. We also outline the proposed approach for combining NGAME with CBNS.

In chapter 4, we describe the dataset used for training the two-tower model, as well as the model architecture, configuration of model parameters, and the environment

used for the implementation of the various negative sampling and mining algorithms. We also discuss the results obtained. Finally, in chapter 5, we conclude the report and discuss the scope for future work.

Chapter 2

Related work

2.1 Two-Tower Recommendation Systems

Two-tower recommendation systems[4] represent a powerful architecture for predicting user preferences and driving personalized recommendations across diverse platforms. These systems operate by learning distinct low-dimensional vector representations, or embeddings, for both users and items within a latent factor space. This allows the model to capture complex relationships and user-item interactions beyond simple co-occurrence patterns.

At the core of a two-tower system lies the embedding generation process. This typically involves neural network architectures like Multi-Layer Perceptrons (MLPs) or Convolutional Neural Networks (CNNs) for item embeddings, and Recurrent Neural Networks (RNNs) for user embeddings that can capture sequential behavior. These networks are trained on user-item interaction data, such as ratings, clicks, or purchases. The user tower processes user interaction history to learn a user embedding that encapsulates their preferences and interests. Similarly, the item tower processes item attributes and descriptions to generate item embeddings that capture the inherent characteristics and relationships between items.

Once user and item embeddings are obtained, the two towers interact to predict the likelihood of a user interacting with a specific item. This interaction often takes the form of a dot product between the user and item embedding vectors in the latent factor space. The resulting score reflects the model's estimated relevance of the item to the user based on their learned representations. During training, the model optimizes its parameters by minimizing a loss function that compares the predicted score to the

observed interaction[3, 11] (e.g., an item is considered a positive in case the user has previously interacted with it, negative if no previous interaction). This optimization process allows the model to refine the user and item embeddings, leading to more accurate predictions and personalized recommendations.

A crucial aspect influencing the effectiveness of two-tower architectures is the selection of negative examples during training. Traditional in-batch negative sampling, where negatives are drawn from the same mini-batch as positive examples, suffers from limitations in efficiency and the exploration of the negative space. We now look at various negative sampling techniques that aim to address the limitations, specifically focusing on Mixed Negative Sampling (MNS), Cross-Batch Negative Sampling (CBNS), and Negative Mining-aware Mini-batching (NGAME). Additionally, we explore two seminal papers that provide a comprehensive foundation for understanding negative sampling strategies in recommendation systems.

2.2 Deep Neural Networks for Recommendation

The application of deep learning techniques in recommendation systems has gained significant attention due to their ability to capture complex patterns in large-scale datasets. Deep neural networks (DNNs) have shown promise in learning intricate user-item interactions and improving recommendation accuracy. Various deep learning architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and deep autoencoders, have been explored for recommendation tasks. YouTube, as one of the largest online video-sharing platforms, employs sophisticated recommendation algorithms to personalize the content displayed to users. The paper "Deep Neural Networks for YouTube Recommendations"[8] presents an in-depth analysis of YouTube's recommendation system, focusing on the use of DNNs to enhance recommendation quality.

The paper highlights two primary components of YouTube's recommendation system: the candidate generation model and the ranking model. The candidate generation model efficiently selects a subset of videos from a large pool of candidates, leveraging DNNs to predict user engagement probabilities. The ranking model further refines recommendations based on user preferences and historical interactions, employ-

ing complex neural network architectures to optimize video ranking. The architecture of the DNNs used in YouTube's recommendation system is described, emphasizing the incorporation of both video and user features. Video features include metadata, content descriptors, and user engagement signals, while user features encompass demographic information, browsing history, and social interactions. The paper elucidates the feature engineering process and the representation learning techniques employed to extract meaningful features from raw data.

Training deep neural networks for recommendation entails addressing challenges such as data sparsity, scalability, and model complexity. The paper discusses the training pipeline adopted by YouTube, which involves large-scale distributed training using frameworks like TensorFlow. Techniques such as mini-batch optimization, regularization, and gradient clipping are employed to improve model convergence and generalization performance.

2.3 Mixed Negative Sampling

The paper "Mixed Negative Sampling for Learning Two-tower Neural Networks in Recommendations"[11] proposes Mixed Negative Sampling (MNS) as a novel approach to tackle the inherent selection bias present in traditional in-batch negative sampling. In-batch sampling suffers from a fundamental limitation: it restricts the model's exposure to negative examples solely within the current mini-batch. This can lead to the model over-fitting to implicit user feedback within the mini-batch, potentially neglecting relevant negative items from the broader dataset.

MNS tackles this challenge by introducing a mixture of negative sampling strategies within a single mini-batch. The core idea lies in leveraging two distinct negative sampling sources:

- *Batch Negatives*: A portion of negatives are randomly sampled from the current mini-batch, similar to traditional in-batch sampling. This component retains a degree of context within the mini-batch and allows the model to learn from implicit user feedback.

- *Uniform Negatives*: An additional set of negatives are uniformly sampled from the entire item pool. This component injects broader diversity into the neg-

ative sample distribution, ensuring the model is exposed to a wider range of potentially relevant negative items not necessarily present within the current mini-batch.

The key lies in the optimal ratio between batch and uniform negatives. This ratio is introduced as a hyperparameter that can be tuned through techniques like grid search or Bayesian optimization. Finding the optimal ratio necessitates careful consideration of the dataset characteristics and the specific two-tower architecture employed. An overly high proportion of uniform negatives could dilute the model's ability to learn from the context within the mini-batch, while a low ratio might limit the exploration of the broader negative space.

While MNS presents a promising approach to alleviate selection bias, the paper leaves some aspects open for further exploration. The theoretical justification for the specific mixture ratio selection remains an area for deeper investigation. Future research could delve into adaptive mixture ratios that dynamically adjust based on user behavior within the mini-batch or item characteristics like popularity or category. This could potentially lead to a more nuanced and data-driven approach to negative sampling within two-tower recommendation systems.

2.4 Cross Batch Negative Sampling

While MNS focuses on enriching the negative sample distribution within a single mini-batch, Cross-Batch Negative Sampling (CBNS)[6] takes a fundamentally different approach. Unlike a single, definitive research paper, CBNS has emerged as a technique explored and implemented in various studies. It represents a paradigm shift in negative sampling for two-tower recommendation systems, particularly when dealing with large-scale datasets.

The core idea of CBNS lies in the utilization of a memory bank. This memory bank serves as a repository of pre-generated item embeddings, meticulously crafted from previous mini-batches during the training process. During each training iteration, instead of re-encoding all items within the current mini-batch, the model draws negative examples from this memory bank. This approach offers several compelling advantages:

- *Enhanced Scalability*: Traditional in-batch negative sampling suffers from significant scalability bottlenecks when dealing with massive datasets. Re-encoding all items in each mini-batch becomes computationally expensive and memory-intensive. CBNS circumvents this limitation by leveraging pre-generated embeddings.

- *Improved Training Efficiency*: In-batch sampling suffers from a redundancy issue. Frequently encountered items are repeatedly encoded within each mini-batch, leading to wasted computational resources and slowing down the training process. CBNS eliminates this redundancy by reusing stored item embeddings from the memory bank.

- *Potentially Broader Negative Sample Exploration*: In-batch sampling restricts the model's exposure to negative examples solely within the current mini-batch. This limitation can hinder the model's ability to learn a comprehensive representation of the negative space, potentially leading to suboptimal recommendation accuracy. CBNS offers a potential solution by drawing negative samples from a broader pool stored within the memory bank. This broader negative sample distribution allows the model to explore a more diverse set of negative items, potentially leading to a more nuanced understanding of user preferences and ultimately more accurate recommendations.

2.5 Negative Mining Aware Mini Batching

The paper "NGAME: Negative Mining-aware Mini-batching for Extreme Classification"[7] introduces a novel technique called NGAME, specifically designed for extreme classification tasks with large transformer-based models. However, the core concepts underlying NGAME hold immense potential for adaptation to the context of two-tower recommendation systems with negative sampling.

While traditional negative sampling techniques focus on randomly selecting negative examples, NGAME takes a more sophisticated approach. It prioritizes the creation of mini-batches that emphasize informative negative samples – those most likely to challenge the model during training. This focus on informativeness stems

from the inherent difficulty of extreme classification tasks, where the positive class is significantly outnumbered by the negative class.

Adapting NGAME for two-tower recommendation systems necessitates a shift in perspective on informativeness. In this context, informative negatives are not simply rare items but rather those that exhibit a high degree of ambiguity for the model. These could be items with user-item interaction patterns that closely resemble positive examples or items belonging to categories frequently confused with the target item.

By exposing the model to a greater proportion of informative negative examples within each mini-batch, NGAME offers several potential benefits:

- *Improved Training Signal*: Traditional negative sampling methods might expose the model to negatives that are easily distinguishable from positive examples. This can lead to a weak training signal, hindering the model's ability to learn complex decision boundaries and potentially resulting in sub-optimal recommendation accuracy. NGAME, by focusing on informative negatives, provides a more challenging training landscape, forcing the model to refine its decision-making capabilities and potentially leading to a stronger training signal.

- *Enhanced Generalization Capabilities*: A model trained primarily on easily distinguishable negatives might struggle to generalize well to unseen data during deployment. NGAME, by exposing the model to more challenging and ambiguous negatives, can potentially improve its ability to handle unseen user-item interactions and recommend relevant items even for users with limited interaction history.

2.6 Sampling-Bias-Correction

The paper "Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations"[10] tackles the challenge of sampling bias in recommendation systems, particularly for large-scale applications with millions of items. Recommendation systems are ubiquitous, influencing user choices in various domains like e-commerce, video streaming services, and news platforms. These systems typically learn user preferences and recommend items based on collaborative filtering, content-based filtering, or a hybrid approach.

Matrix factorization (MF) is a widely used technique for recommendation systems, especially when dealing with data sparsity, where limited user-item interactions make personalized recommendations difficult. MF decomposes the user-item interaction matrix into lower-dimensional latent factors, capturing user preferences and item characteristics. However, MF can be susceptible to sampling bias if the training data does not accurately reflect the true item distribution.

Neural Recommendation Systems and Existing Bias Correction Methods: The rise of deep learning has led to the exploration of neural networks for recommendation systems. These systems often employ a two-tower architecture, where one tower encodes user features and the other encodes item features. The final recommendation score is obtained by calculating the similarity between the encoded representations. However, a common challenge with neural recommendation systems is the negative sampling strategy used during training. In-batch negative sampling, where negative items are randomly sampled from the mini-batch, can introduce bias, particularly for datasets with power-law item distributions where a small number of items are very popular, while the majority are rarely interacted with.

The Proposed Approach and Novelty: The key contribution lies in their proposed algorithm for estimating item frequency from streaming data to address sampling bias. Their approach avoids the need for a fixed item vocabulary and adapts to changes in item popularity over time. This is particularly beneficial for large-scale recommendation systems where item distributions can be dynamic.

The authors demonstrate the effectiveness of their method by implementing it in a large-scale neural retrieval system for YouTube recommendations. The NDR system retrieves personalized video suggestions from a corpus of tens of millions of videos. Offline experiments on real-world datasets and live A/B testing on YouTube validate the proposed approach, showing improvements in recommendation quality compared to baseline models susceptible to sampling bias.

2.7 Problem Formulation

The retrieval goal of recommendation systems is to quickly select hundreds to thousands of candidate items from the entire item set given a certain query. A query could

be a piece of text, an item (e.g., book, app, video), a user, or a combination of these. The queries and items can be represented as feature vectors capturing a wide variety of information.

The two-tower DNN architecture (also called Siamese encoder) is widely used in large scale recommendation systems. It consists of two neural networks with similar architectures: one of them is used to encode users, and the other is used to encode items. The left tower and right tower learn latent representations, or embeddings, of given users and items separately. The objective of training is to obtain model weights such that embeddings of positive items (i.e., items that the user has interacted with) appear closer to the user embedding, whereas embeddings of negative items appear farther away than the positives.

Training a two-tower recommendation model over a large corpus of users and items is typically formulated as an extreme multi-class classification task using sampled softmax[3] and log-loss. The effectiveness of two-tower architectures depends greatly on the selection of informative negatives during training. Therefore, informative negative mining and negative sampling algorithms are crucial for the efficient training of two-tower models.

2.8 Objectives

This project sets out to achieve advancements in the performance of two-tower recommender systems by tackling the limitations of naive in-batch negative sampling. Our primary objectives are to:

Improve scalability: We aim to overcome the bottleneck faced by in-batch negative sampling when dealing with massive datasets, by incorporating different negative sampling strategies. In Mixed Negative Sampling, a random subset of the item set is initially obtained before the training commences, from which negatives are sampled during training. Using Cross-Batch Negative Sampling (CBNS), we intend to leverage a memory bank of 'out-of-date' item embeddings. This eliminates the need for redundant encoding within each mini-batch, allowing the system to efficiently handle large-scale data and pave the way for the application of two-tower architectures to ever-growing datasets.

Improve training efficiency: The repetitive encoding of frequently encountered items in in-batch sampling leads to wasted computational resources and slows down the training process. We aim to address this inefficiency by utilizing CBNS. By reusing stored item embeddings, we can significantly reduce the computational burden associated with encoding and expedite the training pipeline.

Expand negative sample space exploration: Traditional in-batch sampling restricts the model's exposure to negative examples within the current mini-batch. This potentially hinders its ability to learn a comprehensive representation of the negative space and can lead to suboptimal recommendation accuracy. Our objective is to leverage the capabilities of CBNS to draw negative samples from a broader pool stored in a memory bank. This wider distribution of negative samples will allow the model to explore a more diverse negative space, facilitating a more comprehensive understanding of user preferences and ultimately leading to more accurate recommendations.

Quantify performance improvement: We aim to evaluate the effectiveness of the negative sampling strategies by implementing CBNS and NGAME within a two-tower recommendation system. Our benchmark will be a large-scale dataset with millions of users and items. We will measure the system's performance using established metrics like Recall@k and Top-k Accuracy[1, 9]. Through extensive experimentation, we aim to achieve a minimum of 10% improvement in these key metrics, demonstrating the positive impact of our proposed approach on recommendation accuracy.

Chapter 3

Proposed Methodology

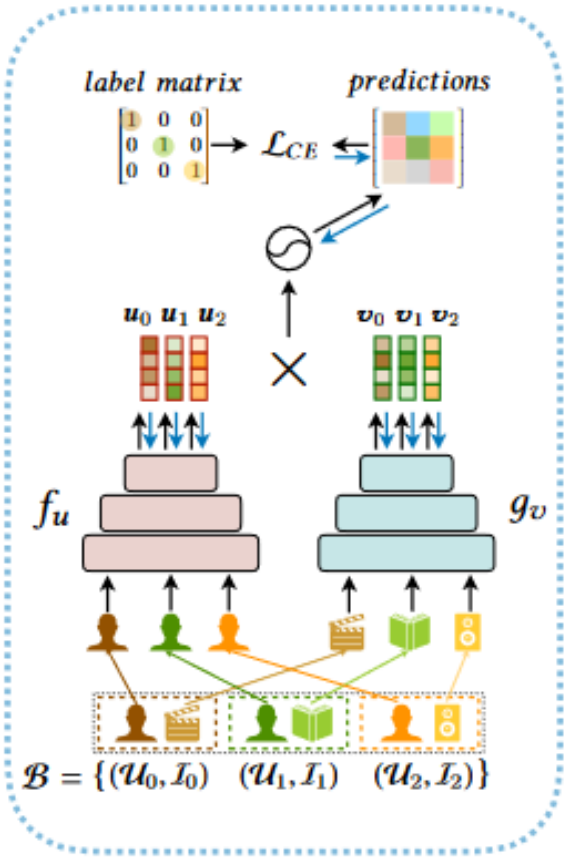

Figure 3.1: Two-tower model architecture

The two-tower models used in large scale recommendation systems, consists of two neural networks(or towers) with similar structures, as shown in Figure 3.1. After the data collection and pre-processing stage, the pre-processed set of user and item feature vectors are passed as input to the user and item towers respectively. The output produced by the user tower and item towers are embedding vectors, belonging

to a common vector space. The model aims to learns weights that minimize the distance between user embeddings and embeddings of corresponding positive items (i.e., items that the user has interacted with), while simultaneously maximizing the distance of negative item embeddings from the user embedding.

3.1 Basic Approach for Two-Tower Recommendation

1. We have two sets; user set U and item set I, where $x_i \in U$ and $y_j \in I$ are sets of pre-processed vectors of features (e.g. user IDs, age, location and item types, creator, description). Given a user feature vector x_i, the goal is to retrieve a subset of items of interest.

2. This is done using the two-tower neural network as shown in Fig 3.1. The user/item tower takes user/item feature vectors as input, respectively.

3. Both the towers produce d-dimensional, real-valued vectors as output. Formally, the two DNN towers are denoted by the functions $f_{u,\Theta} : U \rightarrow R^d$ for users and $g_{v,\Theta} : I \rightarrow R^d$ for items. Thus, the **encoded embeddings** for user feature vectors and item feature vectors are both real-valued vectors in the d-dimensional space. Here Θ denotes the model parameters.

4. After this, we estimate how relevant an item y is to a user x by using a scoring function s. Let $\mathbf{u} = f_{u,\Theta}(x)$ denote the encoded embedding for user x, produced by the user-tower, and $\mathbf{v} = g_{v,\Theta}(y)$ denote the encoded embedding for item y, produced by the item-tower. Then, the output relevance score between the user x and an item y is given as follows: $s(x, y) = \left\langle f_{u,\Theta}(x), g_{v,\Theta}(y) \right\rangle = (\mathbf{u}^T)\mathbf{v} = \mathbf{u.v}$ (the inner-product of user and item embeddings $\mathbf{u}$ and $\mathbf{v}$).

5. The large-scale retrieval of relevant items for users is treated as an extreme-classification problem, with activation function as uniformly sampled softmax[3]: $p(y|x; \Theta) = \frac{e^{s(x,y)}}{\sum_{z \in C} e^{s(x,z)}}$, where C denotes the item corpus.

6. The two towers are trained using the cross-entropy loss function[11]: $L_{CE} = \frac{1}{|B|} \sum_{i=1}^{|B|} log(p(y_i|x_i; \Theta))$, where B is the mini-batch, |B| is the size of the mini-batch and $(x_i, y_i) \in B$. Taking gradient w.r.t Θ gives: $\nabla_\Theta(-log(P(y|x))) =$

$-\nabla_\Theta(\mathbf{u.v})+\sum_{z\in C}\Big(\frac{e^{s(x,z)}}{\sum_{j\in C}e^{s(x,j)}}\nabla_\Theta(\mathbf{u.w})\Big) = -\nabla_\Theta(\mathbf{u.v})+\sum_{z\in C}\Big(\nabla_\Theta(\mathbf{u.w})*P(z|x)\Big)$, where u is the embedding of user x (i.e., $\mathbf{u} = f_{u,\Theta}(x)$), v and w are the embeddings of items y and z respectively (i.e., $\mathbf{v} = g_{v,\Theta}(y), \mathbf{w} = g_{v,\Theta}(z)$).

In step 6 of the above approach, it is impractical to compute the second term over all items in a huge corpus. Therefore, this value is approximated by sampling a smaller number of items using negative mining (discussed later). The negatives for an item x are sampled from the set $B_{items} - P^+(x)$, where B_{items} denotes the items in the mini-batch B and $P^+(x)$denotes the positive items of x. The negative distribution q is a unigram distribution based on item frequency (i.e.,$q(y) = \frac{frequency(y)}{Totalno.ofitems}$). Hence, the equation in step 5 is modified as[6]:

$$p(y|x;\Theta) = \frac{e^{s'(x,y)}}{e^{s'(x,y)} + \sum_{y-\in N} e^{s'(x,y-)}} \tag{3.1}$$

, where N is the set of sampled negatives and the superscript "-" denotes the negatives, and modified score $s'(x,y) = s(x,y) - log(q(y))$ (i.e., the score after incorporating sampling bias correction[2, 3, 10]). The calculation of loss and its gradient is also modified accordingly (by replacing $s(.,.)$ with $s'(.,.)$ and C with N).

3.2 Sampling Probability Estimation

Calculating the sampling probability q(y) for every item y in the mini-batch all at once can be quite time consuming. Instead, the item frequency and the corresponding sampling probability can be dynamically updated during the training process, whenever we encounter an item. The steps of the algorithm are as follows:

1. Input: Learning rate α (not necessarily the same as the one used to train the model), 2-dimensional arrays A and B, each consisting of m arrays of size H each, a set of m independent hash functions hf, with output ranging from 0 to H-1.

2. *For each training step t = 1, 2, 3..., do:*

3. Let B be the current mini batch. Then, for each item $y \in B$:

4. Let max_py = 0 initially

5. **For i = 0 to m-1, do:**

   h = hf[i] //(i'th hash function)

   B[i][h(y)] = (1-α)*B[i][h(y)] + α*(t - A[i][h(y)])

   A[i][h(y)] = t

   max_py = max(max_py, B[i][h(y)])

   end for

6. The estimated probability of item y is q(y) = 1/max_py

The arrays in A are initialized with all zeroes, and the arrays in B are all initialized with 100. The obtained value of q(y) is used in the calculation of the modified score function and the subsequent gradient, as mentioned in Section 3.1.

3.3 Summary of CBNS

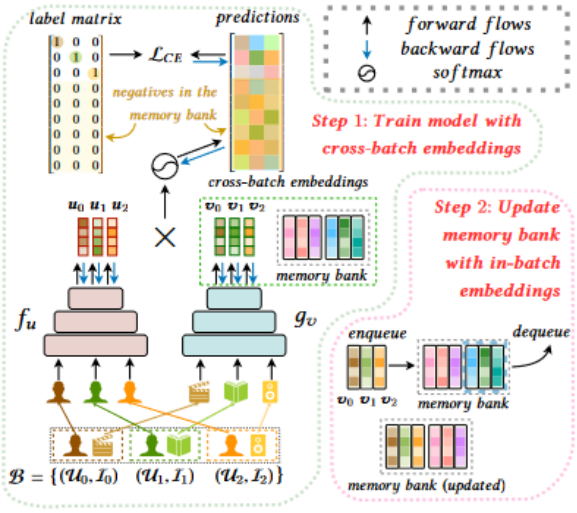

Figure 3.2: Cross-batch Negatives

As the encoder keeps updating during training, the item embeddings from past mini-batches are usually considered out-of-date and discarded. Nevertheless, these can be reused as valid negatives in the current mini-batch.

Feature drift is the sum of the Euclidean distances between the item embeddings of the t'th and (t-d)'th iterations, denoted as $D(y,t;d) = \sum_{y \in I} \left(\| g_v(y;\Theta_t) - g_v(y;\Theta_{t-d}) \|_2^2 \right)$.

FIFO Memory Bank: Since the item embeddings change relatively violently at the early stages, we start the training of the item encoder by using naïve in-batch

negative samplings for the first few iterations. We will use a FIFO memory bank $M = \{(v_i, q(y_i)\}_{1<=i<=|M|}$, where $|M|$is the memory size. This bank stores past item embeddings from previous mini-batches.

At the end of each iteration, we enqueue the embeddings and the respective sampling probabilities of the current mini-batch, and dequeue the earliest ones. The size of the memory bank can be relatively large, because it does not require much memory cost to store those embeddings. In each epoch, calculate the feature drift w.r.t the previous epoch (i.e., $D(y, t; 1)$). When the value of feature drift becomes small enough (< 1), we start sampling negatives from the memory bank M as well.

The softmax output of CBNS is the same as eqn. 3.1, with the only change being in the sampled negative set N for a particular user : $N = B_{items} - P_+(x) \cup M_{items}$.

3.4 Summary of NGAME

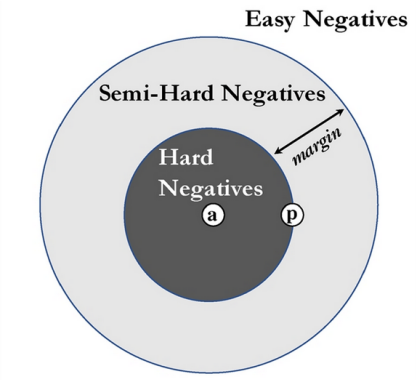

Figure 3.3: Types of negatives

Negative sampling is important for the training of two tower models, as the selection of good quality negatives can help the model learn faster.

Negatives can be broadly classified as easy, semi-hard and hard negatives[5]. They can be categorized into one of the three based on their distance from the anchor data point. A hard negative is closer to the anchor than the positive sample, a semi-hard negative is farther away but still within a margin distance, whereas easy negatives are located beyond the margin. For our purposes, anchor data point refers to a user embedding, and the negative points correspond to the embeddings of the negative/irrelevant items of the user.

Easy negatives can be classified by the model quite easily. Having many easy negatives in the negative set could end up slowing down training as your model could struggle to classify the not-easy samples. Hard negatives on the other hand are difficult to classify, and therefore, are informative negatives that can help the model learn faster. Hard negative mining helps in constructing a smaller size negative set,

by extracting only the hard negatives. This helps the model learn faster and also, due to the reduced size of the negative set, computation costs are reduced in the loss gradient calculation.

NGAME's negative mining strategy relies on the clustering of user embeddings based on Euclidean distance, followed by the grouping together of random clusters to form a mini-batch. Balanced k-means clustering is preferred, as it generates clusters of nearly equal size, which makes it easier to form a mini-batch of specified size, by grouping random clusters together.

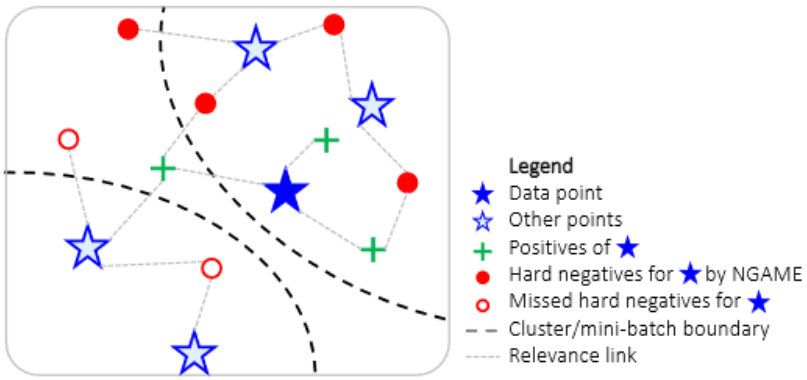

Figure 3.4: NGAME clustering and mini-batching

The main assumption of NGAME is that, if batches are formed from clusters of data point embeddings that are close to each other, the items in the mini-batch are likely to be good quality hard negatives, as they will also occur in close proximity to the user embedding. The framework of the NGAME algorithm is:

Step 0: Initialize the model with weights $\Theta(0)$, mini-batch size |B|, cluster size C, refresh interval τ, hardness threshold r (alternatively, it can be calculated during the training), N user points. f_u - user embedding function, g_v - item embedding function.
Step 1: **for training iterations t = 1, 2, 3, . ., T, do**:
Step 2: **if** $(t-1)\%\tau = 0$**, then**: re-cluster the current user embeddings $f_u(u)_t$ into $\lceil N/C \rceil$ balanced clusters, with each cluster containing $\approx C$ data points each.
Step 3: Choose $\lceil |B|/C \rceil$ random clusters to create a mini-batch B_t of size |B|.
Step 4: Take relevant(positive) items $P^+(i)$ for each user data point $i \in B_t$ from item embeddings $g_v(j)_t$, and let $L_t = \bigcup_{i \in B_t} [P^+(i)]$ (i.e., set of all positive items of every user.

This is the item set for the current mini-batch B_t).

Step 5: Compute $f_u(i)_t$, $g_v(j)_t$ for all $i \in B_t, j \in L_t$.

Step 6: Let r_i = squared maximum Euclidean distance of user i's embedding $f_u(i)_t$ from the embedding $g_v(l)_t$ of any of its positive items in $P^+(i)$. That is, $r_i = max\{\|f_u(i)_t - g_v(l)_t\|_2^2, l \in P^+(i)\}$.

Step 7: Hard-negatives for a user $i \in B_t : \{l \in L_t - P^+(i) : \|f_u(i)_t - g_v(l)_t\|_2^2 \leq r_i\}$, i.e., if l is a positive item of another user and the Euclidean distance between user embedding $f_u(i)_t$ and item embedding $g_v(l)_t$ is $\leq r_i$, then l is a hard negative for user i.

Step 8: Compute the loss and its gradient, then update $\Theta(t)$ using mini-batch-SGD, over the mini-batch B_t.

Step 9: **end for**

The use of extremely hard negatives may affect the stability of training in the early epochs when the model is not well-trained. Thus, start the training with easier negatives. NGAME can tweak the hardness of its negatives by changing the cluster size C in the above algorithm. Start training with small values of C and gradually increase it to get progressively harder negatives in later epochs.

3.5 Cross-Batch NGAME

The modified algorithm for training the two-tower recommender model uses NGAME's negative mining strategy in conjunction with CBNS. The algorithm is specified as follows: -

Initialization step: Initialize the model weights and hyperparameters, mini-batch size |B|, cluster size $C = 0.5$ initially, refresh interval τ, cluster size update interval c, hardness threshold r (alternatively, it can be calculated during the training), N user points. f_u - user embedding function, g_v - item embedding function, empty FIFO memory bank M, learning rate α (used in bias correction; not necessarily the same as the one used in training the DNN), m: the number of independent hash functions, array size H, 2-dimensional arrays A and B, each consisting of m arrays of size H each, a set of m independent hash functions hf, with output ranging from 0 to H-1. The arrays in A are initialized with all zeroes, and arrays in B are all initialized with 100.

Step 1: **for training iterations t = 1, 2, 3, . ., T, do**:

Step 2: **If** $(t-1)\%c = 0$**, then**: set C = C * 2. **If** $(t-1)\%\tau = 0$**, then**: re-cluster the current user embeddings $f_u(u)_t$ into $\lceil N/C \rceil$ balanced clusters, with each cluster containing $\approx C$ data points each.

Step 3: Choose $\lceil |B|/C \rceil$ random clusters to create a mini-batch Bt of size |B|

Step 4: Take relevant(positive) items $P^+(i)$ for each user data point $i \in B_t$ from item embeddings $g_v(j)_t$, and let $L_t = \bigcup_{i \in B_t} [P^+(i)]$ (i.e., set of all positive items of every user. This is the item set for the current mini-batch B_t)

Step 5: Compute $f_u(i)_t$, $g_v(j)_t$ for all $i \in B_t, j \in L_t$. Also find sampling probability q(j) using the previously specified algorithm, for all items $j \in L_t$.

Step 6: Let r_i = squared maximum Euclidean distance of user i's embedding $f_u(i)_t$ from the embedding $g_v(l)_t$ of any of its positive items in $P^+(i)$. That is, $r_i = max\{\|f_u(i)_t - g_v(l)_t\|_2^2, l \in P^+(i)\}$.

Step 7: Let $H_{IN}(i) = \{(v_l, q(l)) : l \in L_t - P^+(i), \|f_u(i)_t - v_l\|_2^2 \leq r_i\}$, where user $i \in B_t, v_l = g_v(l)_t$. i.e., if l is a positive item of another user and the Euclidean distance between user embedding $f_u(i)_t$ and item embedding $g_v(l)_t$ is $\leq r_i$, then l is a hard negative for user i. Here, $H_{IN}(i)$ denotes the in-batch negatives for user i.

Step 8: calculate the feature drift w.r.t the previous epoch (i.e., $D(y, t; 1)$). **If** $D \geq 1$**, then**: $N(i) = H_{IN}(i)$.

Step 8.1: **else**, let $H_{CB}(i) = \{(v, q) \in M : \|f_u(i)_t - v\|_2^2 \leq r_i\}$ (i.e., negatives mined from the item embeddings $v \in M$ of previous mini-batches). $N(i) = H_{IN}(i) \cup H_{CB}(i)$.

end if

Step 9: Let $s(i,j) = \langle f_u(i)_t, g_v(j)_t \rangle$, and $s'(i,j) = s(i,j) - log(q(j))$, where $i \in B_t$ and $j \in P^+(i)$ are the user and positive item feature vectors from the current mini-batch. Calculate sampled-softmax: $P_{cb}(j|i;\Theta) = \frac{e^{s'(i,j)}}{e^{s'(i,j)} + \sum_{(v,q) \in N(i)} e^{w(v,u,q)}}$, where $u = f_u(i)_t$ and $w(v,u,q) = \mathbf{u.v} - log(q)$.

Step 10: The loss is calculated as: $L_{CE} = \frac{1}{|B|} \sum_{i \in B_t, j \in P^+(i)} [log(P_{cb}(j|i;\Theta))]$.

The loss gradient is calculated as: $\nabla_\Theta(-L_{CE}) = \frac{1}{|B|} \sum_{i \in B_t, j \in P^+(i)} [\nabla_\Theta(-log(P_{cb}(j|i;\Theta)))] = \frac{1}{|B|} \sum_{i \in B_t, j \in P^+(i)} \Big(-\nabla_\Theta(\mathbf{u.v}) + \frac{1}{ZS} \sum_{(z,q) \in C} [\nabla_\Theta(\mathbf{u.z}) e^{w(z,u,q)}] \Big)$, where $ZS = \sum_{(z,q) \in C} (e^{w(z,u,q)})$, $u = f_u(i)_t, v = g_v(j)_t$ and $C = N(i) \cup \{v\}$.

Step 11: Enqueue $(v_l, q(l))$ into the memory bank M, for all items $l \in L_t$. (i.e., item embeddings v_l of current mini-batch, along with their respective sampling probabili-

ties $q(l)$).

Step 12: Back-propagate and update $\Theta(t)$ using mini-batch-SGD, over the mini-batch B_t, using the cross-entropy loss and its derivative as calculated above.

Step 13: **end for**

Chapter 4

Results and Analysis

4.1 Dataset Used

The MovieLens 100k dataset serves as a cornerstone for our exploration into the efficacy of CBNS and NGAME within two-tower recommendation systems. It contains 943 users, 1682 movies, 100k ratings. This intricate dataset presents a captivating challenge, encapsulating a rich tapestry of user-movie interactions. Each data point meticulously captures a user's rating for a specific movie, documented on a scale of 1 (dislike) to 5 (like). For our purposes, we considered any movie rated with a score less than 2.5 as a negative, while movies rated at, or higher that 2.5 are considered as positives for a particular user. Beyond the core user-movie ratings, the dataset delves deeper, incorporating supplementary information about both users and movies. User demographics, encompassing factors like age, occupation, and gender, offer valuable insights into user preferences and potential biases. Movie information, including genre, director, and actor details, provides a multifaceted representation of the film landscape.

The intricate relationships between user and movie characteristics within the dataset requires meticulous data pre-processing and feature extraction. To unlock the full potential of the two-tower architecture, meticulous data cleaning techniques will be employed to address missing values, outliers, and potential inconsistencies within the data. Subsequently, feature engineering strategies will be tailored to transform both user and movie information into low-dimensional, informative representations suitable for neural network processing. This transformation may involve techniques like dimensionality reduction, categorical encoding of non-numeric data, and

potential incorporation of external knowledge graphs to enrich the feature space. By meticulously crafting informative user and movie embeddings from the MovieLens 100k dataset, we pave the way for the two-tower recommendation system to leverage CBNS and NGAME for superior recommendation accuracy.

4.2 Experimental Setup

Model architecture and environment: First, the user features such as zip codes, occupation and age are converted into dense vectors of size 32 each, with the help of an embedding layer. These vectors are then concatenated together, and passed as input to the hidden layers of the user tower. The activation function used is Relu. A similar process is followed for producing the embedding of the movie features as well. The hidden layers of the two towers are specified as follows:

Layer	Input size	Output size
Feature-to-dense vector embedding + concatenation	(variable)	160
Hidden	160	512
Output	512	128

Table 4.1: User tower layers

Layer	Input size	Output size
Feature-to-dense vector embedding + concatenation	(variable)	52
Hidden	52	512
Output	512	128

Table 4.2: Movie tower layers

The model was run on a Kaggle notebook, using Python. GPU acceleration was used. Two T4 GPUs were used, each with a memory of 15GB.

Model Configuration: The embedding dimension d is set to 128. The size of mini-batch is set to 8192. The Adagrad optimizer was used, with the base $lr = 0.01$. The distance metric used is Euclidean distance. The effective negative sample size is 2048 across all the algorithms. The memory bank size M for CBNS, as well as the number of additional negatives in MNS, was set to 1152. The model was trained for 200 epochs. We started sampling negatives from the memory bank after 100 epochs. The initial cluster size for NGAME is set to 1, and it doubles every 30 epochs. The minimum permissible number of clusters is set as 10. Clustering of the user embeddings is performed every 10 epochs. 80% of the dataset was reserved as training data, and 20% was reserved for testing purposes.

Metrics: We report results of convergence for all negative sampling strategies, and use the following metrics in the evaluation of the model's performance and training efficiency. i) Recall. ii) Top k accuracy, iii) Training and validation losses, iv) Average time taken per epoch. Here, k represents the number of items retrieved and recommended to the user.

4.3 Results Obtained

We compare the performances of the Cross-Batch Negative Sampling(CBNS), In-Batch Negative Sampling(IBNS), Mixed Negative Sampling(MNS) and Negative Mining Aware Mini-Batching(NGAME) algorithms, based on the aforementioned performance metrics. The values of the various performance metrics, after evaluation of the test dataset, is detailed in the tables below. The values depicted in **bold** represent the highest obtained scores for the respective metrics.

Algorithm	(Avg. time(s)	Recall@1	Recall@5	Recall@10	Recall@50	Recall@100
In-Batch	**0.6905**	0.021	0.096	0.174	0.3105	0.3948
MNS	0.894	0.021	0.097	0.176	0.314	**0.401**
CBNS	9.775	0.017	0.075	0.135	0.266	0.347
NGAME	12.325	**0.024**	**0.099**	**0.179**	**0.319**	0.400

Table 4.3: Comparison of Average Time per Epoch, and Recall@k metrics

Algorithm	Top-1	Top-5	Top-10	Top-50	Top-100
In-Batch	0.0016	0.020	0.046	0.270	0.462
MNS	0.0022	0.019	0.047	0.273	0.471
CBNS	**0.027**	**0.024**	**0.051**	0.252	0.432
NGAME	0.0024	0.022	**0.051**	**0.275**	**0.473**

Table 4.4: Comparison of Top-k Accuracy, for different values of k

The variations of the performance metrics with time during the training of the two-tower model, are plotted graphically, for training and validation data. In the below figures, (a) is CBNS, (b) is IBNS, (c) is MNS, and (d) is NGAME.

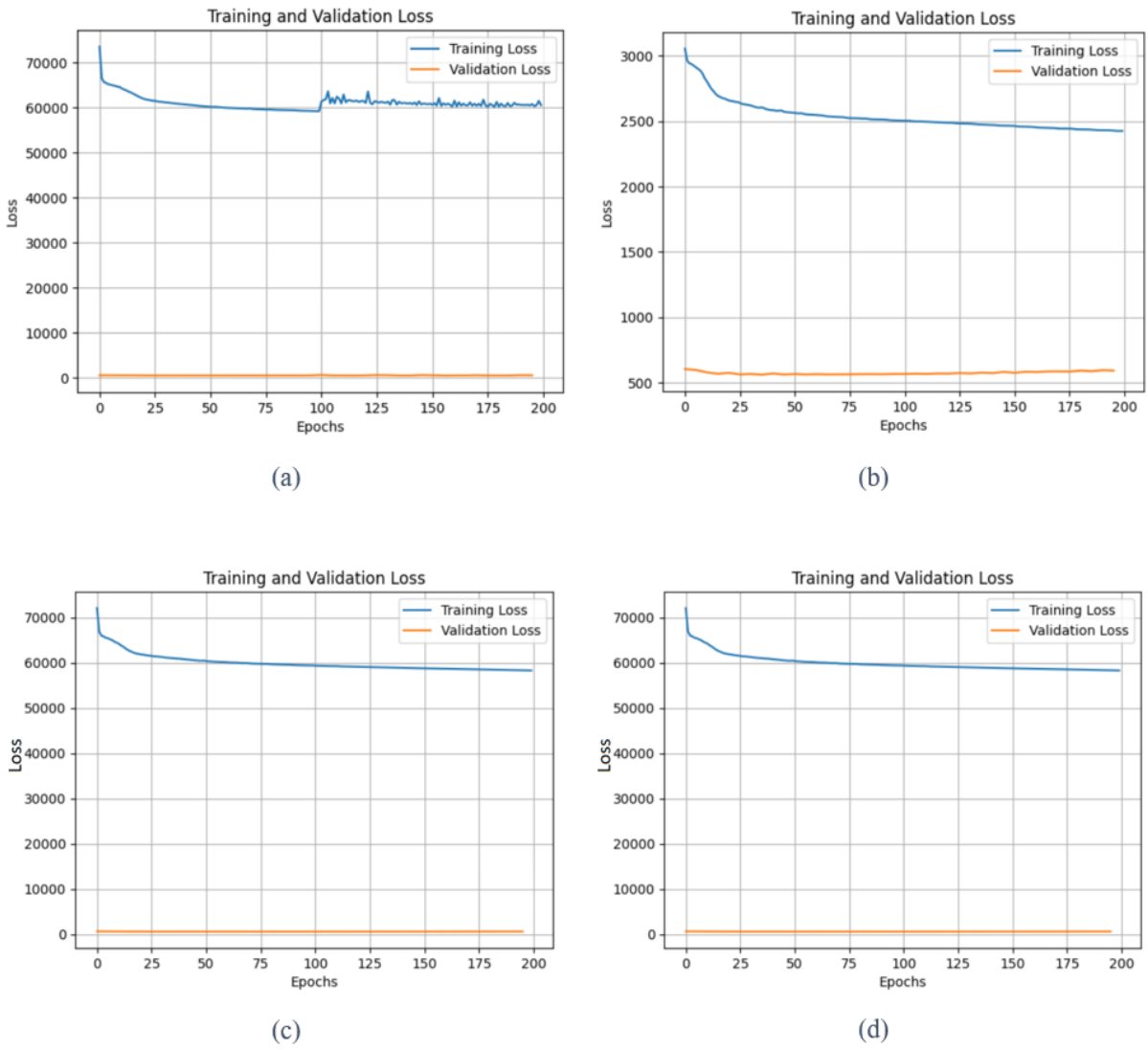

Figure 4.1: Loss curves w.r.t training epoch

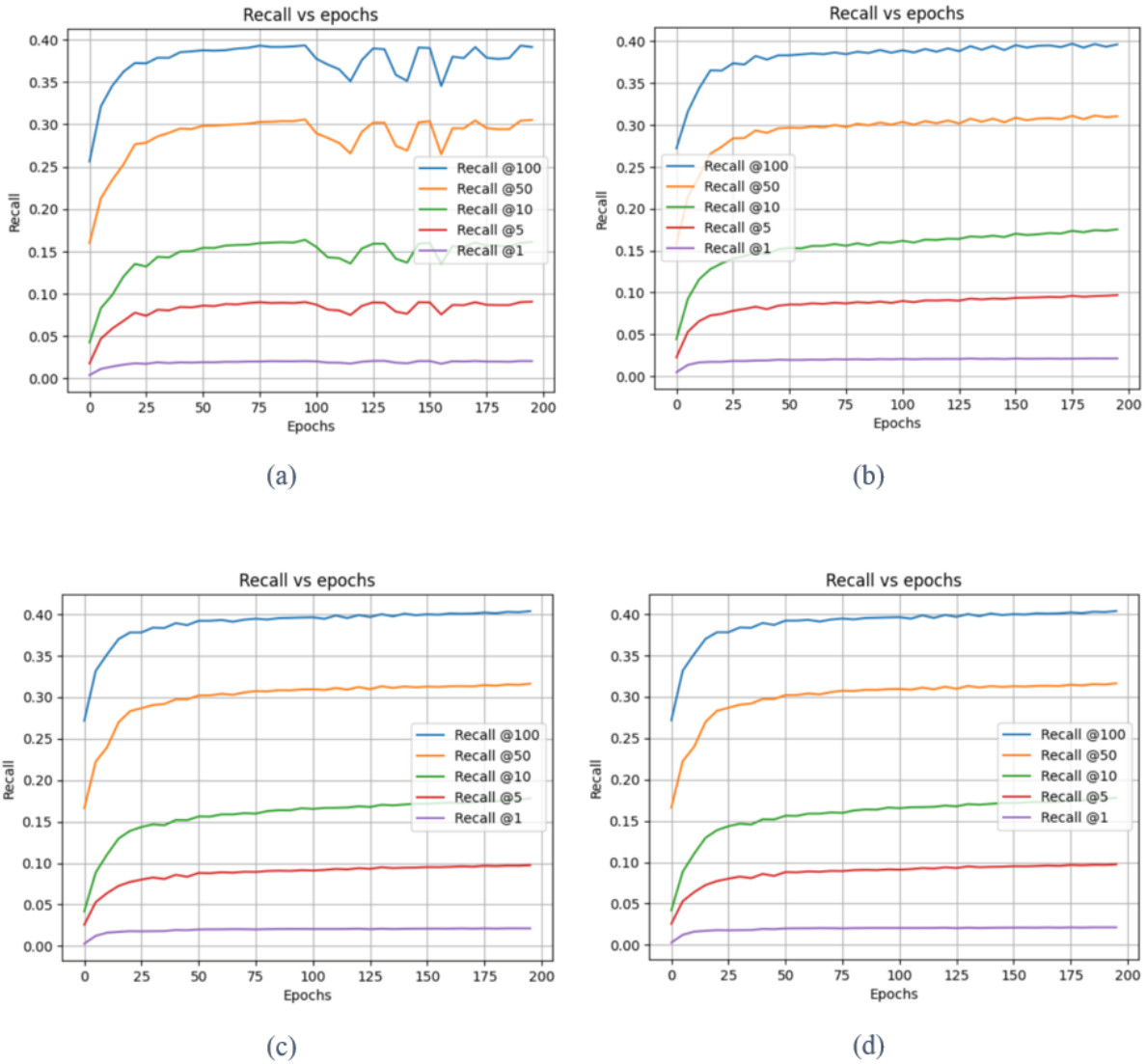

Figure 4.2: Recall@k curves w.r.t training epoch

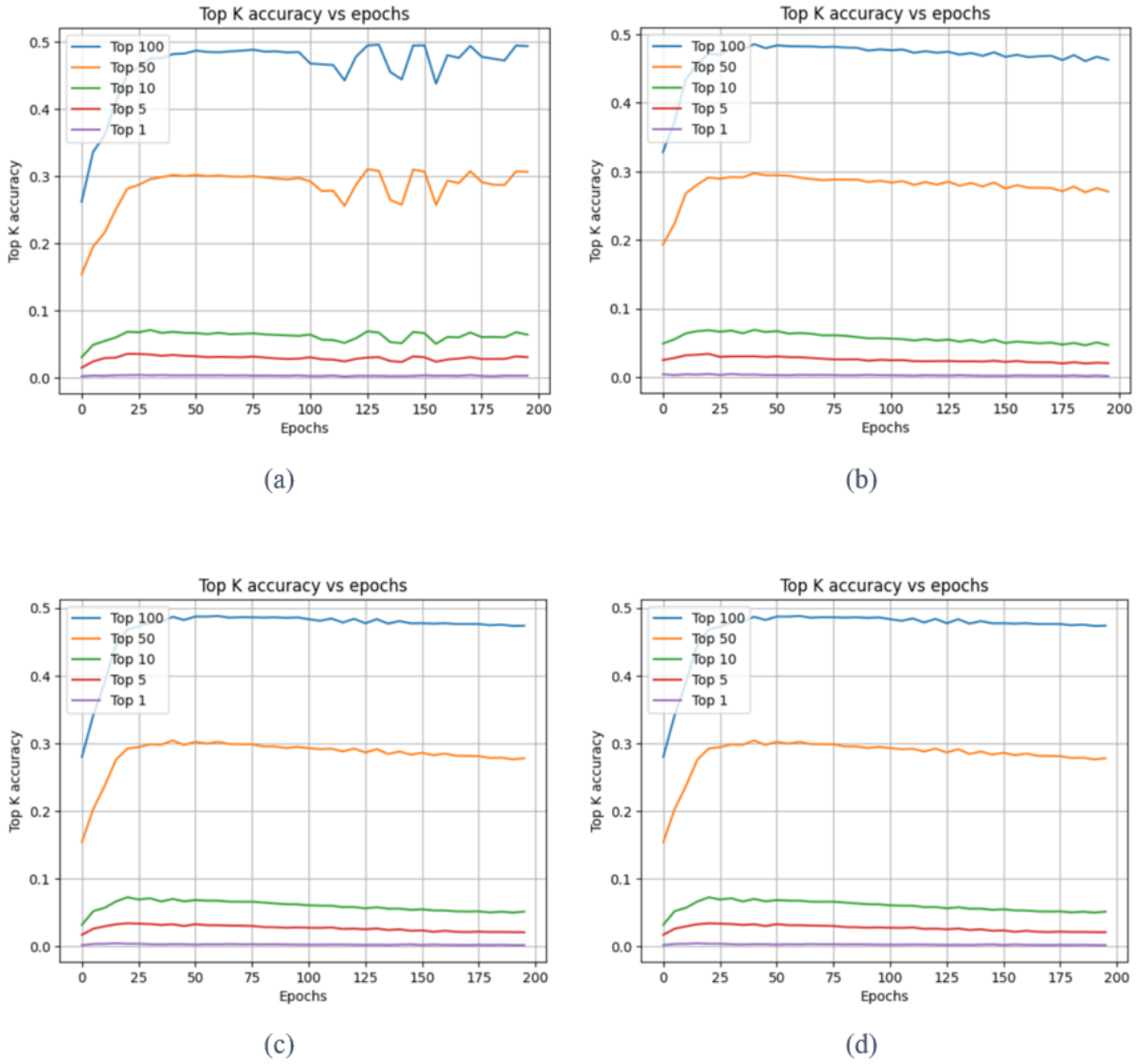

Figure 4.3: Top-k Accuracy curves w.r.t training epoch

On average, IBNS took the least amount of time. NGAME out-performed the other algorithms in all the Recall@k metrics, except for k=100, where it was surpassed by MNS. For Top-k Accuracy, CBNS gave the best results for lower values of k(<=10), and NGAME gave the best results for higher values of k(>=10). Thus, NGAME showed the best results for most of the given metrics. However, it performed poorly in terms of average time spent per epoch. This is due to the clustering step, which is required for mini-batch formation. CBNS also needs to spend more time per epoch, due to its calculating the loss for a large number of negatives, as well as the feature drift. It also requires additional time to enqueue all the movie embeddings of the current mini-batch into the memory bank.

Chapter 5

Conclusions and Future Work

This project investigated the potential of Cross-Batch Negative Sampling (CBNS) and Negative Mining-aware Mini-batching (NGAME) to enhance the accuracy of two-tower recommender systems. We identified the limitations of traditional in-batch negative sampling, particularly regarding scalability, efficiency, and exploration of the negative space.

CBNS emerged as a promising solution, leveraging memory banks to overcome scalability bottlenecks and expedite training. Additionally, NGAME, inspired by extreme classification, prioritizes informative negative examples within each mini-batch, potentially leading to a more robust training signal and improved generalization capabilities.

Integration of NGAME with the CBNS memory bank to select informative negative examples holds immense potential, and could achieve a superior level of negative sampling, encompassing both the diversity offered by CBNS and the focus on informativeness from NGAME.

Future work lies in exploiting the synergy between CBNS and NGAME, by experimenting with different loss functions(eg., triplet loss), distance metrics, bias correction algorithms and embedding similarity measures, during training of two-tower DNN model. Furthermore, NGAME's clustering step, the selection of hard-negatives based on a threshold distance for each user, and computation of the loss can be quite time consuming. The memory requirements of the algorithm can also be optimized; storing the negatives for each user, as well storing additional negatives in case of MNS and CBNS, can lead to the consumption of quite a lot of memory.

The project's implications extend beyond immediate accuracy improvements. Op-

timizing negative sampling techniques paves the way for recommender systems that handle massive datasets, train efficiently, and capture user preferences with exceptional precision. This translates to online platforms delivering hyper-personalized recommendations that not only meet user needs but anticipate them, fundamentally transforming user experiences across various industries. Ultimately, this project serves as a springboard for developing recommender systems that evolve beyond mere suggestion, becoming invaluable companions in our digital journeys.

Bibliography

[1] *3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 1.4.2 documentation.* URL: https://scikit-learn.org/stable/modules/model_evaluation.html#top-k-accuracy-score.

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

[3] Yoshua Bengio and Jean-Sébastien Senécal. "Quick Training of Probabilistic Neural Nets by Importance Sampling". In: *International Conference on Artificial Intelligence and Statistics (AISTATS). PMLR* (2003), pp. 1–9.

[4] Jianqiao Cheng. *Understanding the Two-Tower Model in Personalized Recommendation Systems.* Jan. 2023. URL: https://hackernoon.com/understanding-the-two-tower-model-in-personalized-recommendation-systems.

[5] Dmitry Kalenichenko Florian Schroff and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)* (2015), pp. 815–823. URL: https://doi.org/10.48550/arXiv.1503.03832.

[6] Jieming Zhu Jinpeng Wang and Xiuqiang He. "Cross-Batch Negative Sampling for Training Two-Tower Recommenders". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21). Association for Computing Machinery, New York, NY, USA* (2021), pp. 1632–1636. URL: https://doi.org/10.1145/3404835.3463032.

[7] Deepak Saini et al Kunal Dahiya Nilesh Gupta. "Negative Mining-aware Mini-batching for Extreme Classification". In: *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining (WSDM '23). Association for Computing Machinery, New York, NY, USA* (2023), pp. 258–266. URL: https://doi.org/10.1145/3539597.3570392.

[8] Jay Adams Paul Covington and Emre Sargin. "Deep Neural Networks for YouTube Recommendations". In: *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16). Association for Computing Machinery, New York, NY, USA* (2016), pp. 191–198. URL: https://doi.org/10.1145/2959100.2959190.

[9] *Precision and recall at K in ranking and recommendations.* URL: https://www.evidentlyai.com/ranking-metrics/precision-recall-at-k.

[10] Xinyang Yi. Ji Yang. Lichan Hong. Derek Zhiyuan Cheng. Lukasz Heldt. Aditee Kumthekar. Zhe Zhao. Li Wei and Ed Chi. "Sampling-bias-corrected neural modeling for large corpus item recommendations". In: *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys '19). Association for Computing Machinery, New York, NY, USA* (2019), pp. 269–277. URL: https://doi.org/10.1145/3298689.3346996.

[11] Ji Yang. Xinyang Yi. Derek Zhiyuan Cheng. Lichan Hong. Yang Li. Simon Xiaoming Wang. Taibai Xu and Ed H Chi. "Mixed Negative Sampling for Learning Two-tower Neural Networks in Recommendations". In: *Companion Proceedings of the Web Conference 2020 (WWW '20). Association for Computing Machinery, New York, NY, USA* (2020), pp. 441–447. URL: https://doi.org/10.1145/3366424.3386195.