Multitask Mixture Embeddings for Large Scale Recommendations

Erik Brinkman
erk@fb.com
Facebook AI Applied Research

Dario Garcia-Garcia
dariogg@fb.com
Facebook AI Applied Research

Henry Ehrenberg
TBD
TBD

Yang Liu
yangliuyl@fb.com
Instagram

Ivan Medvedev
ivanm@fb.com
Instagram

Lin Su
linsucai@fb.com
Facebook AI Applied Research

Chandler Zuo
chandlerzuo@fb.com
Facebook AI Applied Research

ABSTRACT

Embedding-based retrieval is a fundamental piece of many real-world, massive-scale recommendation systems. Embedding both users and items in a common space and then applying nearest-neighbors to collect potentially relevant items for a given user is intuitive, flexible and efficient. We propose an extension of this approach in which users are embedded as sets of points, yielding mixture embedding-based retrieval. We describe this method together with some other key ingredients used as a retrieval source in Instagram Explore, one of the largest and most successful unconnected content recommendation surfaces in the world.

ACM Reference Format:
Erik Brinkman, Dario Garcia-Garcia, Henry Ehrenberg, Yang Liu, Ivan Medvedev, Lin Su, and Chandler Zuo. 2020. Multitask Mixture Embeddings for Large Scale Recommendations. In ,. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The abundance of content in today's digital platforms has made recommendation systems an important piece to connect creators and consumers in an efficient manner. This prominent role comes with significant responsibilities and challenges. The decisions made by recommendation systems will to a large extent define the user experience and have ecosystem-level implications because of e.g. the incentives that they impose on content creators or providers. At the same time, these systems have to overcome a series of complex challenges such as how to alleviate the effect of biased data collection (feedback loops) or how to make sure that recommendations are simultaneously relevant and diverse. This last aspect, diversity, is particularly important and complicated, especially when working with unconnected content recommender systems with a huge (in terms of both size and range) inventory. If the system designers are not very careful it is likely that the recommendations will end up focusing on a relatively narrow subset of items, since there are several forces at play that could encourage this behavior, including feedback loops and limited model capacity.

Many large-scale recommendation systems use a two-step architecture. Starting with a very large pool (potentially billions or even more) of candidates to be recommended for a given user, the first step, called retrieval or candidate generation, trims down that pool into a manageable set, typically in the order of a few thousands. The second step performs the final ranking of the surviving candidates. The retrieval stage must thus be very computationally efficient to be able to handle the potentially extremely large candidate pool. Typical alternatives include rule-based retrieval and embedding methods followed by nearest-neighbor (NN) lookups. Modern approximate NN method [3] are extremely efficient, providing both sub-linear complexity and fine-tuned implementations, including GPU-optimized and distributed versions. Together with the flexibility that embeddings provide (they can be constructed from features or "explicitly" from a co-occurrence matrix to optimize a wide range of losses depending on the specifics of the application), this makes embedding based retrieval a key tool when building practical recommendation systems.

The diversity and recall of the final system is heavily determined by the retrieval phase. Items that are dropped during retrieval will not be visible to the ranker and thus won't be presented to the user, while irrelevant items that pass the retrieval stage can be culled by the ranker. One of the main motivations of this work is how to extend embedding-based retrieval methods to improve the diversity of the candidates while maintaining computational efficiency and not requiring too drastic changes to the overall system. To that end we introduce mixture-of-embeddings based retrieval. The idea here is to represent each user as a set of K mixture components in a space $\mathbb{R}^D$ and then retrieve items for that user by performing K nearest neighbor lookups, one around each one of the components. Intuitively, this partially decouples the different interests of a user: in a single embedding model, all of the items that can be retrieved for a user must be close to the user embedding, and thus by virtue of the triangle inequality also not too far from each other. In a mixture model, each mixture component can capture items in

IRS 2020, Workshop on Industrial Recommendation Systems
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

arbitrarily distant parts of the embedding space, dropping the aforementioned constraint on retrieved item similarity. Of course these effects are not that clear cut in very high dimensional spaces and one might argue that increasing the dimension of the embedding space gives a more direct solution to increase the expressiveness of the models. We'll also discuss how moving from a single embedding to a K-mixture is different to just increasing the dimension of the embedding space to KD (spoiler: the non-linearity in the total loss function that combines the distances from an item to each one of the embeddings) and why this approach might be preferable in many scenarios.

Another challenge when building recommendation systems is defining a proper objective function. Many different aspects contribute to this challenge:

- Many times the user experience can't be accurately measured by a single observable magnitude: should we care about clicks, time spent, other types of engagement signals, ...
- Even if there's a gold "business metric", that metric might not be amenable to direct optimization (e.g. daily users)
- Sometimes the best metrics don't have great coverage (e.g. explicit surveys)
- Feedback loops make optimization of any single metric potentially dangerous unless a lot of care is put into thinking through side effects

As a consequence, many times there will not be a single objective but a set of proxy metrics which are correlated with the final goal. For this reason we focus also on multi-task models that can learn simultaneously from all these signals, contributing to alleviate the feedback loop issues and enabling flexible trade-offs between coverage and signal quality.

In this paper we bring together all the aforementioned ideas to develop a multitask, mixture embedding based retrieval system that can achieve both the scale and performance necessary to help Instagram Explore, one of the largest and most popular unconnected content recommendation surfaces in the world.

2 MIXTURE EMBEDDINGS FOR GRAPHS

Let's consider a bipartite graph $G = (S, D, E)$ where S is the set of *source* nodes, D is the set of *destination* nodes and E is a set of edges between S and D.[1] In a general embedding model, each entity $i \in S \cup D$ is represented through an embedding U_i, while a scoring function $score(U_s, U_d)$ computes a similarity score for each pair of entities $s\ inS$, $d \in D$ with the goal of maximizing the score for $(s, d) \in E$ and minimizing it for $(s, d) \notin E$. In the mixture embedding model, $U_s \in \mathbb{R}^{C_s \times n}$, $U_d \in \mathbb{R}^{C_d \times n}$, where each row represents a different mixture component with dimension n, and C_s, C_d are the number of mixture embeddings on each side.

Each component embedding is then defined with a standard embedding model. Let $U_{sc}, U_{dl} \in \mathbb{R}^n$ denote the cth and the lth embedding component for node s and d respectively. In this paper we use standard cosine similarity

$$sim(U_{sc}, U_{dl}) = \frac{U_{sc}^T U_{dl}}{||U_{sc}||||U_{dl}||}.$$

[1]We abuse the naming source and destination even if the graph is not directed

With our component embedding model defined, the mixture scoring function is simply the composition of two functions

$$score(U_s, U_d) = pool(product(U_s, U_d)).$$

product computes a similarity kernel matrix $M \in \mathbb{R}^{C_s \times C_d}$ where $M_{c,l} = sim(U_{sc}, U_{dl})$. *pool* pools the similarity matrix into a final score, and is one of the main design considerations of the overall loss function as it defines the semantic meaning of mixtures in the model. The mixture embedding model is simply a restriction on the class of scoring functions, providing an inductive bias over the explanations for an interaction between two entities.

2.0.1 Pooling Functions. The most obvious pooling function is simply the max pool. This function says: two entities need to have one similar component to be scored highly. While seeming fairly natural at first, the highly nonlinear cost surface makes this hard to optimize. A second order approach would be to instead use the smooth maximum function with temperature γ

$$\text{smoothmax}(M; \gamma) = \frac{\sum_{c,l} M_{c,l} e^{M_{c,l}/\gamma}}{\sum_{c',l'} e^{M_{c',l'}/\gamma}}. \quad (1)$$

When $\gamma \longrightarrow 0$, Eqn. (1) approaches the maximum cosine similarity between all pairs of components of the node pair.

2.0.2 Regularized Smoothmax. We further define a regularized smoothmax

$$\text{regsmoothmax}(M; \theta, \gamma) = \sum_{c,l} M_{c,l} \left(\theta + (1 - \theta) \frac{e^{M_{c,l}/\gamma}}{\sum_{c',l'} e^{M_{c',l'}/\gamma}} \right). \quad (2)$$

where θ is the regularization hyperparameter. $\theta = 0$ yields the standard smooth max function. Larger values of γ and θ push such the smooth max similarity towards the average cosine similarity among all pairs of components. Section 3.1 discusses more details on how these parameters are chosen in training.

2.1 Nonlinear Similarity Function

The idea of using multiple embeddings to represent nodes in a graph is not new. [1] refers to this as the "personal decomposition" problem in the context of representing people's social network, while [5] refers to this as representing "node polysemy". In addition to the advantages of mixture embeddings mentioned in these papers, our motivation behind this framework is mainly to improve diversity. In a large-scale recommendation system with a huge pool of retrieval candidates, it is typical for a user to be interested in different candidates with very different semantic meaning. In an embedding space with Euclidean distance, items close to a single user have to be close to each other as dictated by the triangular inequality. The distance metric in Eqn. (2) breaks free of such structure. By introducing nonlinearity in the distance metric, the underlying embedding space allows multiple items to be close to a single user while still be distant towards each other.

The nonlinearity in distance construction of Eqn. (2) is essential in this framework. Using a variety of datasets we have compared our mixture embedding model against a benchmark model that embeds each node with a single vector with the dimensionality

$\mathbb{R}^{C_{s/d}n}$, using various link retrieval and node label prediction tasks. We have found our framework to significantly outperform the benchmark setting, even though the two models have the same degrees of freedom.

We want to highlight one important difference between our framework and the standard finite mixture modeling framework. Standard finite mixture models include mixture weight parameters that determine the distribution across mixture components. The hierarchical probabilistic formulation in turn determines the distance metric, the loss function and the training algorithm. Such a "generative modeling" framework is followed by [5]. Our framework's distance metric, Eqn. (2), is independent of such mixture weight parameters. Training algorithm in Section 3.1 is also only contigent on the distance metric. The framework itself still accepts the notion of mixture weight parameters, as we will see in Section 3.4, but those weights are calculated in a separate post processing step to facilitate nearest neighbor retrieval. From this angle, we argue that our framework is a "discriminative model" that doesn't yield a coherent probabilistic story. Nevertheless, our framework offers significant computational advantage. Compared to the hierarchical model in [5], our training algorithm doesn't require expensive conditional sampling through nested loops, which is a typical artifact in hierarchical modeling. Such computational efficiency makes this framework directly applicable to graphs with billions edges. More details of the algorithm are discussed in the next section.

3 MIXTURE EMBEDDING BASED RETRIEVAL

Because the main motivation of this work is to improve the candidate generation step in large-scale recommendation systems, we designed the model to focus on the retrieval performance from every aspect.

3.1 Loss Function

The model is trained using a margin ranking loss (MRL) function, which has been proved superior in many of our retrieval tasks. Given a positive edge $e = (s, d) \in E$, we construct a set of negative edges $\{e' = (s, d') : d' \in N(s)\}$, where $N(s)$ is the negative sampling function that encompasses the negative sample size and the pool of potential negative examples. Note that $N(s)$ is a function of the source node s, and we will discuss the reason in Section 3.2. For each pair of positive and negative edges, the corresponding loss is:

$$\begin{aligned} l(e, e') = &\max(\text{smoothmax}(product(U_s, U_{d'}); \gamma) + \lambda \\ &-\text{regsmoothmax}(product(U_s, U_d); \theta, \gamma), 0), \end{aligned} \tag{3}$$

where λ is the margin hyperparameter. Eqn. (3) applies different pooling functions for the positive and negative examples in order for regularization. By the definition of regsmoothmax, scores of positive examples are pooled more towards the average cosine similarity among mixture components compared to scores of negative examples. In back-propagation, positive examples influence more on all mixture component pairs, while negative examples influence mostly on the closest mixture component pair. Our intuition is that mixture embeddings of the same entity are similar to each other because they represent different aspect of the same entity. Regularization encourages such similarity structure. Because our training randomly initializes all embeddings, such regularization helps the training algorithm to converge to a favorable local optimum. Our experiments have shown such regularization significantly improves retrieval.

3.2 Multi-task Model

We have compressed the notations of tasks for simplicity in the discussions above. In order to leverage collective information from multiple proxy objectives, we actually focus on multi-task models. In multi-task training, the edge set E consists of the union of edges for individual tasks $E = \cup_{T \in \mathscr{T}} E_T$, and E_T is composed by source node type S_T and destination node type D_T, where S_T and D_T can be the same, and different tasks should share the same set of node types. In this setting, nodes from all types are embedded in the same space, which enables us to conduct retrieval between any two of them.

We train this multi-task model by implementing an additive loss from all tasks. In addition, our framework allows for the loss of each training edge to be associated with a weight $\omega(e, e')$. The form of this weight function is user defined. In multi-task training, we apply pre-defined weights for all edges belonging to the same task. Such weight function can be extended. For example, one may use a function of task weight, source and destination node degree that can adjust the importance of high-degree or low-degree nodes in the training. Given this, our model minimizes the following loss function:

$$\mathcal{L} = \sum_{T \in \mathcal{T}} \sum_{(s,d) \in E_T} \sum_{d' \in N(s)} \omega(e, e') l(e, e'). \tag{4}$$

In the negative smapling function $N(s)$, in order to make the negative sample hard enough for the loss to learn, we ensure that d' come from the same node type as d, so that (s, d') is a potential edge within the same task. For destination nodes of different types from another task, they are potentially embedded into different subspaces far away from the subspace of s. Therefore, for d' of the same type as d and d'' of different type, distances between (s, d'') are usually further than (s, d') naturally.

In Eqn. (4), each component in this loss function is computed based on one positive edge and one negative edge, and the final loss is simply additive. Therefore, our model is highly scalable utilizing a distributed training system. We can partition the entire bi-partite graph from both the source side and the destination side in the same way as in [4].

3.3 One-sided Negative Sampling

In our training data, the end users of the recommendation system usually correspond to a node type on one side in the bi-partite graph, while the content candidates appear on the other side. In most cases, we only care about directionally retrieving one type of node from another but not the opposite direction. For example, we are mostly interested in recommending unconnected contents to users instead of sourcing potential users to a content. We enforce such focus on directional retrieval by one-sided negative sampling during training. We prepare our input graph so that the users in downstream retrieval tasks always appears as the source node. In

Eqn. (3), we always sample from the destination side, and connect the sampled destination node d' with original source s of the positive edge to form a negative edge.

3.4 Mixture Weights

Through computing mixture distance using smooth-max in (2), mixture embeddings of one node forms a set without certain order. Therefore, training step does not explicitly estimate mixture component weights. In order to enable sub-linear retrieval, a post processing step is implemented to compute mixture weights after training is done. Mixture weights are adaptively calculated based on a set of tasks $\mathcal{T}'$ defined on all or a subset of all training tasks $\mathcal{T}$. Typically, $\mathcal{T}'$ is the whole set of training tasks, but in certain applications we also let $\mathcal{T}'$ include only one training task in order to compute task specific mixture weights. Denote mixture weights of source node s by $\{w_{sc}, 1 \le c \le C_s\}$. These weights are computed by

$$w_{sc} \propto \sum_{\{T \in \mathcal{T}'\}} \sum_{\substack{(s,d)\in E_T \\ \text{or } (d,s)\in E_T}} \sum_{l=1}^{C_d} e^{M_{c,l}}/\gamma_T, \tag{5}$$

and normalized by $\sum_{c=1}^{C_s} w_{sc} = 1$. Comparing to (2), (5) does not include the regularization θ hyperparameter, and the smooth-max temperature γ_T can be chosen to be different from γ used in training. The mixture weights of destinaton nodes are defined in the same way.

3.5 Round-robin KNN

Once we have mixture embeddings trained and their corresponding weights computed, we are empowered to conduct nearest-neighbor search to obtain a diverse candidate pool per node. For node s with C_s mixture embeddings, if we are interested in retrieving the top K nearest neighbors from the set of destination nodes D_t with C_d mixture embeddings, we assign each candidate node d a score through an aggregation in the following way. First, for each component in the source mixture embedding we obtain the top $K * C_d$ nearest embeddings from the overall set of destination mixture embeddings $\{U_{cl} : d \in D_t, l = 1, \dots, C_d\}$. We choose the top $K * C_d$ nearest neighbors in order to guarantee we have at least K nearest neighbors after aggregation and deduplication. We obtain the corresponding pairwise scores $\alpha_{s,d}(c,l)$ between the cth embedding of s and the lth embedding of d. If the lth embedding of d does not appear in the top $K * C_d$ nearest embeddings in this step, then it is scored as 0. Leveraging modern approximate NN techniques (e.g. [3]), this step can be done very efficiently. Assuming the larger α is, the closer the two individual embeddings are, then for each source embedding we aggregate the score of each destination node by summing the product of destination mixture weight and NN score, which is $\alpha_{s,d}(c) = \sum_{l=1}^{C_d} w_{dl}\alpha_{s,d}(c,l)$. We assign the rank of destination node d for the cth embedding of node s by ordering $\alpha_{s,d}(c)$ in descending order, and denote it by $r_{s,d}(c)$. We finally aggregate the scores of retrieved destination nodes for each source embedding up to each source node by computing $\alpha_{s,d} = \max_{c=1}^{C_s} w^{sc}/r_{s,d}(c)$. The nodes with highest $\alpha_{s,d}$ are the nearest neighbors for s. We do two rounds of aggregations in this process: aggregate the scores of destination mixture embeddings first, then aggregate the scores of source mixture embeddings. In practice we choose to use summation and maximum respectively as aggregation functions in each step.

4 REAL-WORLD APPLICATION

4.1 Implementation

We implement the techniques described in the previous sections on top of Pytorch-BigGraph (PBG) [4], a state-of-the-art library for massive multi-relation embeddings [2]. Three specific changes need to be made to the default options supported by PBG to support milti-task mixture embedding training. First, we implement a custom comparator that returns our mixture *score* function instead of the built in score functions to PBG. Second, we enable support of heterogeneous embedding dimension for each entity. Mixture embeddings can be represented as a single large embedding formed by each component concatenated together, and while it's possible to define PBG's dimension as the largest such dimension, in practice we found this considerably slower even if there were only a small portion of entities with "truncated" embeddings. Finally, we enable single sided negative sampling as described in Section 3.3.[3]

4.2 Instagram Explore

Instagram's Explore tab [2] presents users with unconnected content, i.e. content from creators that the user doesn't already follow. Over half of the Instagram community visits Explore every month to discover new photos, videos, and Stories relevant to their interests out of billions of high quality pieces of media on that eligible inventory.

Multitask mixture embedding-based retrieval is actively used in Instagram Explore as a source of candidates for the ranking stage, co-existing with several other sources mostly based on account similarity [2]. We explode each account into two logical entities: a "consumer" and a "creator" persona. Each consumer persona is modeled as a 5 component mixture, while each creator is represented by a single component (we are currently experimenting with having mixtures on both sides). We consider different types of "edges" between consumers and creators corresponding to different types of engagement (e.g. clicks, likes) to construct a bi-partite graph (i.e. consumer personas are only connected to creator personas). To perform the actual candidate generation we apply the retrieval techniques mentioned in Section 3 to each consumer persona to collect relevant accounts from which to source content that is then passed on to the ranking stage.

5 CONCLUSIONS

We have presented a type of mixture embedding models that can extend the expressive power of standard embedding models while being able to scale to extremely large datasets, together with the ingredients necessary to make good use of these mixture embeddings in the context of candidate generation for large scale recommendation systems based on our experience with Instagram's Explore recommender.

[2]https://github.com/facebookresearch/PyTorch-BigGraph
[3]The second and third changes have already been upstreamed to PBG.

Future research includes ways of making the most efficient possible use of the expressive capacity provided by the multiple mixture components per user while adapting to the wide range of data volumes across users typical in recommender systems. For example, this could include dynamic adjustment of the number of components for each user so that e.g. power users or users with very diverse interests have a larger number of components than more occasional or "focused" users. Another interesting line is quantifying the uncertainty of the components and the density of the embedding space to properly allocate the total candidate quota across the components.

REFERENCES

[1] Alessandro Epasto and Bryan Perozzi. Is a single embedding enough? learning node representations that capture multiple social contexts. In *The World Wide Web Conference*, pages 394–404, 2019.

[2] Facebook. Powered by AI: Instagram's Explore recommender system. https://ai.facebook.com/blog/powered-by-ai-instagrams-explore-recommender-system, 2019.

[3] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

[4] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287*, 2019.

[5] Ninghao Liu, Qiaoyu Tan, Yuening Li, Hongxia Yang, Jingren Zhou, and Xia Hu. Is a single vector enough? exploring node polysemy for network embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 932–940, 2019.