Embedding-based Retrieval in Facebook Search

Jui-Ting Huang
juiting@fb.com
Facebook Inc.

Ashish Sharma
ashishsharma@fb.com
Facebook Inc.

Shuying Sun
shuyingsun@fb.com
Facebook Inc.

Li Xia
xiali824@fb.com
Facebook Inc.

David Zhang
shihaoz@fb.com
Facebook Inc.

Philip Pronin
philipp@fb.com
Facebook Inc.

Janani Padmanabhan
jananip@fb.com
Facebook Inc.

Giuseppe Ottaviano
ott@fb.com
Facebook Inc.

Linjun Yang*
yang.linjun@microsoft.com
Microsoft

ABSTRACT

Search in social networks such as Facebook poses different challenges than in classical web search: besides the query text, it is important to take into account the searcher's context to provide relevant results. Their social graph is an integral part of this context and is a unique aspect of Facebook search. While embedding-based retrieval (EBR) has been applied in eb search engines for years, Facebook search was still mainly based on a Boolean matching model. In this paper, we discuss the techniques for applying EBR to a Facebook Search system. We introduce the unified embedding framework developed to model semantic embeddings for personalized search, and the system to serve embedding-based retrieval in a typical search system based on an inverted index. We discuss various tricks and experiences on end-to-end optimization of the whole system, including ANN parameter tuning and full-stack optimization. Finally, we present our progress on two selected advanced topics about modeling. We evaluated EBR on verticals[1] for Facebook Search with significant metrics gains observed in online A/B experiments. We believe this paper will provide useful insights and experiences to help people on developing embedding-based retrieval systems in search engines.

CCS CONCEPTS

• **Information systems** → **Retrieval models and ranking**; *Search engine architectures and scalability*; • **Computing methodologies** → *Learning latent representations.*

KEYWORDS

Embedding, deep learning, search, information retrieval

ACM Reference Format:
Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based Retrieval in Facebook Search. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403305

*This work was performed when the author was at Facebook.
[1]In Facebook search, verticals are based on result types, e.g., people, page, group, etc.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403305

1 INTRODUCTION

Search engines have been an important tool to help people access the huge amount of information online. Various techniques have been developed to improve search quality in the last decades, especially in web search engines including Bing and Google. Since it is difficult to accurately compute the search intent from query text and represent the semantic meaning of documents, search techniques are mostly based on various term matching methods [1], which performs well for the cases that keyword match can address. It still remains a challenging problem for *semantic matching* [12], which is to address desired results that are not exact match of the query text but can satisfy users' search intent.

In the last years, deep learning has made significant progress in speech recognition, computer vision, and natural language understanding [10]. Among them *embedding*, which is also called *representation learning*, has been proven to be successful techniques contributing to the success [2]. In essence, *embedding* is a way to represent a sparse vector of ids as a dense feature vector, which is also called *semantic embedding* in that it can often learn the semantics. Once the embeddings are learned, it can be used as a representation of query and documents to apply in various stages of a search engine. Due to the huge success of this technique in other domains including computer vision and recommendation system, it has been an active research topic in information retrieval community and search engine industry as the next generation search technology [13].

In general, a search engine comprises a recall layer targeting to retrieve a set of relevant documents in low latency and computational cost, usually called *retrieval* , and a precision layer targeting to rank the most desired documents on the top with more complex algorithms or models, usually called *ranking*. While embeddings can be applied to both layers, it usually has more opportunities to leverage embeddings in the retrieval layer, since it is at the bottom of the system which is often the bottleneck. The application of embeddings in retrieval is called *embedding-based retrieval* or *EBR* for short. Briefly, embedding-based retrieval is a technique to use embeddings to represent query and documents, and then convert

the retrieval problem into a nearest neighbor (NN) search problem in the embedding space.

EBR is a challenging problem in search engines because of the huge scale of data being considered. Different from ranking layers which usually takes hundreds of documents into consideration per session, retrieval layer needs to process billions or trillions of documents in the index of a search engine. The huge scale imposes challenges on both training of embeddings and serving of embeddings. Second, different from embedding-based retrieval in computer vision tasks, search engine usually needs to incorporate both embedding-based retrieval and term matching based retrieval together to score documents in the retrieval layer.

Facebook search, as a social search engine, has unique challenges compared with traditional search engines. In Facebook search, the search intent does not only depend on query text but is also heavily influenced by the user who is issuing the query and the context where the searcher is. Because of this, embedding-based retrieval in Facebook search is not a text embedding problem, as is actively researched in the IR community [13]. Instead it is a more complex problem that requires understanding of text, user, and the context altogether.

To deploy embedding-based retrieval in Facebook search, we developed approaches to address challenges on *modeling*, *serving*, and *full-stack optimization*. In modeling, we proposed *unified embedding*, which is a two sided model where one side is search request comprising query text, searcher, and context, and the other side is the document. To effectively train the model, we developed approaches to mine training data from search log and extract features from searcher, query, context, and documents. For fast model iteration, we adopted a recall metric on an offline evaluation set to compare models.

Building retrieval models for search engine has its unique challenges, such as how to build a representative training task for models to learn effectively and efficiently. We investigated two different directions, *hard mining* to address the challenge of representing and learning retrieval tasks effectively, as well as *ensemble embedding* to divide the model in multiple stages where each stage has different recall and precision tradeoff.

After the model is developed, we need to develop ways to effectively and efficiently serve the model in the retrieval stack. While it is straightforward to build a system combining the candidates from existing retrieval and embedding KNN, we found it is suboptimal because of several reasons: 1) it has huge performance cost from our initial experiment; 2) there is high maintenance cost because of dual index; 3) the two candidate sets might have significant overlap which makes it inefficient overall. Thereafter, we developed a hybrid retrieval framework to integrate embedding KNN and Boolean matching together to score documents for retrieval. To this purpose, we employed Faiss [9] library for embedding vector quantization and integrated it with inverted index based retrieval to build a hybrid retrieval system. Besides addressing the above challenges, this system has two main advantages: 1) it enables the joint optimization of embedding and term matching to address Search retrieval problem; 2) it supports embedding KNN constrained by term matching, which not only helps address the system performance cost issue but also improves the precision of embedding KNN results.

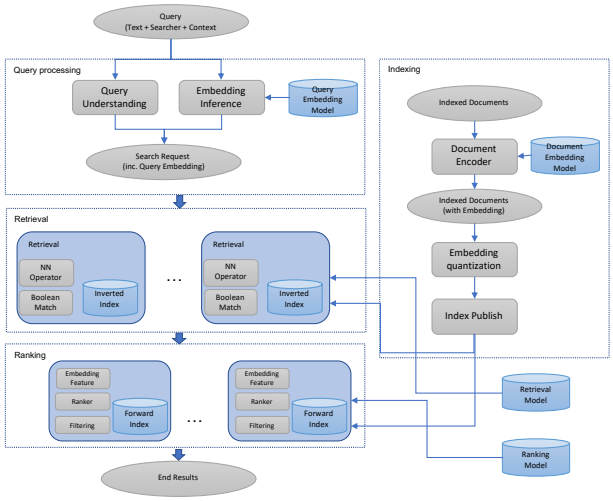

Figure 1: Embedding Based Retrieval System Overview

Search is a multi-stage ranking system where retrieval is the first stage, followed by various stages of ranking and filtering models. To wholly optimize the system to return those new good results and suppress those new bad results in the end, we performed later-stage optimization. In particular, we incorporated embeddings into ranking layers and built a training data feedback loop to actively learn to identify those good and bad results from embedding-based retrieval. Figure 1 is an illustration of embedding-based retrieval system. We evaluated EBR on verticals for Facebook Search with significant metrics gains observed in online A/B experiments.

The paper is organized as follows. We start with modeling to present our solutions about loss function, model architecture, training data and feature engineering in Section 2 and Section 3. Next we discuss about details on model serving and system implementation in Section 4. We discuss the techniques we developed on later-stage optimization to unleash the power from embedding-based retrieval end to end in Section 5. Finally, we dedicate Section 6 to selected topics on advanced modeling techniques, followed by conclusions in Section 7.

2 MODEL

We formulate the search retrieval task as a recall optimization problem. Specifically, given a search query, its target result set $T = \{t_1, t_2, ...t_N\}$, and top K results returned by a model, $\{d_1, d_2, ...d_K\}$, we want to maximize recall by the top K results,

$$recall@K = \frac{\sum_{i=1}^{K} d_i \in T}{N}. \quad (1)$$

The target results are the documents related to the given query based on certain criteria. For example, it could be results with user clicks, or relevant documents based on human rating.

We formulate the recall optimization as a ranking problem based on distances computed between a query and documents. The query and documents are encoded with a neural network model into dense vectors, on which we use cosine similarity as the distance metric. We propose to use triplet loss [14] to approximate the recall

objective to learn the neural network encoder, which is also called embedding model.

While semantic embedding is commonly formulated as text embedding problem in information retrieval, it is insufficient for Facebook search, which is a personalized search engine that considers not only text query but also searcher's information as well as the context in a search task to satisfy users' personalized information need. Taking people search as an example, while there might be thousands user profiles named "John Smith" on Facebook, the actual target person that a user searches for with query "John Smith" is likely to be their friends or acquaintances. To model this problem, we propose *unified embedding* which considers not only text but also user and context information in deriving embeddings.

2.1 Evaluation Metrics

While our end goal is to deliver quality improvement end to end through online A/B test, it is important to develop offline metrics to quickly evaluate model quality before online experiments and isolate problems from complicated online experiment setup. We propose to run KNN search in the whole index and then use *recall@K* as defined in equation 1 as the model evaluation metric. In particular, we sampled 10000 search sessions to gather the query and target result set pairs for the evaluation set and reported averaged *recall@K* over 10000 sessions.

2.2 Loss Function

For a given triplet $(q^{(i)}, d_+^{(i)}, d_-^{(i)})$, where $q^{(i)}$ is a query, $d_+^{(i)}$ and $d_-^{(i)}$ are the associated positive and negative documents, respectively, the triplet loss is defined as

$$L = \sum_{i=1}^{N} \max(0, D(q^{(i)}, d_+^{(i)}) - D(q^{(i)}, d_-^{(i)}) + m), \tag{2}$$

where $D(u, v)$ is a distance metric between vector u and v, m is the margin enforced between positive and negative pairs, and N is the total number of triplets selected from the training set. The intuition of this loss function is to separate the positive pair from the negative pair by a distance margin. We found that tuning margin value is important – the optimal margin value varies a lot across different training tasks, and different margin values result in 5-10% KNN recall variance.

We believe that using random samples to form negative pairs for the triplet loss can approximate the recall optimization task. The reason is as follows. If we sample n negatives for each one positive in the training data, the model will be optimizing for recall at *top one position* when the candidate pool size is n. Assuming the actual serving candidate pool size is N, we are approximately optimizing recall at top $K \approx N/n$. In Section 2.4, we will verify this hypothesis and provide comparisons of different positive and negative label definitions.

2.3 Unified Embedding Model

To learn embeddings that are optimizing the triplet loss, our model comprises three major components: a query encoder $E_Q = f(Q)$ which produces a query embedding, a document encoder $E_D = g(D)$ which produces a document embedding, and a similarity function $S(E_Q, E_D)$ which produces a score between query Q and document D. An encoder is a neural network which transforms an input into a low-dimensional dense vector, also known as embedding. In our model, these two encoders $f(\cdot)$ and $g(\cdot)$ by default are two separate networks but have the option of sharing part of the parameters. As for similarity function, we choose cosine similarity as it is one of the commonly used in embedding learning [7]:

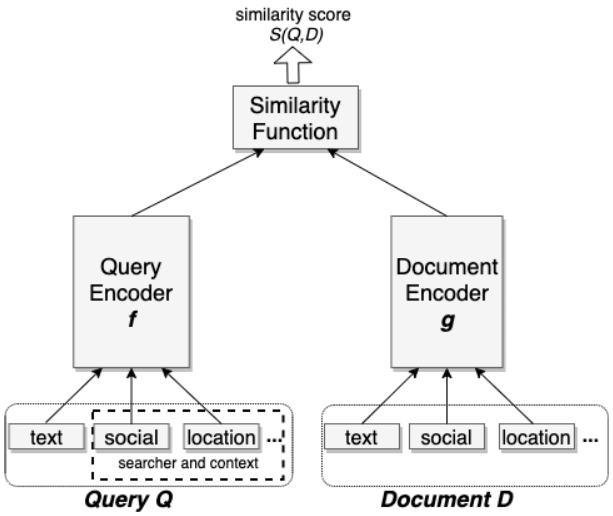

Figure 2: Unified Embedding Model Architecture

$$S(Q, D) = \cos(E_Q, E_D) = \frac{\langle E_Q,\ E_D \rangle}{\|E_Q\| \cdot \|E_D\|}. \tag{3}$$

The distance to be used in the loss function in Equation 2 is hence the cosine distance defined as $1 - \cos(E_Q, E_D)$.

The inputs to the encoders are what distinguishes unified embedding from conventional text embedding model. Unified embedding encodes textual, social and other meaningful contextual features to represent query and document, respectively. For example, for query side we can include searcher location and their social connections, whereas for the document side we can include aggregated location and social clusters about a Facebook group by taking groups search as an example.

Most of features are categorical features of high cardinality, which could be either one-hot or multi-hot vectors. For each categorical feature, an embedding look-up layer is inserted to learn and output its dense vector representation before feeding into encoders. For multi-hot vectors, a weighted combination of multiple embeddings is applied for the final feature-level embedding. Figure 2 illustrates our unified embedding model architecture, and we will discuss more about feature engineering in Section 3.

2.4 Training Data Mining

Defining positive and negative labels for a retrieval task in a search ranking system is a non-trivial problem. Here we compared several options based on the model recall metric. For negative, we experimented with the following two options of negatives in our initial study, while using click as positive:

- *random samples*: for each query, we randomly sample documents from the document pool as negatives.
- *non-click impressions*: for each query, we randomly sample those impressed but not clicked results in the same session as negatives.

The model trained using non-click impressions as negative has significantly worse model recall compared to using random negative: absolute 55% regression in recall for people embedding model. We believe it is because these negatives bias towards hard cases which might match the query in one or multiple factors, while the majority of documents in index are easy cases which do not match the query at all. Having all negatives being such hard negatives will change the representativeness of the training data to the real retrieval task, which might impose non-trivial bias to the learned embeddings.

We also experimented with different ways of mining positives and have interesting findings as follows:

- *clicks*: it is intuitive to use clicked results as positives, since clicks indicates users' feedback of the result being a likely match to users' search intent.
- *impressions*: the idea is that we treat retrieval as an approximation to ranker but can execute fast. Thereafter, we want to design the retrieval model to learn to return the same set of results that will be ranked high by the ranker. In this sense, all results shown or impressed to the users are equally positive for retrieval model learning.

Our experimental results showed that both definitions are equally effective; models trained using click vs impressions, given the same data volume, resulted in similar recalls. Additionally, we experimented with augmenting click-based training data with impression-based data, however we did not observe additional gain over the click-based model. It showed that adding impression data does not provide additional value, and the model does not benefit from increased training data volume either.

Our above study suggested that using click as positive and random as negative can provide a reasonable model performance. On top of it, we further explored hard mining strategies to improve the model's ability of differentiating between similar results. We will present more details in Section 6.1.

3 FEATURE ENGINEERING

One of the advantages of unified embedding model is that it can incorporate various features other than text to improve the model performance. We observed consistently across different verticals that unified embedding is more effective than text embedding. For example, there are +18% recall improvement when switching from text to unified embeddings for events search, and +16% recall improvement for groups search. The effectiveness of unified embeddings highly depends on the success of identifying and crafting informative features. Table 1 shows the incremental improvement by adding each new feature category to the group embedding model (with text features as baseline). In this section we discuss several important features that contributed to the major model improvements.

Text features. Character n-gram [7] is a common approach to represent text for text embedding. Its advantages compared to word n-grams are two folds. First, due to its limited vocabulary size, the embedding lookup table has a smaller size and can be learned more effectively during training. Second, the subword representation is robust to out-of-vocabulary problem the we encounter for both query (e.g. spelling variations or errors) and document sides (due to large content inventory in Facebook). We compared models trained with character n-grams vs word n-grams and found the former can yield a better model. However, on top of character trigrams, including word n-gram representations additionally provides small but consistent model improvement (+1.5% recall gain). Note that since the cardinality of word n-grams is usually very high (e.g. 352M for query trigrams), hashing is needed to reduce the size of embedding lookup table. But even with the downside of hash collision, adding word n-ngrams still provides extra gain.

For a Facebook entity[2], the main field to extract text features is name for people entities or title for non-people entities. Compared to Boolean term matching techniques, we found embeddings trained with purely text features are particularly good at addressing the following two scenarios:

- Fuzzy text match. For example, the model is able to learn to match between query "kacis creations" and the *Kasie's creations* page while the term-based match cannot.
- Optionalization. In the case of query "mini cooper nw", the model can learn to retrieve the expected group *Mini cooper owner/drivers club* by dropping "nw" for an optional term match.

Location features. Location match is advantageous in many search scenarios such as searching for local business/groups/events. In order for embedding models to consider locations in generating the output embeddings, we added location features into both query and document side features. For query side, we extracted searcher city, region, country, and language. For document side, we added publicly available information, such as explicit group location tagged by the admin. Together with text features, the model was able to successfully learn implicit location match between query

Table 1: Group Embedding Improvement with Feature Engineering

Unified Embedding	Abs. Recall Gain
+ location features	+ 2.20%
+ social embedding features	+ 1.77%

Table 2: Top Similar Groups Before and After Adding Location Embeddings

Searcher Location: Louisville, Kentucky
Query: *"equipment for sale"*

	Text Model		Text + Location Model
1	equipment for sale	1	Kentucky Farm Equipment for sale
2	EQUIPMENT FOR SALE WORLDWIDE	2	Pre-Owned Farm Equipment for Sale in KY, Southern Indiana and Tennessee
3	EQUIPMENT SALE	3	Farm Equipment For Sale In Ky
4	Equipment for Sale or Wanted	4	Central Ky Farm Equipment For Sale
5	Musical Equipment For Sale or Trade	5	sed Farm Equipment for sale or trade East Ky.
6	Used Heavy Equipment For Sale in US	6	Farm Equipment KY for Sale
7	Sale equipment	7	kentucky hay and farm equipment for sale

[2]Facebook entity includes people (profile), group, page, and event.

and results. Table 2 shows a side-by-side comparison of top similar documents returned by text embedding model versus text + location embedding model for groups search. We can see the model with location features can learn to fuse location signals into embeddings, ranking documents that has the same location as the searcher who is from Louisville, Kentucky to higher positions.

Social embedding features. To leverage the rich Facebook social graph to improve unified embedding model, we trained a separate embedding model to embed users and entities based on the social graph. This can help incorporate the comprehensive social graph into a unified embedding model which may not have full social graph information otherwise.

4 SERVING

4.1 ANN

We deployed an inverted index based ANN (approximate near neighbor) search algorithms to our system because of the following advantages. First, it has smaller storage cost due to the quantization of embedding vectors. Second, it is easier to be integrated into the existing retrieval system which is based on inverted index. We employed Faiss library [9] to quantize the vectors and then implemented the efficient NN search in our existing inverted table scanning system.

There are two major components for the embedding quantization, one is the *coarse quantization* which quantizes embedding vectors into coarse clusters typically through *K-means* algorithm, and the other is *product quantization* [8] which does a fine-grained quantization to enable efficient calculation of embedding distances. There are several important parameters we need to tune:

- *Coarse quantization.* There are different algorithms for coarse quantization. It is useful to compare between IMI [11] and IVF [15] algorithms. And it is important to tune number of coarse clusters *num_cluster*, which will affect both perf and recall.
- *Product quantization.* There are multiple variants of product quantization algorithms, including vanilla PQ, OPQ, PQ with PCA transform. And number of bytes for PQ *pq_bytes* is an important parameter to tune.
- *nprobe. nprobe* is the parameter to decide how many clusters will be assigned to the query embedding, which will further decide how many coarse clusters will be scanned. This parameter will affect the perf and recall.

We built an offline pipeline to efficiently tune these parameters. Besides, we needed to run online experiment to decide the final setting from the selected candidates based on offline tuning. Below we will share the tricks and learnings we got from ANN tuning.

- *Tune recall against number of scanned documents.* Initially we compared recall of different coarse quantization algorithms for the same setting of *num_cluster* and *nprobe*. However, from more data analysis we found that the clusters are imbalanced, especially for IMI algorithm – around half of clusters only got a few samples. This would cause the number of scanned documents different for the same setting of *num_cluster* and *nprobe*. Therefore, we employed the number of scanned documents as a better metric to approximate perf impact in ANN tuning, as shown in Figure 3. We measured ANN accuracy using *1-recall@10*, which is averaged recall of getting top result from exact KNN search in top 10 results of ANN search.
- *Tune ANN parameters when there is non-trivial model change.* We observed that ANN performance is related with model characteristics. For example, when we employed ensemble techniques with model trained with non-click impressions, we found that while the model showed a better recall than baseline, the recall was worse than baseline after applying quantization to both. It should always be considered to tune the ANN parameters when there is a non-trivial change of the model training task, e.g., add more hard negatives.
- *Always try OPQ.* It is often useful to transform data prior to applying the quantization. We experimented with both PCA and OPQ [5] to transform the data, and observed that OPQ is generally more effective, as shown in Table 3 and Figure 3. One caveat of OPQ is: as it applied rotation of the embeddings, we might also need to retune *num_cluster* and *nprobe* to have similar documents scanned.

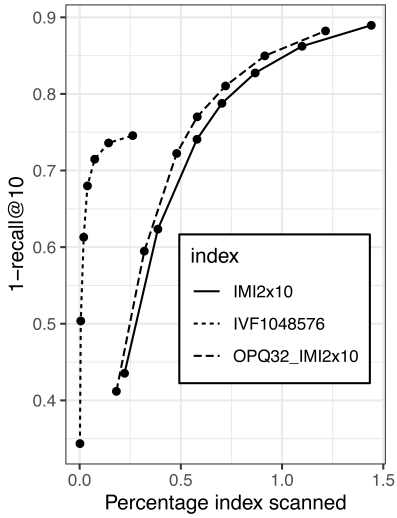

Figure 3: 1-Recall@10 of IVF and IMI Index

Table 3: Impact of Product Quantization on 1-Recall@10 for 128-Dimension Embedding.

Quantization Scheme	1-Recall@10	Index Scanned
PCA,PQ16	51.62%	0.48%
PCA,PQ16	54.11%	0.60%
PQ16	67.54%	0.58%
OPQ16,PQ16	70.51%	0.48%
OPQ16,PQ16	74.29%	0.60%
PQ32	74.27%	0.58%
PQ64	74.81%	0.58%
Flat (No Quantization)	74.81%	0.58%

- *Choose* pq_bytes *to be d/4.* Product quantizer compresses the vectors into x byte codes. For the choice of x, it is related to the dimension d of the embedding vector. Bigger x results in higher search accuracy but at cost of increased memory and latency. From empirical results, we found that the accuracy improvement is limited after $x > d/4$.
- *Tune nprobe, num_clusters, and pq_bytes online to understand the real perf impact.* While it is important to tune ANN algorithms and parameters offline to get a reasonable understanding of perf vs. recall trade-off, we found it is important to deploy several configs of the ANN algorithms and parameters online to get a better understanding of the perf impact from embedding-based retrieval to the real system. That is important to decide the capacity budget and reduce the scope of parameter search in offline tuning.

4.2 System Implementation

In order to integrate embedding-based retrieval into our serving stack, we implemented first-class support for NN search in Unicorn [3], a retrieval engine powering most search products at Facebook. Unicorn represents each document as a bag of terms, which are arbitrary strings that express binary properties about the document, conventionally namespaced with their semantics. For example, a user *John* living in *Seattle* would have the terms `text:john` and `location:seattle`. Terms can have payloads attached to them.

A query can be any Boolean expression on the terms. For example, the following query would return all the people that have *john* and *smithe* in the name, and live in *Seattle* or *Menlo Park*:

```
(and (or (term location:seattle)
         (term location:menlo_park))
     (and (term text:john)
          (term text:smithe)))
```

To support NN, we extended the document representation to include embeddings, each with a given string key, and added a `(nn <key> :radius <radius>)` query operator which matches all documents whose `<key>` embedding is within the specified radius of the query embedding.

At indexing time, each document embedding is quantized and turned into a term (for its coarse cluster) and a payload (for the quantized residual). At query time, the `(nn)` is internally rewritten into an `(or)` of the terms associated to the coarse clusters closest to the query embedding (*probes*), and for matching documents the term payload is retrieved to verify the radius constraint. The number of probes can be specified with an additional attribute `:nprobe`. By implementing NN support in terms of pre-existing primitives, instead of writing a separate system, we inherited all the features of the existing system, such as realtime updates, efficient query planning and execution, and support for multi-hop queries (see [3]).

The latter allows us to support top-K NN queries, where instead of matching by radius we select only the K documents closest to the query, and then evaluate the rest of the query. However, from our experimental study, we found that radius mode can give better trade-off of system performance and result quality. One possible reason is that radius mode enables a constrained NN search (constrained by other parts of the matching expression) but top K mode provides a more relaxed operation which needs to scan the whole index to get top K results. Hence, we use radius based matching in our current production.

4.2.1 Hybrid Retrieval. By having the `(nn)` operator as part of our Boolean query language we can now support *hybrid* retrieval expressions, with arbitrary combinations of embeddings and terms. This can be used for model based fuzzy matching which can improve on cases like spelling variations, optionalization etc. while reusing and benefiting from other parts of retrieval expression. For example, say a mis-spelled query *john smithe* is looking for a person named *john smith* in *Seattle* or *Menlo Park*; the retrieval expression would look like the one above.

That expression will fail to retrieve the user in question since the term `text:smithe` will fail to match that document. We can add fuzzy matching to this expression through `(nn)` operator:

```
(and (or (term location:seattle)
         (term location:menlo_park))
     (or (and (term text:john)
              (term text:smithe))
         (nn model-141795009 :radius 0.24 :nprobe 16)))
```

where `model-141795009` is the key for the embedding. In this case the target user will be retrieved if the cosine distance between the query (*john smithe*) embedding and document (*john smith*) embedding is less than 0.24.

4.2.2 Model Serving. We served the embedding model in the following way. After the two-sided embedding model was trained, we decomposed the model into a query embedding model and a document embedding model and then served the two models separately. For query embedding, we deployed the model in an online embedding inference service for real-time inference. For documents, we used Spark to do model inference in batch offline, and then published the generated embeddings together with other metadata into forward index. We did additional embedding quantization including coarse quantization and PQ to publish it into inverted index.

4.3 Query and Index Selection

To improve efficiency and quality of EBR, we performed query and index selection. We applied the query selection technique to overcome problems like over-triggering, huge capacity cost and junkiness increase. We did not trigger EBR for certain queries as EBR would be poor at and provide no extra value for them, such as easy queries with which searchers are looking for a specific target searched and clicked before, or queries with clearly different query intents from what the embedding model was trained for. On index side, we did index selection to make searching faster. For example, we only chose monthly active users, recent events, popular pages and groups.

5 LATER-STAGE OPTIMIZATION

Facebook search ranking is a complex multi-stage ranking system where each stage progressively refines the results from the preceding stage. At the very bottom of this stack is the retrieval layer, where embedding based retrieval is applied. Results from the retrieval layer are then sorted and filtered by a stack of ranking layers. The model at each stage should be optimized for the distribution of

results returned by the preceding layer. However, since the current ranking stages are designed for existing retrieval scenarios, this could result in new results returned from embedding based retrieval to be ranked sub-optimally by the existing rankers. To solve this problem, we proposed two approaches:

- *Embedding* as ranking feature. Propagating embedding similarities further down the funnel not only helps the ranker recognize new results from embedding-based retrieval, but also provides a generic semantic similarity measure for all results. We explored several options to extract features based on embeddings, including cosine similarity between the query and result embeddings, Hadamard product, and raw embeddings. From our experimental studies, cosine similarity feature consistently showed better performance than other options.
- *Training data feedback loop.* While embedding-based retrieval can improve retrieval recall, it might have a lower precision in comparison with term matching. To address the precision issue, we built a closed feedback loop based on human rating pipeline. In particular, we logged the results after enabling embedding-based retrieval, and then sent these results to human raters to label whether they are relevant or not. We used these human rated data to re-train the relevance model so that it can be used to filter out the irrelevant results from embedding-based retrieval while keeping the relevant ones. This proved to be a useful technique to achieve high precision for the recall improvement in embedding-based retrieval.

6 ADVANCED TOPICS

Embedding-based retrieval requires extensive research to continue improve the performance. We investigated two important areas for embedding modeling: hard mining and embedding ensemble, to continue advance the state of the art of embedding-based retrieval solutions.

6.1 Hard Mining

The data space for a retrieval task has diverse data distribution in degrees of text/semantic/social matches, and it is important to design a training data set for an embedding model to learn efficiently and effectively on such space. To tackle this, hard mining is one major direction as well as an active research area for embedding learning. However, most of the research are from computer vision field and for the classification task [6, 14, 16, 17], while search retrieval does not have concept of "classes" and therefore is a unique problem for which existing techniques do not necessarily work. In this direction, we divided our solutions into two parts: hard negative mining and hard positive mining.

6.1.1 Hard negative mining (HNM). When analyzing our embedding model for people search, we found that the top K results from embeddings given a query were usually with the same name, and the model did not always rank the target results higher than others even though the social features are present. This motivated us believe that the model was not able to utilize social features properly yet, and it's very likely because the negative training data were too easy as they were random samples which are usually with different names. To make the model better at differentiating between similar results, we can use samples that are closer to the positive examples in the embedding space as hard negatives in training.

Online hard negative mining. As model training is based on mini-batch updates, hard negatives can be selected in every batch in a dynamic but efficient way. Each batch comprises n positive pairs $\{(q^{(i)}, d_+^{(i)})\}_{i=1}^{n}$. Then for each query $q^{(i)}$, we formed a small document pool using all other positive documents $\{d_+^{(1)}, ..., d_+^{(j)}, ..., d_+^{(n)} | j \neq i\}$ and select the documents which received the highest similarity scores as the hardest negatives to create the training triplets. Enabling online hard negative mining was one major contributor to our modeling improvement. It consistently improved embedding model quality significantly across all verticals: +8.38% recall for people search; +7% recall for groups search, and +5.33% recall for events search. We also had observations that the optimal setting is at most two hard negatives per positive. Using more than two hard negatives will start to regress model quality.

One limitation of online HNM is that the probability of having any hard negative from random samples could be low and therefore cannot produce hard enough negatives. Next, we look at how to generate harder negatives based on the entire result pool, also known as offline Hard Negative Mining.

Offline hard negative mining. Offline hard negative mining has the following procedure:

(1) generate top K results for each query.
(2) select hard negatives based on *hard selection strategy*.
(3) retrain embedding model using the newly generated triplets.
(4) the procedure can be iterative.

We performed extensive experiments to compare offline hard negative mining and online hard negative mining. One finding that may first seem counterintuitive is that models trained simply using hard negatives cannot outperform models trained with random negatives. A further analysis suggested that the "hard" model put more weights on non-text features but did worse in text match than the "easy" model. Therefore, we worked to adjust the sampling strategy and finally produce a model that can outperform online HNM model.

The first insight is about *hard selection strategy*. We found that using the hardest examples is not the best strategy. We compared sampling from different rank positions and found sampling between rank 101-500 achieved the best model recall. The second insight is about *retrieval task optimization*. Our hypothesis is that presence of easy negatives in training data is still necessary, as a retrieval model is to operate on an input space which comprises data with mixed levels of hardness, and the majority of which are very easy negatives. Therefore, we explored several ways of integrating random negatives together with hard negatives, including transfer learning from an easy model. From our empirical study the following two techniques showed highest effectiveness:

- Mixed easy/hard training: blending random and hard negatives in training is advantageous. Increasing the ratio of easy to hard negatives continues to improve the model recall and saturated at easy:hard=100:1.
- Transfer learning from "hard" model to "easy" model: while transfer learning from easy to hard model does not produce a better model, transfer learning from hard to easy achieved further model recall improvement.

Last but not least, computing exhaustively KNN for each data point in the training data is very time-consuming, and the total model training time will become unrealistic due to the limited computing resources. It is important to have an efficient top K generation for the offline hard negative mining algorithm. Approximate nearest neighbor search is a practical solution here to reduce the total computational time significantly. Furthermore, running ANN search on one random shard is sufficient to generate effective hard negatives, as we only rely on *semi-hard* negatives during training.

6.1.2 Hard positive mining. Our baseline embedding model used clicks or impressions as positives, which the existing production can already return. To maximize the complementary gain by embedding based retrieval, one direction is to identify new results that have not been retrieved successfully by the production yet but positive. To this aim, we mined potential target results for failed search sessions from searchers' activity log. We found that the positive samples mined in this way is effective to help model training. Model trained using hard positives alone can achieve similar level of model recall as click training data while the data volume is only 4% if it. It can further improve model recall by combining both hard positives and impressions as training data.

6.2 Embedding Ensemble

We learned from HNM experiments that both easy and hard examples are important for EBR model training – we need hard examples to improve model precision, but easy example are also important to represent the retrieval space. The model trained using random negatives simulates the retrieval data distribution and is optimized for recall at a very large K, but it has poor precision at top K when K is small. On the other hand, the model trained to optimize precision, e.g. models trained using non-click impressions as negatives or offline hard negatives, is good at ranking for smaller set of candidates but failed for retrieval tasks. Thereafter we propose to combine models trained with different levels of hardness by a *multi-stage approach*, in which the first stage model focuses on recall and the second stage model specializes at differentiating more similar results returned by the first stage model. We shared the same spirit as the cascaded embedding training in [18], which ensembled a set of models trained with different level of hardness in a cascaded manner. We explored different forms of ensemble embeddings, including weighted concatenation and cascade model and found both effective.

Weighted Concatenation. As different models provided different cosine similarity scores for (query, document) pair, we used weighted sum of cosine similarity as the metric to define how close this pair is. To be more specific, given a set of models $\{M_1, \cdots, M_n\}$ and their corresponding weights, $\alpha_1, \cdots, \alpha_n > 0$, for any query Q and document D, we define the weighted ensemble similarity score $S_w(Q, D)$ between Q and D as

$$S_w(Q, D) = \sum_{i=1}^{n} \alpha_i \cos(V_{Q,i},\ U_{D,i}),$$

where $V_{Q,i}, 1 \le i \le n$, represent query vector of Q by model M_i, and $U_{D,i}, 1 \le i \le n$, represent document vector of D by model M_i.

For the serving purpose, we needed to ensemble multiple embedding vectors into a single representation, for query and document sides, respectively, that can satisfy the above metric property. We can prove that applying weighting multiplication to one side of normalized vector can satisfy the need. Specifically, we constructed the query vector and document vector in the following way:

$$E_Q = \big(\alpha_1 \frac{V_{Q,1}}{\|V_{Q,1}\|}, \cdots, \alpha_n \frac{V_{Q,n}}{\|V_{Q,n}\|}\big), \tag{4}$$

and

$$E_D = \big(\frac{U_{D,1}}{\|U_{D,1}\|}, \cdots, \frac{U_{Q,n}}{\|U_{Q,n}\|}\big). \tag{5}$$

It is easy to see that the cosine similarity between E_Q and E_D is proportional to $S_w(Q, D)$:

$$\cos(E_Q,\ E_D) = \frac{S_w(Q, D)}{\sqrt{\sum_{i=1}^{n} \alpha_i^2} \cdot \sqrt{n}}. \tag{6}$$

With above, we served the ensemble embedding in the same way as described in Section 4. The choices of weights were empirical which could be based on performance on the evaluation data set.

We explored several model options for the second stage embedding models. The experiments showed that models trained using non-click impressions achieved the best kNN recall (4.39% recall improvement over the baseline model). However, it suffered from much more accuracy loss compared to single model when applying embedding quantization, and therefore the actual benefit diminished when serving it online. We found that to have a best recall after embedding quantization, the best strategy was to ensemble a relatively easier model with offline hard negative mining model, for which the hardness level of training negatives had been modified and tuned. This ensemble candidate had slightly lower offline model improvement but was able to achieve significant recall improvement online.

Cascade Model. Unlike the parallel combination as in weighted ensemble, cascade model runs the second stage model on the output of the first stage model, in a serial fashion. We compared different second-stage model options. We found that models trained using non-click impressions was not a good candidate; the overall improvement was much smaller than the weighted ensemble approach. Moreover, the gain diminished as number of results to be reranked by the second-stage model increases. However, using offline hard negative model for the second stage achieved 3.4% recall improvement over the baseline. It was a more suitable model candidate for cascade because the constructed training data for offline HNM were exactly based on the output of the first-stage model.

Additionally, we explored another cascade model composition. We observed that while unified embedding overall had greater model recall compared to text embedding, it generated new text match failures because of its shifted focus to social and location matches. To enhance the text matching capability of the model hence improve the model precision we adopted the cascade strategy: using text embedding to pre-select text-matching candidates and then using unified embedding model to re-rank result to return the final top candidates. It achieved significant online improvement compared to using unified embedding model alone.

7 CONCLUSIONS

It has long term benefits to introduce semantic embeddings into search retrieval to address the semantic matching issues by leveraging the advancement on deep learning research. However, it is also a highly challenging problem due to the modeling difficulty, system implementation and cross-stack optimization complexity, especially for a large-scale personalized social search engine. In this paper, we presented our approach of unified embedding to model semantics for social search, and the implementation of embedding-based retrieval in a classical inverted index based search system.

It is only the first step to implement the unified embedding model and embedding-based retrieval system. There is still a long way to go to optimize the system end to end to make it perform well in terms of result quality and system performance. We introduced our experience in model improvement, serving algorithm tuning, and later-stage optimization. We believe this will be valuable experience to help people onboard embedding-based retrieval faster in real search engines. The successful deployment of embedding-based retrieval in production opens a door for sustainable improvement of retrieval quality by leveraging the latest semantic embedding learning techniques. We introduced our progress and learnings from the first step along this direction, especially on hard mining and embedding ensemble.

There are tremendous opportunities ahead to continuously improve the system. In the future, there are two main directions to pursue. One is to go *deep*. In terms of modeling we could apply the latest advanced models such as BERT [4] or build task-specific models to address particular segments of problems. We could investigate deeper in different stages including serving algorithm tuning and ranking model improvement, guided by full-stack failure analysis to identify the opportunities of improvements in different stacks. The other is to go *universal*. We could leverage the pre-trained text embedding models to develop a universal text embedding sub-model to be applied in different tasks. Furthermore, we could develop a universal query embedding model across all use cases.

REFERENCES

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. 2011. *Modern Information Retrieval: The Concepts and Technology behind Search* (2nd ed.). Addison-Wesley Publishing Company, USA.

[2] Y. Bengio, A. Courville, and P. Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 8 (Aug 2013), 1798–1828.

[3] Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko, Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin, Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, and Ning Zhang. 2013. Unicorn: a system for searching the social graph. *Proceedings of the VLDB Endowment* 6, 11, 1150–1161.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[5] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization for Approximate Nearest Neighbor Search. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[6] Alexander Hermans, Lucas Beyer, and Bastian Leibe. 2017. In Defense of the Triplet Loss for Person Re-Identification. *CoRR* abs/1703.07737 (2017). arXiv:1703.07737 http://arxiv.org/abs/1703.07737

[7] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM '13)*. Association for Computing Machinery, New York, NY, USA, 2333–2338.

[8] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (Jan. 2011), 117–128.

[9] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[10] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[11] Victor Lempitsky. 2012. The Inverted Multi-Index. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (CVPR '12)*. IEEE Computer Society, USA, 3069–3076.

[12] Hang Li and Jun Xu. 2014. *Semantic Matching in Search*. Now Publishers Inc., Hanover, MA, USA.

[13] Bhaskar Mitra and Nick Craswell. 2018. An Introduction to Neural Information Retrieval. *Foundations and Trends® in Information Retrieval* 13, 1 (December 2018), 1–126.

[14] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering.. In *CVPR*. IEEE Computer Society, 815–823. http://dblp.uni-trier.de/db/conf/cvpr/cvpr2015.html#SchroffKP15

[15] Josef Sivic and Andrew Zisserman. 2003. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2 (ICCV '03)*. IEEE Computer Society, USA, 1470.

[16] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. 2015. Deep Metric Learning via Lifted Structured Feature Embedding. *CoRR* abs/1511.06452 (2015). arXiv:1511.06452 http://arxiv.org/abs/1511.06452

[17] Chao-Yuan Wu, R. Manmatha, Alexander J. Smola, and Philipp Krähenbühl. 2017. Sampling Matters in Deep Embedding Learning. *CoRR* abs/1706.07567 (2017). arXiv:1706.07567 http://arxiv.org/abs/1706.07567

[18] Yuhui Yuan, Kuiyuan Yang, and Chao Zhang. 2017. Hard-Aware Deeply Cascaded Embedding. In *The IEEE International Conference on Computer Vision (ICCV)*.