Towards Personalized and Semantic Retrieval: An End-to-End Solution for E-commerce Search via Embedding Learning

Han Zhang [1†], Songlin Wang [1†], Kang Zhang [1], Zhiling Tang [1], Yunjiang Jiang [2], Yun Xiao [2], Weipeng Yan [1,2], Wen-Yun Yang [2*]
[1] JD.com, Beijing, China
[2] JD.com Silicon Valley Research Center, Mountain View, CA, United States
{zhanghan33, wangsonglin3, zhangkang1, tangzhiling, yunjiang.jiang, xiaoyun1, paul.yan, wenyun.yang}@jd.com

ABSTRACT

Nowadays e-commerce search has become an integral part of many people's shopping routines. Two critical challenges stay in today's e-commerce search: how to retrieve items that are semantically relevant but not exact matching to query terms, and how to retrieve items that are more personalized to different users for the same search query. In this paper, we present a novel approach called DPSR, which stands for Deep Personalized and Semantic Retrieval, to tackle this problem. Explicitly, we share our design decisions on how to architect a retrieval system so as to serve industry-scale traffic efficiently and how to train a model so as to learn query and item semantics accurately. Based on offline evaluations and online A/B test with live traffics, we show that DPSR model outperforms existing models, and DPSR system can retrieve more personalized and semantically relevant items to significantly improve users' search experience by +1.29% conversion rate, especially for long tail queries by +10.03%. As a result, our DPSR system has been successfully deployed into JD.com's search production since 2019.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Information systems** → **Information retrieval**.

KEYWORDS

Search; Semantic matching; Neural networks

ACM Reference Format:
Han Zhang [1†], Songlin Wang [1†], Kang Zhang [1], Zhiling Tang [1], Yunjiang Jiang [2], Yun Xiao [2], Weipeng Yan [1,2], Wen-Yun Yang [2*]. 2020. Towards Personalized and Semantic Retrieval: An End-to-End Solution for E-commerce Search via Embedding Learning. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401446

† Both authors contributed equally
* Corresponding author

SIGIR '20, July 25–30, 2020, Virtual Event, China

ACM ISBN 978-1-4503-8016-4/20/07…$15.00
https://doi.org/10.1145/3397271.3401446

Figure 1: Search interface on JD's e-commerce mobile app.

1 INTRODUCTION

Over the recent decades, online shopping platforms (e.g., Ebay, Walmart, Amazon, Tmall, Taobao and JD) have become increasingly popular in people's daily life. E-commerce search, which helps users to find what they need from billions of products, is an essential part of those platforms, contributing to the largest percentage of transactions among all channels [18, 27, 28]. For instance, the top e-commerce platforms in China, *e.g.*, Tmall, Taobao and JD, serve hundreds of million active users with gross merchandise volume of hundreds of billion US dollar. In this paper, we will focus on the immense impact that deep learning has recently had on the e-commerce search system. At a glance, Figure 1 illustrates the user interface for searching on JD's mobile app.

1.1 Three Components of Search System

Figure 2 illustrates a typical e-commerce search system with three components, query processing, candidate retrieval, and ranking.

Query Processing rewrites a query (*e.g.*, "cellphone for grandpa") into a term based presentation (*e.g.*, [TERM cellphone] AND [TERM grandpa]) that can be processed by downstream components. This stage typically includes tokenization, spelling correction, query expansion and rewriting.

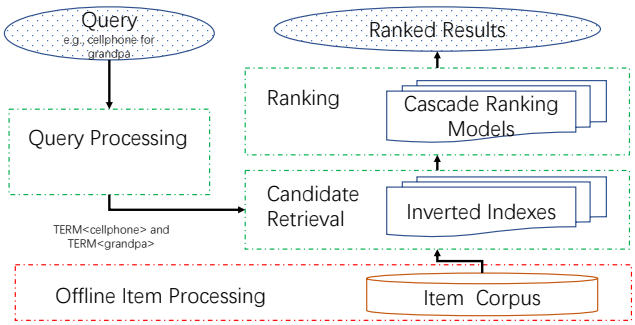


Figure 2: Major stages of an e-commerce search system.

Candidate Retrieval uses offline built inverted indexes, to efficiently retrieve candidate items based on term matching. This step greatly reduces the number of items from billions to hundreds of thousands, in order to make the fine ranking feasible.

Ranking orders the retrieved candidates based on factors, such as relevance, predicted conversion ratio, etc. A production system may have cascading ranking steps, which sequentially apply simpler to more complex ranking functions from upstream to downstream.

In this paper, we focus solely on the *candidate retrieval* stage to achieve more personalized and semantic search results, since this stage contributes the most bad cases in our search production. Based on our analysis, around 20% dissatisfaction cases of search traffic of JD.com, one of the largest e-commerce search engine in the world, can be attributed to the failure of this stage. How to deal with that in the ranking stage is out of scope for this paper, but will be our future work.

1.2 Two Challenges in Candidate Retrieval

How to efficiently retrieve more personalized and semantically relevant items remains two major challenges in modern e-commerce search engines.

Semantic Retrieval Problem refers to that, items that are semantically relevant but do not contain the exact terms of a query cannot be retrieved by traditional inverted indexes. As reported in [17], the most critical challenge for search systems is term mismatch between queries and items, especially for e-commerce search, where item titles are often short. Traditional web search often uses query rewriting to tackle this problem, which transforms the original query to another similar query that might better represent the search need. However, it is hard to ensure that the same search intention can be kept through a "middle man", *i.e.*, rewritten queries, and there is also no guarantee that relevant items containing different terms can be retrieved via a limited set of rewritten queries.

Personalized Retrieval Problem refers to that, traditional inverted indexes cannot retrieve different items according to the current user's characteristics, *e.g.*, gender, purchase power and so on. For example, we would like to retrieve more women's T-shirt if the user is female, and vice versa. Some rule-based solutions have been used in our system for years include that, 1) indexing tags for items, *e.g.*, purchase power, gender and so on, the same way as tokens into the inverted index, 2) building separate indexes for different group of users. However, these previous approaches are too hand-crafted. Thus, they are hard to meet more subtle personalization needs.

1.3 Our Contributions

In this paper, we propose DPSR: Deep Personalized and Semantic Retrieval, to tackle the above two challenges in a leading industrial-scale e-commerce search engine. The contributions of our work can be summarized as follows.

In Section 3, we present an overview of our full DPSR embedding retrieval system composed of offline model training, offline indexing and online serving. We share our critical design decisions for productionizing this neural network based candidate retrieval into an industry-level e-commerce search engine.

In Section 4, we develop a novel neural network model with a two tower architecture, a multi-head design of query tower, an attention based loss function, a negative sampling approach, an efficient training algorithm, and human supervision data, all of which are indispensable to train our best performing models.

In Section 5, we present our efforts on building a large-scale deep retrieval training system where we significantly customize the off-the-shelf TensorFlow API for online/offline consistency, input data storage and scalable distributed training, and on building an industrial-scale online serving system for embedding retrieval.

In Section 6, we conduct extensive embedding visualization, offline evaluation and online AB test to show that our retrieval system can help to find semantically related items and significantly improve users' online search experience, especially for the long tail queries, which are difficult to handle in traditional search systems (*i.e.*, improving conversion rate by around 10%).

2 RELATED WORK

2.1 Traditional Candidate Retrieval

For candidate retrieval, most research focuses on learning query rewrites [2, 10] as an indirect approach to bridge vocabulary gap between queries and documents. Only a few new approaches, including latent semantic indexing (LSI) [6] with matrix factorization, probabilistic latent semantic indexing (PLSI) [12] with probabilistic models, and semantic hashing [25] with an auto-encoding model, have been proposed. All of these models are unsupervised learned from word co-occurrence in documents, without any supervised labels. Our approach differs from the previous methods in that we train a supervised model to directly optimize relevance metrics based on a large-scale data set with relevant signals, *i.e.*, clicks.

2.2 Deep Learning Based Relevance Model

With the success of deep learning, a large number of neural network based models have been proposed to advance traditional information retrieval (IR) methods (*e.g.*, BM2.5) and learning to rank methods [19] in the manner of learning semantic relevance between queries and documents. See [17] and [20] for a comprehensive survey in semantic match and deep neutral network based IR. Particularly, DSSM [13] and its following work CDSSM [26] have pioneered the work of using deep neural networks for relevance scoring. Recently, new models including DRMM [9], Duet [21] have been further developed to include traditional IR lexical matching

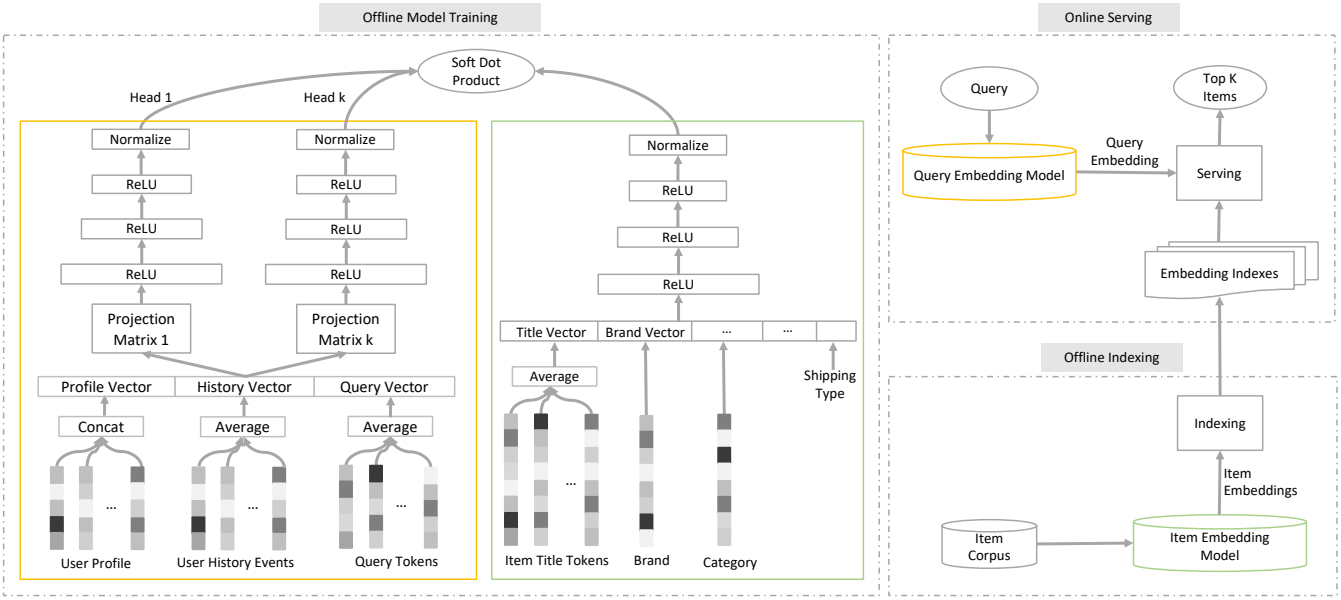


Figure 3: Overview of our DPSR retrieval system.

signals (*e.g.*, query terms importance, exact matching) in neural networks. However, as reported by [20], most of the proposed works in this direction focus on ranking stage, where the optimization objectives and requirements are very different from candidate retrieval that our work in this paper focuses on.

Two tower architecture for deep neural network has been widely adopted in existing recommendation works [33, 34] to further incorporate item features. This model architecture is also known as dual encoder in natural language processing [4, 11]. Here we propose a more advanced two tower model which is composed of a multi-head tower for query and an attention loss based on soft dot product instead of simple inner product.

2.3 Embedding Retrieval in Search Engine

Recently, embedding retrieval technologies have been widely adopted in modern recommendation and advertising systems [5, 16, 36], while have not been widely used in search engine yet. We find a few works about retrieval problems in search engine [23, 30], while they have not been applied to industrial production system. To the best of our knowledge, we are one of the first practical explorations in this direction of applying embedding retrieval in industrial search engine system.

3 OVERVIEW OF EMBEDDING RETRIEVAL SYSTEM

Before we present the details, let us first show a full picture of our embedding retrieval system. Figure 3 illustrates our production system with three major modules as follows.

Offline Model Training module trains a two tower model consisting of a query embedding model (*i.e.*, query tower) and an item embedding model (*i.e.*, item tower) for the uses in online serving and offline indexing respectively. This two tower model structure is a careful and essential design to enable fast online embedding retrieval, which we will discuss more in Section 4. Moreover, We will also talk about our effort of optimizing offline training system in Section 5.1.

Offline Indexing module loads the item embedding model (*i.e.*, the item tower) to compute all the item embeddings from the item corpus, and then builds an embedding index offline to support efficient online embedding retrieval. As it is infeasible to exhaustively search over the item corpus of billions of items, to find similar item embeddings for a query embedding, we employ one of state-of-the-art algorithms [15] for efficient nearest neighbor search of dense vectors.

Online Serving module loads the query embedding model (*i.e.*, the query tower) to transform any user input query text to query embedding, which is then fed to the item embedding index to retrieve K similar items. Note that this online serving system has to be built with low latency of tens of milliseconds. Also, it must be scalable to hundreds of thousands queries per second (QPS), and flexible for agile iterations of experiments. We will talk about our efforts of building such an online serving system in Section 5.2.

4 EMBEDDING LEARNING MODEL

In this section, we introduce the embedding learning model in a stepwise fashion, in the order of two tower architecture, multi-head design of query tower, attentive loss function, hybrid negative sampling, and human supervision data, all of which are indispensable to train our best performing model.

4.1 Two Tower Model Architecture

As shown in offline model training module in Figure 3, the model is composed of a query tower Q and an item tower S. For a given

query q and an item s, the scoring output of the model is

$$f(q, s) = G(Q(q), S(s))$$

where $Q(q) \in \mathbb{R}^{d \times m}$ denotes query tower outputs of m query embeddings in d-dimensional space. Similarly, $S(s) \in \mathbb{R}^{d \times n}$ denotes item tower outputs. The scoring function $G(.,.)$ computes the final score between the query and item. Researchers and practitioners often let query tower Q and item tower S both output one single embedding, *i.e.*, $m = 1$ and $n = 1$, and choose G as inner product, *i.e.*, $G(Q(q), S(s)) = Q(q)^\top S(s)$ where the superscript $^\top$ denotes matrix transpose. This simplest setup has been proved to be successful in many applications [5].

The key design principle for such two tower architecture is to make the query embedding and the item embedding independent on each other after the model is trained. So we can compute them separately. All item embeddings can be computed offline in order to build an item embedding index for fast nearest neighbor search online, and the query embedding can be computed online to handle all possible user queries. Even though the embeddings are computed separately, due to the simple dot product interaction between query and item towers, the query and item embeddings are still theoretically in the same geometric space. Thus, finding K nearest items for a given query embedding is equivalent to minimizing the loss for K query item pairs where the query is given.

In below sections, we will introduce a novel design of query tower Q and an interaction function G to achieve outperforming and explainable retrieval results. Since item representations are normally straightforward, we still keep the item tower S typically simple. It concatenates all item features as input layer, then goes through multi-layer perceptron (MLP) of fully connected Rectified Linear Units (ReLU) to output a single item embedding, which is finally normalized to the same length as query embedding, as shown in the right side of offline model training panel in Figure 3. Similar MLP structure can be found in previous work [5].

4.2 Query Tower with Multi-heads

As shown in the left side of offline model training panel in Figure 3, query tower differs from item tower in two places, 1) a projection layer that projects the one input dense representation to K dense representations. Another choice here is to use K independent embedding set, but it requires larger model size. In practice, we choose the projection layer to achieve similar results but with much smaller model size. 2) K separate encoding MLPs, each of which independently outputs one query embedding that potentially would capture different intention for the query. We refer to these K output embeddings as *multi-head representations*.

These multiple query embeddings provide rich representations for the query's intentions. Typically, we find in practice that it could capture different semantic meanings for a polysemous query (*e.g.*, "apple"), different popular brands for a product query (*e.g.*, "cellphone"), and different products for a brand query (*e.g.*, "Samsung").

It is worth mentioning that the encoding layer can use any other more powerful neural network, such as Recurrent Neural Network (RNN) and other state-of-the-art transformer based models [7, 24, 29]. In a separate offline study, we have achieved similar or slightly better results with these advanced models. However, we would like to emphasize that a simple MLP is more applicable to our industrial production modeling system, since it is much more efficient for both offline training and online serving, which means that we are able to feed more data to the model training, and deploy fewer machines to serve the model. These are strong deal breakers in industrial world.

4.3 Optimization with Attention Loss

Apart from the single embedding and inner product setup, here we develop a more general form for multiple query embeddings. As a shorthand, we denote each output of query tower $Q(q)$ as $\{e_1, e_2, \ldots, e_m\}$ where $e_i \in \mathbb{R}^d$, and the single output of item tower $S(s)$ as $g \in \mathbb{R}^d$. Then the soft dot product interaction between query and item can be defined as follows,

$$G(Q(q), S(s)) = \sum_{i=1}^{m} w_i e_i^\top g. \tag{1}$$

This scoring function is basically a weighted sum of all inner products between m query embeddings and one item embedding. The weights w_i are calculated from softmax of the same set of inner products,

$$w_i = \frac{\exp(e_i^\top g/\beta)}{\sum_{j=1}^{m} \exp(e_j^\top g/\beta)},$$

where β is the temperature parameter of softmax. Note that the higher the β is, the more uniform the attention weights appear. If $\beta \to 0$, then the soft dot product in Equation (1) would be equivalent to selecting the largest inner product, *i.e.*, $\max_i e_i \top g$.

A typical industrial click log data set usually contains only click pairs of query and item. The pairs are usually relevant, thus can be treated as positive training examples. Besides that, we also need to collect negative examples by various sampling techniques that we will talk about later in Section 4.4. Let us define the set $\mathcal{D}$ of all training examples as follows,

$$\mathcal{D} = \left\{ \left(q_i, s_i^+, \mathcal{N}_i\right) \;\middle|\; i,\ r(q_i, s_i^+) = 1,\ r(q_i, s_j^-) = 0 \ \forall\ s_j^- \in \mathcal{N}_i \right\}, \tag{2}$$

where each training example is a triplet composed of, a query q_i, a positive item s_i^+ that is relevant to the query denoted as $r(q_i, s_i^+) = 1$, and an negative item set $\mathcal{N}_i$ where every element s_j^- is irrelevant to the query, denoted as $r(q_i, s_j^-) = 0$. Then we can employ hinge loss with margin δ over the training data set $\mathcal{D}$ as follows,

$$\mathcal{L}(\mathcal{D}) = \sum_{(q_i, s_i^+, \mathcal{N}_i) \in \mathcal{D}} \sum_{s_j^- \in \mathcal{N}_i} \max\left(0, \delta - f(q_i, s_i^+) + f(q_i, s_j^-)\right).$$

The above attention loss is only applied in the offline training. During the online retrieval, each query head retrieves the same number of items. Then all the items will be sorted and cut off based on their inner products with one of the heads.

4.4 Click Logs with Negative Sampling

Training a deep model requires a huge amount of data. We explore click logs, which represents users' implicit relevance feedback and consists of a list of queries and their clicked items, to train our embedding retrieval model. Intuitively, we can assume that an item is relevant, at least partially, to the query if it is clicked for that query. Formally, we can consider click logs as a special case of data set with only positive examples. Then how to efficiently

collect negative examples is a crucial question here. In our practice, we employ a hybrid approach that mixes two sources of negative samples, including random negatives and batch negatives.

4.4.1 Random Negatives. Random negative set $\mathcal{N}_i^{rand}$ are uniformly sampled from all candidate items. Formally, given a set of all N available items, we draw a random integer variable from a uniform distribution $i \sim Uniform(1, N)$, and take the i-th element from the item set into random negative set $\mathcal{N}_i^{rand}$. However, if we apply this uniform sampling in a straightforward way, it would be very computational expensive, since each negative sample has to go through the item tower, not to mention the cost for sampling those negative examples and fetching their features. To minimize the computational cost while retaining its effect, we use the same random negative set for all training examples in a batch. In practice, we found the results are similar to that using pure random negatives but the training speed is much faster.

4.4.2 Batch Negatives. Batch negative set $\mathcal{N}_i^{batch}$ are collected by permuting the positive query item pairs in a training batch. In detail, for a training batch

$$\mathcal{B} = \{(q_i, s_i^+, \mathcal{N}_i) \mid i \},$$

we can collect more negative examples for the i-th example as

$$\mathcal{N}_i^{batch} = \left\{ s_k^+ \mid k \neq i, 1 \leq k \leq |\mathcal{B}| \right\}.$$

We can see that batch negatives are basically sampled according to item frequency in the dataset. These randomly generated query and item pairs are very unlikely to be relevant by chance. Specifically, the chance is equal to that two randomly drawn click logs having relevant items for each other. Given a dataset of hundreds of millions of click logs, this chance is basically ignorable in terms of training accuracy. Also, the main advantage of the above batch negatives is the reuse of the item embedding computations. Each item embedding in the batch serves once as positive example, and $|\mathcal{B}| - 1$ times as negative examples for other queries in the batch, but with only one feature fetching and forward pass of the item tower.

4.4.3 Mixing Ratio. Eventually, the complete negative item set $\mathcal{N}_i$ in Equation (2) is a union set of above two sets,

$$\mathcal{N}_i = \mathcal{N}_i^{rand} \cup \mathcal{N}_i^{batch}.$$

In our practice of e-commerce search retrieval, we find it is typically useful to have a mixing ratio parameter $0 \leq \alpha \leq 1$ for the composition of negative sample set. Formally, we use proportion α of random negatives, and proportion $(1 - \alpha)$ of batch negatives. We find the value of α highly correlates with the popularity of items retrieved from the model (see Experiments), thus highly influential to online metrics. Intuitively, we can see that the mixing ratio α determines the item distribution in negative examples, from uniform distribution ($\alpha = 1$) to actual item frequency ($\alpha = 0$). In this manner, the model tends to retrieve more popular items for larger α, as popular items appear relatively less frequently in negative examples.

4.4.4 Summary. We summarize the full training algorithm with batch negatives and random negatives in Algorithm 1. The computational complexity for each training step is $O(b^2)$, *i.e.*, quadratic with the batch size b, since the batch negatives require an inner product between every query and item embedding pair in the batch. In practice, since the batch size is usually small, *e.g.*, 64 or 128, the quadratic effect is actually much smaller than other computational cost, *i.e.*, feature fetching, gradient computation, and so on. In fact, with batch negatives, the total convergence is actually faster, due to the efficient use of every item tower outputs.

Algorithm 1 DPSR training algorithm

1: **input**: Dataset $\mathcal{D}$, batch size b, max number of steps T, mixing ratio α.
2: **for** $t = 1 \ldots T$ **do**
3: Sample a batch of b examples $\mathcal{B} \subseteq \mathcal{D}^+$.
4: Sample a set of random negatives $\mathcal{N}^{rand}$ for this batch. Note that all examples in the batch shares this set.
5: Compute query head embeddings $Q(q)$ from query tower.
6: Compute item embeddings $S(s)$ for all item s_i in the batch, and that in the random negative set $\mathcal{N}^{rand}$.
7: Compute loss function value $\mathcal{L}(\mathcal{B})$ for this batch $\mathcal{B}$. The batch negatives $\mathcal{N}^{batch}$ are implicitly computed and included in the loss.
8: Update towers Q and S by back propagation.
9: **end for**
10: **return** query tower Q and item tower S.

4.5 Human Supervision

Beyond using click logs data, our model is also able to utilize additional human supervision to further correct corner cases, incorporate prior knowledge and improve its performance. The human supervision comes from three sources:

- *Most skipped items* can be automatically collected from online logs [14]. These items and the associated queries can be used as negative examples.
- *Human generated data* can be collected based on domain knowledge as artificial negative query item pairs (*e.g.*, cellphone cases are generated as negative items for query "cellphone", because they share similar product words literally but differ significantly in semantic meaning) and positive query item pairs (*e.g.*, iPhone 11 items are generated as positive items for query "newest large screen iphone").
- *Human labels and bad case reports* are normally used to train relevance models [35]. We also include them as both positive and negative examples in the training data set.

These human supervision data can be fed into the model as either positive query item pairs or an item in the random negative set.

5 EMBEDDING RETRIEVAL SYSTEM

We employ TensorFlow [1] as our training and online serving framework, since it has been widely used in both academia and industry. Particularly, it has the advantage of high-performance of training speed with static graph pre-built before training, and seamless integration between training and online serving. We built our system based on the high level TensorFlow API, called Estimator [32]. To ensure best performance and system consistency, we have also made

significant efforts to abridge an off-the-shelf TensorFlow package and an industry level deep learning system.

5.1 Training System Optimizations

5.1.1 Consistency Between Online and Offline. One of the common challenges for building a machine learning system is to guarantee the offline and online consistency. A typical inconsistency usually happens at the feature computation stage, especially if two separate programming scripts are used in offline data pre-processing and online serving system. In our system, the most vulnerable part is the text tokenization, carried on three times in data preprocessing, model training and online serving. In aware of this, we implement one unique tokenizer in C++, and wrap it with a very thin Python SWIG interface [3] for offline data vocabulary computation, and with TensorFlow C++ custom operator [1] for offline training as well as online serving. Consequentially, it is guaranteed that the same tokenizer code runs through raw data preprocessing, model training and online prediction.

5.1.2 Compressed Input Data Format. A typical data format for industrial search or recommendation training system is usually composed of three types of features, user features (*e.g.*, query, gender, locale), item features (*e.g.*, popularity), and user-item interaction features (*e.g.*, was it seen by the user). The plain input data will repeat user and item features many times since the training data store all user item interaction pairs, which results in hundreds of terabytes of disk space occupation, more data transferring time and slow training speed. To solve this problem, we customized TensorFlow Dataset [31] to assemble training examples from three separate files, a user feature file, an item feature file and an interaction file with query, user id and item id. The user and item feature files are first loaded into memory as feature lookup dictionaries, then the interaction file is iterated over the training steps with the user and item features appended. With this optimization, we successfully reduced the training data size to be 10% of the original size.

5.1.3 Scalable Distributed Training. In the scenario of distributed training with parameter servers, one of the common bottlenecks is network bandwidth. Most of mainframe network bandwidth in industry is 10G bits that are far from enough for large deep learning models. We observed that the off-the-shelf TensorFlow Estimator implementation is not optimized enough when handling embedding aggregation (*e.g.*, sum of embeddings), thus the network bandwidth becomes a bottleneck quickly while adding a handful of workers. To further scale up the training speed, we improved the embedding aggregation operator in TensorFlow official implementation by moving the embedding aggregation operation inside parameter server, instead of in the workers. Thus, only one embedding is transferred between parameter server and worker for each embedding aggregation, instead of tens of them. Therefore, network bandwidth is significantly reduced, and the distributed training can be scaled up to five times more machines.

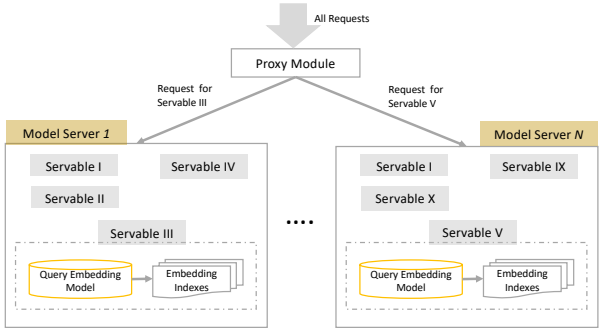


Figure 4: Online serving system for DPSR.

5.2 Online Serving System

The overview of DPSR online serving system is shown in Figure 4. The system consists of two novel parts that we would like to elaborate on, one TensorFlow Servable [22] model, and a proxy for model sharding.

5.2.1 One Servable Model. The straightforward implementation of DPSR can be composed of two separate parts, query embedding computation, and nearest neighbor lookup. Without careful design, one can simply build two separate online services for them. However, this is not the optimal system design in the sense of two points, a) it introduces complexity to manage the mapping between query embedding model and item embedding indexes, which could completely cause system failure if mapping mistake happens. b) it needs two network round trips to compute the nearest neighbors for a given query text. To overcome these issues, we take a more optimized approach by utilizing TensorFlow Servable [22] framework, where we can unify the two parts into one model. As shown in Figure 4, the two parts can be encapsulated into one Servable. The query embedding is sent directly from query embedding model to item embedding index, via computer memory, instead of via computer network.

5.2.2 Model Sharding. The further scale up of the system needs to support hundreds of DPSR models online at the same time, for different retrieval tasks, and for various model A/B experiments. However, one servable model consisting of one query embedding model and one item embedding index usually takes tens of Gigabytes of memory. Thus, It becomes infeasible to store all the models in one machine's memory, and we have to build a system to support serving hundreds of DPSR models. We solve this problem by a proxy module, which plays the role of directing model prediction requests to one of the model servers that hold the corresponding model, as shown in Figure 4. This infrastructure is not only designed for DPSR, but as a general system for supporting all deep learning models at our search production.

6 EXPERIMENTS

In this section, we first visualize the embedding results leveraging t-SNE in Section 6.1, so we can get the intuition of how the model works. Then we report offline evaluations by comparing with different methods in Section 6.2. Next, we report online A/B test results in our search production, one of largest e-commerce

search engines in the world, in Section 6.3. Furthermore, we also report the offline indexing and online serving time of our DPSR system in Section 6.4, to demonstrate its efficiency, which is crucial in the industrial world.

Our production DPSR model is trained on a data set of 60 days user click logs, which contains 5.6 billion sessions. We conducted distributed training in a cluster of five 48-cores machines, with a total of 40 workers and 5 parameters servers launched. We used margin parameter $\delta = 0.1$, AdaGrad [8] optimizer with learning rate 0.01, batch size $b = 64$, embedding dimension $d = 64$. The training converges in about 400 million steps for about 55 hours.

6.1 Embedding Visualization and Analysis

6.1.1 Embedding Topology. To have an intuition of how our embedding retrieval model works, we illustrate the 2-D t-SNE coordinates for frequent items chosen from the most popular 33 categories in our platform. As shown in Figure 5, we can see that the item embeddings are structured in a very explicit and intuitive way. Basically, we can see that the electronics related categories, *e.g.*, phones, laptops, tablets, earphones, monitors are well placed on the left side of the figure. The appliance related categories, *e.g.*, refrigerator, flat TV, air conditioner, washer and so on are placed on the lower left side. The food related categories, *e.g.*, snacks, instant food, cookies, milk powder, are placed on the lower right part. The cleaning and beauty related categories, *e.g.*, face cream and shampoo, are placed on the right part. The clothes related categories, *e.g.*, shoes, running shoes, sweaters and down jackets, are placed on the upper right part. Overall, this reasonable and intuitive embedding topology reflects that the proposed model well learns the item semantics, which in turn enables query embeddings to retrieve relevant items.

6.1.2 Multi-Head Disambiguation. In Figure 6b, we also compute the 2-D t-SNE coordinates for frequent items chosen from 10 commodity categories to illustrate the effect of having multi-heads in query tower. We use two polysemous queries as an example here, "apple" and "cellphone", which are also within the top-10 queries in our platform. We can see that the two heads for the query "apple" separately retrieve iPhone/Macbook and apple fruit. In Figure 6c, we can see that the two heads for the query "cellphone" retrieve the two most popular brands, Huawei and Xiaomi, separately. The illustration shows that different heads are able to focus on different possible user intentions. In contrast, the single head model in Figure 6a does not cluster well for cellphone category, where the iPhones are forming another cluster far away from other cellphones, potentially due to the ambiguity of the very top query "apple".

6.1.3 Semantic Matching. For better understanding of how our proposed model performs, we show a few good cases from our retrieval production in Table 1. We can observe that DPSR is surprisingly capable of bridging queries and relevant items by learning the semantic meaning of some words, such as big kid to 3-6 years old, free-style swimming equipment to hand paddle, and grandpa to senior. Also, DPSR is able to correct typos in the query, such as v bag to LV bag, and ovivo cellphone to vivo cellphone, partially because we leverage English letter trigrams in the token vocabulary. We also observed similar typo corrections for Chinese characters, which are mainly learned from user clicks and n-gram embeddings.

6.2 Offline Evaluations

6.2.1 Metrics. We use the following offline metrics to evaluate the retrieval methods.

Top-k is defined as the probability that a relevant item is ranked within the top k retrieved results among N (we used $1,024$) random items for a given query. This top-k value is empirically estimated by averaging $200,000$ random queries. A higher top-k indicates a better retrieval quality, *i.e.*, hit rate.

AUC is computed in a separate data set with human labeled relevance for query item pairs. The labels can be categorized into relevant and non-relevant ones, and then the embedding inner products or any relevancy scores (BM2.5) can be treated as prediction scores. A higher AUC here indicates a better retrieval relevancy.

Time is the total retrieval time on a 48-core CPU machine from a query text to $1,000$ most relevant items out of a set of 15 million items. This metric value decides whether a method is possible to apply to industry-level retrieval system or not. Typically, the cutoff is 50 milliseconds, but preferably 20 milliseconds.

6.2.2 Baseline Methods. We compared DPSR with BM2.5 and DSSM as baselines. BM2.5 is a classical information retrieval method based on keywords matching using inverted index, and it uses heuristics to score documents based on term frequency and inverted document frequency. We compare with two versions of BM2.5, with only unigrams, and with both unigrams and bigrams (denoted as BM2.5-u&b). DSSM is a classical deep learning model [13] designed for ranking but not retrieval. We still would like to include the comparison to clarify the difference.

6.2.3 Results. In Table 2, we show the comparison results with the above baseline methods. We can make the following observations from the results.

- BM2.5 as a classical method shows good retrieval quality, but it takes more than a minute to retrieve from 15 million items, which means that it is too unrealistic to use it in online retrieval.
- DSSM that samples unclicked items as negative examples performs worst in top-k, MRR and AUC. This is mainly due to that DSSM is optimized for ranking tasks, which is a highly different task from retrieval. Therefore, we can conclude that only using unclicked items as negative examples does not work to train a retrieval model.
- DPSR refers to a vanilla version of our model without any user features. It has the highest AUC score among the baseline methods and other personalized DPSR versions, which indicates that pure semantic DPSR could achieve the highest retrieval relevance.
- DPSR-p refers to a basic personalized version of our model, with additional user profile features, like purchase power, gender and so on. The result shows that those profile features help improve the retrieval quality metrics (Top-k) over the vanilla version, with a slight tradeoff of relevancy.
- DPSR-h refers to a full personalized version of our model, with both user profile and user history events. It has the best retrieval quality metrics (Top-k) over all models, which demonstrates that plenty of signals can be squeezed from the user history events. Note that the personalized model

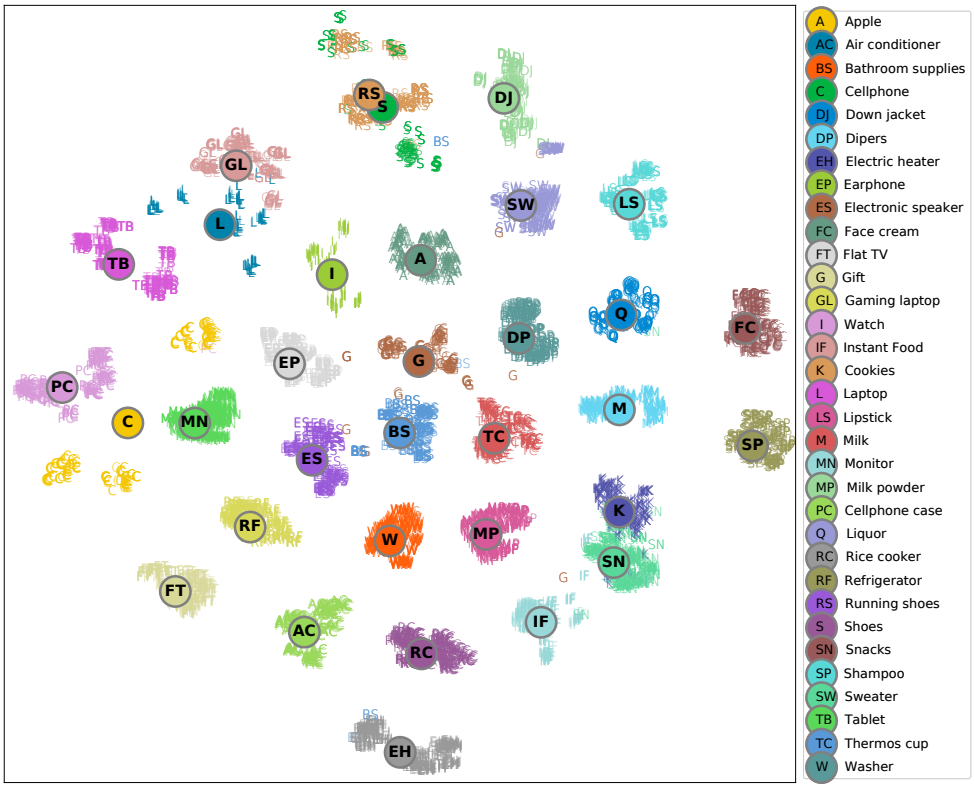
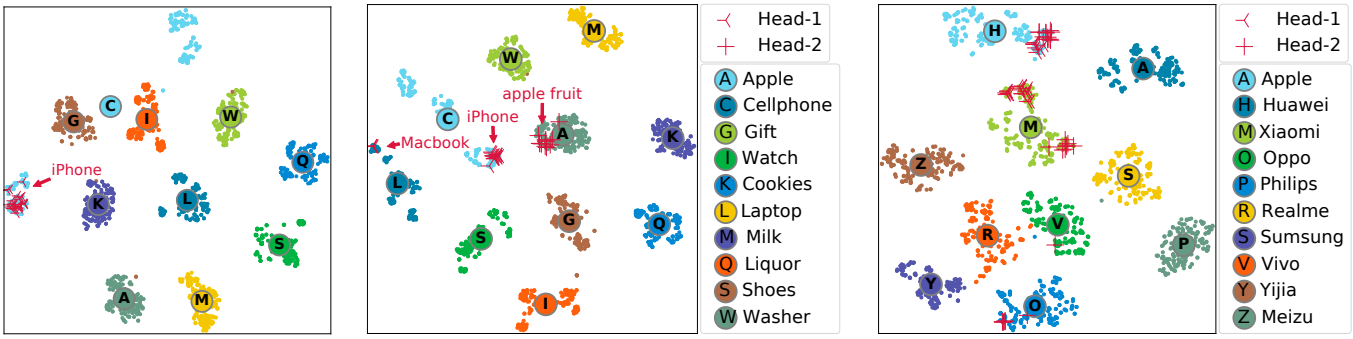


Figure 5: t-SNE visualization of item embeddings from 33 most popular categories.



(a) One head for query "apple". **(b) Two heads for query "apple".** **(c) Two heads for query "cellphone".**

Figure 6: t-SNE visualizations of retrieval results for polysemous queries.

Table 1: Good cases from DPSR system in production.

query	retrieved item
奶粉 大童 (milk powder big kid)	美赞臣 安儿健A+ 4段 (Enfamil A+ level-4 for 3 to 6 years old)
碧倩套装 ("Clinique typo" set)	倩碧(CLINIQUE)经典三部曲套装 (Clinique classic trilogy set)
官网v女包 (authentic v women bag)	路易威登LV女包 (Louis Vuitton LV women bag)
ovivo手机 (ovivo cellphone)	vivo Z1 (vivo Z1 phone)
学习自由泳器材 (learn free-style swimming equipment)	英发/yingfa 划臂 (yinfa hand paddle)

Table 2: Comparison between different methods.

	Top-1	Top-10	AUC	Time
BM2.5	0.718	0.947	0.661	61 s
BM2.5-u&b	0.721	0.948	0.661	157 s
DSSM	0.002	0.016	0.524	20 ms
DPSR	0.839	0.979	**0.696**	20 ms
DPSR-p	0.868	0.984	0.692	20 ms
DPSR-h	**0.889**	**0.998**	0.685	20 ms

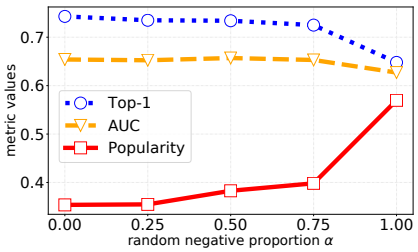


Figure 7: Effect with different mixing ratio of negatives.

improves the retrieval quality metrics with a tradeoff of relevance metrics (AUC), which is also reasonable, since the retrieval quality consisting of more factors besides relevancy, such as item popularity, personalization and so on.

Moreover, Figure 7 illustrates that the mixing ratio α of random negatives and batch negatives (see Section 4.4.3) affects the retrieved item popularity. Basically, we can observe that the more random negatives we have in the negative sampling, the more popular items are retrieved. But too many random negatives, *e.g.*, $\alpha = 1.0$, will hurt the retrieved item's relevancy. Thus, we can treat the parameter of α as a tradeoff between retrieval popularity and relevancy. In practice, we also found a proper choice of $\alpha = 0.5$ or $\alpha = 0.75$ would help online metrics significantly.

6.3 Online A/B Tests

DPSR is designed as a key component in our search system to improve the overall user experience. Thus, we would like to focus on the overall improvement of a search system using DPSR as an additional retrieval method.

In the control setup (baseline), it includes all the candidates available in our current production system, which are retrieved by inverted-index based methods with query rewritten enabled. In the variation experiment setup (DPSR), it retrieves at most $1,000$ candidates from our DPSR system in addition to those in the baseline. For both settings, all the candidates will go through the same ranking component and business logic. The ranking component applies a state-of-the-art learning-to-rank method similar to methods mentioned in [18]. Here, we emphasize that our production system is a strong baseline to be compared with, as it has been tuned by hundreds of engineers and scientists for years, and has applied state-of-the-art query rewriting and document processing methods to optimize candidate generation.

We first conducted human evaluation for the relevance of retrieved items. Specifically, we ask human evaluators to label the relevance of results from the baseline system and DPSR for the same set of 500 long tail queries. The labeled results are categorized into 3 buckets, bad, fair and perfect. Table 3 shows that the proposed method improve search relevancy by reducing around 6% bad cases. It proves that the deep retrieval system is especially effective in handling "difficult" or "unsatisfied" queries, which often require semantic matching.

Table 3: Relevancy metrics by human labeling of 500 long tail queries. DPSR reduces bad cases significantly.

	bad	fair	perfect
Baseline	17.86%	26.04%	56.10%
DPSR	13.70%	33.28%	53.01%

Table 4: DPSR Online A/B test improvements.

	UCVR	GMV	QRR
1-head	+1.13%	+1.78%	−4.44%
2-head	+1.34%	+2.13%	−4.13%
1-head-p13n	+1.29%	+2.19%	−4.29%
2-head on long tail query	+10.03%	+7.50%	−9.99%

We then conducted live experiments over 10% of the entire site traffic during a period of two weeks using a standard A/B testing configuration. To protect confidential business information, only relative improvements are reported. Table 4 shows that the proposed DPSR retrieval improves the production system for all core business metrics in e-commerce search, including user conversation rate (UCVR), and gross merchandise value (GMV), as well as query rewrite rate (QRR), which is believed to be a good indicator of search satisfaction. We can also observe that the 2-heads version of query tower, and personalized version (denoted as 1-head-p13n) both improve the vanilla version of 1-head query tower without any user features. Especially, we observe that the improvements mainly come from long tail queries, which are normally hard for traditional search engines.

6.4 Efficiency

In Table 5, we show the efficiency of our offline index building and online nearest neighbor search excluding the query embedding computation. We report the time consumed for indexing and searching 15 million items with NVIDIA Tesla P40 GPU and Intel 64-core CPU. It shows that DPSR can retrieve candidates within 10ms on CPU, and can benefit from GPUs with 85% reduction on indexing time consumption, 92% reduction on search latency and 14 times more QPS (query per second) throughput.

In Table 6, we report the overall model serving performance with the same CPU and GPU machines as above. The overall latency from query text to $1,000$ nearest neighbors can be done within 15 to 20 milliseconds for GPU or CPU machines, which is even comparable to the retrieval from standard inverted index.

7 CONCLUSION

In this paper, we have discussed how we build a deep personalized and semantic retrieval system in an industry scale e-commerce

Table 5: Latency for index building and search.

	index building (sec.)	search (ms)	QPS
CPU	3453	9.92	100
GPU	499	0.74	1422

Table 6: Overall serving performance.

	QPS	latency (ms)	CPU usage	GPU usage
CPU	4,000	20	> 50%	0.0%
GPU	5,800	15	> 50%	25%

search engine. Specifically, we 1) shared our design of a deep retrieval system, which takes all the production requirements into consideration, 2) presented a novel deep learning model that is tailored for the personalized and semantic retrieval problems, and 3) demonstrated that the proposed approach can effectively find semantically relevant items, especially for long tail queries, which is an ideal complementary candidate generation approach to the traditional inverted index based approach. We have successfully deployed DPSR into JD.com's search production since early 2019, and we believe our proposed system can be easily extended from e-commerce search to other search scenarios.

8 ACKNOWLEDGEMENT

We deeply appreciate Chao Sun, Jintao Tang, Wei He, Tengfei Guan, Wenbin Zhu, Dejun Qiu, Qi Zhu, Hongwei Shen, Wei Wei and Youke Li for their engineering support to build key components of the infrastructure, and Chen Zheng, Rui Li and Eric Zhao for their help at the early stage of this project. We thank the anonymous reviewers for their valuable suggestions.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *OSDI*. 265–283.

[2] Xiao Bai, Erik Ordentlich, Yuanyuan Zhang, Andy Feng, Adwait Ratnaparkhi, Reena Somvanshi, and Aldi Tjahjadi. 2018. Scalable Query N-Gram Embedding for Improving Matching and Relevance in Sponsored Search. In *SIGKDD*. 52–61.

[3] David M. Beazley. 1996. SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop* (Monterey, California). 15–15.

[4] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 169–174.

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys*. 191–198.

[6] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science* 41, 6 (1990), 391–407.

[7] 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).

[8] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12, Jul (2011), 2121–2159.

[9] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *CIKM* (Indianapolis, Indiana, USA). 55–64.

[10] Jiafeng Guo, Gu Xu, Hang Li, and Xueqi Cheng. 2008. A Unified and Discriminative Model for Query Refinement. In *SIGIR* (Singapore, Singapore). 379–386.

[11] Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652* (2017).

[12] Thomas Hofmann. 1999. Probabilistic Latent Semantic Indexing. In *SIGIR* (Berkeley, California, USA). 50–57.

[13] 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 *CIKM* (San Francisco, California, USA). 2333–2338.

[14] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. 2007. Evaluating the Accuracy of Implicit Feedback from Clicks and Query Reformulations in Web Search. *ACM Transactions on Information Systems* 25, 2, Article 7 (April 2007).

[15] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *CoRR* abs/1702.08734 (2017).

[16] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-Interest Network with Dynamic Routing for Recommendation at Tmall. In *CIKM*. 2615–2623.

[17] Hang Li and Jun Xu. 2014. Semantic Matching in Search. *Foundations and Trends in Information Retrieval* 7, 5 (June 2014), 343–469.

[18] Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. 2017. Cascade Ranking for Operational E-commerce Search. In *SIGKDD* (Halifax, NS, Canada). 1557–1565.

[19] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (March 2009), 225–331.

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

[21] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to Match Using Local and Distributed Representations of Text for Web Search. In *WWW* (Perth, Australia). 1291–1299.

[22] Christopher Olston, Fangwei Li, Jeremiah Harmsen, Jordan Soyke, Kiril Gorovoy, Li Lao, Noah Fiedel, Sukriti Ramesh, and Vinu Rajashekhar. 2017. TensorFlow-Serving: Flexible, High-Performance ML Serving. In *Workshop on ML Systems at NIPS*.

[23] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. 2016. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24, 4 (2016), 694–707.

[24] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. *Language Models are Unsupervised Multitask Learners*. Technical Report.

[25] Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Semantic Hashing. *International Journal of Approximate Reasoning* 50, 7 (July 2009), 969–978.

[26] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In *WWW* (Seoul, Korea). 373–374.

[27] Parikshit Sondhi, Mohit Sharma, Pranam Kolari, and ChengXiang Zhai. 2018. A Taxonomy of Queries for E-commerce Search. In *SIGIR*. 1245–1248.

[28] Daria Sorokina and Erick Cantu-Paz. 2016. Amazon Search: The Joy of Ranking Products. In *SIGIR*. 459–460.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*. 5998–6008.

[30] Thanh Vu, Dat Quoc Nguyen, Mark Johnson, Dawei Song, and Alistair Willis. 2017. Search personalization with embeddings. In *European Conference on Information Retrieval*. Springer, 598–604.

[31] Tensorflow Official Website. 2019. *Reading custom file and record formats*. https://www.tensorflow.org/guide/extend/formats

[32] Cassandra Xia, Clemens Mewald, D. Sculley, David Soergel, George Roumpos, Heng-Tze Cheng, Illia Polosukhin, Jamie Alexander Smith, Jianwei Xie, Lichan Hong, Martin Wicke, Mustafa Ispir, Philip Daniel Tucker, Yuan Tang, and Zakaria Haque. 2017. TensorFlow Estimators: Managing Simplicity vs. Flexibility in High-Level Machine Learning Frameworks. In *SIGKDD*. 1763—-1771.

[33] Ji Yang, Xinyang Yi, Derek Zhiyuan Cheng, Lichan Hong, Yang Li, Simon Xiaoming Wang, Taibai Xu, and Ed H Chi. 2020. Mixed Negative Sampling for Learning Two-tower Neural Networks in Recommendations. In *WWW Companion*. 441–447.

[34] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *RecSys*. 269–277.

[35] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, Jean-Marc Langlois, and Yi Chang. 2016. Ranking Relevance in Yahoo Search. In *SIGKDD* (San Francisco, California, USA). 323–332.

[36] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning Tree-Based Deep Model for Recommender Systems. In *SIGKDD*. 1079–1088.