Neural Product Retrieval at Walmart.com

Alessandro Magnani
Walmart Labs
Sunnyvale, United States
amagnani@walmartlabs.com

Feng Liu
Walmart Labs
Sunnyvale, United States
feng.liu@walmartlabs.com

Min Xie
Walmart Labs
Sunnyvale, United States
mxie@walmartlabs.com

Somnath Banerjee
Walmart Labs
Sunnyvale, United States
sbanerjee1@walmartlabs.com

ABSTRACT

For an E-commerce website like Walmart.com, search is one of the most critical channel for engaging customer. Most existing works on search are composed of two steps, a *retrieval* step which obtains the candidate set of matching items, and a *re-rank* step which focuses on fine-tuning the ranking of candidate items. Inspired by latest works in the domain of neural information retrieval (NIR), we discuss in this work our exploration of various product retrieval models which are trained on search log data. We discuss a set of lessons learned in our empirical result section, and these results can be applied to any product search engine which aims at learning a good product retrieval model based on search log data.

KEYWORDS

Product Search; Neural Information Retrieval

ACM Reference Format:
Alessandro Magnani, Feng Liu, Min Xie, and Somnath Banerjee. 2019. Neural Product Retrieval at Walmart.com. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW'19 Companion), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3308560.3316603

1 INTRODUCTION

Search is one of the most important channels for customers to discover products on an E-commerce website such as Walmart.com. Given our huge catalog which contains more than 100 millions products, how to help user surface the right products based on their query becomes a very challenging problem.

Existing literature on information retrieval focuses mostly on web search [18]. While product search shares many common challenges as with web search, there are many aspects in product search which are unique. For example, web search focuses on retrieving documents that satisfy a userâĂŹs *information need*, e.g., when searching for *when is the next olympic*, any web page from trusted source which describes this information will satisfy the user's need; whereas in product search, it serves users' *shopping need*, among many assorted products which match users' query, user will likely only buy one, thus the product search engine has to identify which one is the best.

Similar to existing general search engine, product search usually involves two steps, the first step is to *retrieve* all relevant products from the catalog which form the recall set; then these candidate products will go through a *re-rank* step to identify which products are the best to return to the customer. Most of the recent learning to rank efforts focus on the re-ranking step, and they leverage extensive features available and can potentially involve very complicated ranking model [17]. In the retrieval step, query is potentially matched against every document in the collection thus latency of the retrieval model is a big challenge, and we usually have to trade-off between model complexity and scalability of the retrieval model. In this work, we focus on the retrieval step.

For a classical search engine, retrieval is mostly based on the raw text and a heuristic score function such as Okapi BM25 [18]. Inverted index as implemented in Apache Lucene [1] can be leveraged to make retrieval fast. Matching product and query in this solution is based on exact matching of tokens, which may not be suitable for product search because of the vocabulary mis-match between query and product as found by many recent works [11, 27]. E.g., users might search for a u-disk, whereas the matching items in the catalog are named after flash drive. Many existing works are trying to solve this problem by incorporating knowledge graph [31] or having a dedicated query understanding component [9]. However, these approaches need huge amount of domain expertise, and the cost of maintaining these components is very high as the product vocabulary and the catalog of the E-commerce websites change frequently.

In this work, inspired by recent progress in *neural information retrieval* (NIR) [20], we consider a few end-to-end neural product retrieval solutions which can be trained directly on abundant search query log which is available to any product search engine.

The first model we consider builds its foundation directly on the raw text of the underlying query and product. This model captures directly the text level similarity between query and product, then applies neural transformation on the shared text representation to predict the relevance between query and product. While non-linear transformation can capture the more complicated correlation between text in query and product, still this approach suffers from the need of exact matching of tokens in the text.

Our second model leverages a semantic embedding layer to first transform texts of query and product to their corresponding high dimensional embedding representation, then we apply a neural transformation on the concatenated query and product embedding vector. This model is inspired by recent advancement in word embedding [19], and has the most powerful modeling capacity. However, when applying to the search retrieval problem, applying the neural transformation to query and each product in a huge catalog can be prohibitive.

Finally, we propose a neural retrieval model which applies neural transformation to obtain vector representation on product and query separately, and then apply a simple aggregation function to combine the two vectors to predict the final score. Unlike the second approach, scoring query and product representation in the neural retrieval model can be as simple as vector multiplication. Thus it can be handled efficiently in the product search engine based on Apache Lucene [1], or use fast approximate nearest neighbor search library such as Faiss [15].

We also discuss in our work how we leverage a character level tokenization method inspired by [11, 27], how different rank loss functions such as point-wise loss and pair-wise loss can be implemented in our framework, and how we can incorporate additional product level attribute information such as brand, price, click-through rate which are critical to the search ranking problem into our model. We share in our experiment results section lessons we learned by applying our proposed solutions on a big search log dataset from Walmart.com. We believe these results can be applied to any product search engine which aims at learning a good product retrieval model based on search log data.

The following of the paper is organized as follows: we first discuss a few related works in Section 2, then we present the proposed product retrieval models in Section 3. In Section 4 we present our empirical evaluation results of our proposed solutions. And finally we conclude and discuss future works in Section 5.

2 RELATED WORK

In [26], the authors discuss that product search in E-commerce requires specialized solutions and challenges in ranking products. In [8], the authors propose a keyword based retrieval solution for product search, the idea is to generate keywords related to each product based on query logs and reviews, and then index these keywords based on inverted index. Unlike our end-to-end solution, generating keywords for products can be tricky as it involves tuning of various thresholds and domain expertise. Brenner et al. studies a end-to-end product retrieval problem on Jet.com in [4]. In this work the authors explore a similar setting as our neural retrieval model as discussed in Section 3, however, unlike our work which leverages the abundantly available raw query product engagement data, the authors explore labels which are obtained from query refinement data which are inferred from online user sessions. Though potentially the quality of this data is high, the challenge of this approach lies in the fact that the inferred query refinement data is very sparse. Some recent efforts in product search explore some interesting but orthogonal directions such as multiple objective [29] and product substitutability [10].

Neural information retrieval (NIR) has been a popular topic in the search community recently. It involves a set of sub-topics such as unsupervised learning of text embedding such as word2vec [19], deep autoencoder-based model such as semantic hashing [24], deep siamese model based on query log such as CLSM [27] and DSSM [11], and query document interaction-based model such as kernel pooling [30]. A good summary of the field can be obtained in [20]. Our neural retrieval model in Section 3 is closely related to CLSM and DSSM. Similar to [4], we are focusing more on the problem of product retrieval, and exploring solution which can strike a good balance between retrieval performance and model complexity.

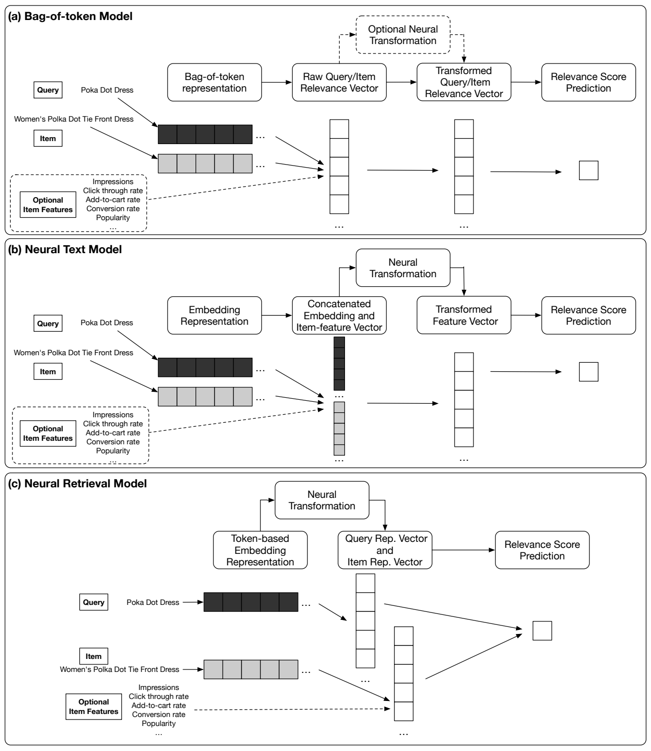
(a) Bag-of-token Model
Optional Neural Transformation
Bag-of-token representation
Raw Query/Item Relevance Vector
Transformed Query/Item Relevance Vector
Relevance Score Prediction
Query
Poka Dot Dress
Women's Polka Dot Tie Front Dress
Item
Optional Item Features
Impressions
Click through rate
Add-to-cart rate
Conversion rate
Popularity
(b) Neural Text Model
Neural Transformation
Embedding Representation
Concatenated Embedding and Item-feature Vector
Transformed Feature Vector
Relevance Score Prediction
(c) Neural Retrieval Model
Neural Transformation
Token-based Embedding Representation
Query Rep. Vector and Item Rep. Vector
Relevance Score Prediction

Figure 1: Models for neural product retrieval

3 NEURAL PRODUCT RETRIEVAL

To solve the problem of product retrieval, the key is to assess the relevance between a query and a product. In this section we propose three different models: A simple bag-of-token model based directly on tokens in query and product, a neural text model which can incorporate semantic similarity between query and product, and lastly a customized neural text model which is more optimized for retrieval. We also discuss how we tokenize text information of query and product for processing, how we can incorporate additional critical product information such as product price, brand and click-through rate into the model, and the different loss functions we have tested for the model.

3.1 Token Representation for Text

One simple way to represent text is to consider it as a bag of words which are contained in the text. The problem with this approach is that usually the vocabulary size is large, and this approach is difficult to handle mis-typed words in user typed queries.

In this work, we consider a token representation approach at sub-word level based on letter-trigram (LTG), which is inspired by [11, 27]. In this approach, for an example query "red dress", we first derive the letter-trigrams from each word, e.g., "red" will be broken down to three letter-trigrams, "#re", "red", "ed#", where # is the special word boundary character added to each word; and then the overall text will be represented as a bag of letter-trigrams. Our approach is also in part inspired by recent effort on subword embedding which demonstrated good performance in a set of natual language processing tasks [3].

3.2 Bag-of-Token Model

Let q be a query, and p be the text information about a product, based on the discussion from previous section, we can represent q and p as bags of tokens, where tokens are formed of letter-trigrams. Then we can apply one-hot encoding to each token and obtain a dense count vector for q and p as $q = [q_1, \ldots, q_n], p = [p_1, \ldots, p_n]$, where each q_i/p_i is the count of the corresponding token appears in q/p, and n is the size of the vocabulary.

A basic bag-of-token based scoring function can feed the query product pair through a simple linear model as below, where w and b are weight parameter and bias term of the model, and $\circ$ is element-wise multiplication.

$$\mathcal{F}_{BOT}^{linear}(q, p) = w \cdot (p \circ q) + b \tag{1}$$

We call this model BOT, short for Bag-of-tokens model. Essentially, this basic model examines the shared tokens between query and product, then it makes prediction based on this shared information using a linear model. This approach can be thought of as a simplified version of the vector space model for information retrieval [25].

The basic model above does not have the capacity to model the complex interaction between tokens and their correlation with the label. Thus we extend the basic approach by replacing the linear scoring component with a neural network model which is shown in Figure 2. In this model, we feed the aggregated count vector between query and product through N hidden layers of a neural network in the forward pass, Specifically, each hidden layer computes:

$$a^{(l+1)} = \phi(W^{(l)} a^{(l)} + b^{(l)}) \tag{2}$$

where l is the layer number and ϕ is the ReLU activation function [21]. $a^{(l)}$, $b^{(l)}$, and $W^{(l)}$ are the activations, bias, and model weights at l-th layer. We also added batch normalization [12] after each dense layer, and a dropout layer [28] at the end. This approach is inspired by Deep Averaging Network [13], and we call this model BOT-N (Bag-of-tokens with Neural Transformation).

We note that though BOT-N can model more complex interactions between tokens in query and product, but text information on product alone, e.g., title, usually are not enough for distinguishing good products from bad products. Two products can have the exact same title like polka dot dress but huge gaps in terms of sales. Some structured attributes of products usually are very important for the search task, e.g., brand of the product, price of the product, click-through rate of the product, add-to-cart rate of the product, and number of orders of the product. Thus we consider extend above models with these additional product level features by appending the transformed product features as also the input to the model, we call the extended model BOT-N$^+$. A list of key product features

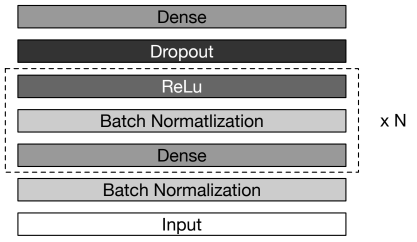

Figure 2: NN transformation

Feature Name	Feature Description
Brand	Brand for the product
PT	Product type for the product
Dept	Department of the product
Price	Price of the product
Popularity	Popularity of the product in terms of orders
CLK	Number of clicks for the prodoct
CTR	Click through rate for the product
ATC	Number of add to cart for the product
ATCR	Add to cart rate for the product
ORD	Number of orders for the product
ORDR	Order rate for the product

Table 1: Key product features used as model input.

used in this work is listed in Table (1), and all feature values in this work were collected in a 6 month window in 2018.

The architecture of the BOT model is shown in Figure 1 (a).

3.3 Neural Text Model

The bag-of-token models score query product pairs by directly considering tokens involved in the underlying query and product. The problem of this approach is that it has limited capability of modeling semantic similarity relationships between tokens, e.g., synonyms such as nightgown versus girls' pajama is difficult to be handled.

Inspired by recent advancement in text embedding [19], we consider a model architecture which first transform the input text to a embedding vector using either a simple convolutional neural network (CNN) similar to [16] or a simple embedding layer learned from scratch. For the simple embedding layer based approach, we use average pooling to obtain the embedding representation for the entire text piece, similar to deep averaging network [13] as used in universal sentence encoder [5]. In our experiment, we found that for our dataset mostly the simply embedding layer based approach will win against the more complicated CNN-based approach, and the CNN-based approach will take way longer time for training, thus we will skip the discussion of this approach in this work.

For a query product pair, we obtain the two corresponding embedding vectors, and then we concatenate the two embeddings and apply a neural transformation of the concatenated embedding vector through a few fully connected layers as shown in Figure 2.

We call this model Neural Text Model or NTM. Similar to the bag-of-token model, we can incorporate product features in the

the model as well for predicting query product relevance. The way we incorporate product feature is very similar to the bag-of-token model, we directly append the product features as input to the embeddings of query and product text, and apply neural transformation layer on top of all embedding the product features.

3.4 Neural Retrieval Models

As discussed in the introduction, BOT-N and NTM models all involve complex neural transformation based on the query and product information. This will introduce significant overhead when implementing the functionality in practical product retrieval application. This is because even though product features and embedding can be pre-computed, query tokens and embeddings have to be generated on the fly, and thus we have to score potentially all products againt query embedding through the more complicated transformation to find which products might have a better matching score w.r.t. the query. This is why even though more complicated learning to rank methods such as Kernel Pooling [30] has good performance on ranking, they can only be applied in the re-ranking phase when doing product search as discussed in [4].

Based on this intuition, we consider a modified neural text model named Neural Retrieval Model (NRM), in which we generate embedding representation of product and product separately, and feed both embeddings into a simple differentiable transformation function such as *cosine* or *dot-product*. The architecture of NRM is shown in Figure 1 (c). And similar to BOT and NTM, we can easily add product features as input when generating product embedding representation, we call this extended model NRM$^+$.

3.5 Model Training

We consider training our model directly on the abundant query log data from product search engine. Similar to many existing works in *learning-to-rank* [17], we consider point-wise approach and pair-wise approach for fitting our product retrieval model.

For the point-wise approach, given a pair of query q and product p, we can apply any of the models proposed in this section to obtain a score $\hat{y}_{q,p} = \mathcal{F}(q,p)$. Then in the loss function, we fit $\hat{y}_{q,p}$ through sigmoid cross entropy and against a label y which is the click-through rate of query q and product p.

$$L(\hat{Y}, Y) = -\sum_{q,p,y} y \times \log(\sigma(\hat{y}_{q,p})) + (1-y) \times \log(1-\sigma(\hat{y}_{q,p})) \quad (3)$$

For the pair-wise approach, we adopt the logistic pair loss defined below, where $\mathbb{I}$ is the indicator function.

$$L(\hat{Y}, Y) = \sum_{q,p_i,p_j,y} \mathbb{I}(\hat{y}_{q,p_i} > \hat{y}_{q,p_j}) \log(1 + exp(\hat{y}_{q,p_i} - \hat{y}_{q,p_j})) \quad (4)$$

To determine the training hyper parameters and to avoid overfitting, we divide the data into two sets that do not overlap, called training and validation datasets, respectively. Details of the data preparation will be discuss in Section 4.

We use Adagard as optimizer for training our model, and we set learning rate to be either 0.128 or 0.02 depending on setting. In our implementation, models are trained using a small GPU cluster with Nvidia Tesla V100.

3.6 Handling Position Bias

One challenge of leveraging implicit query log data is the position bias as discuss in recent literature [6]. Intuitively, products which are ranked higher by the existing algorithm will accumulate more clicks because they are more likely to be examined by customers; whereas products shown lower in the search result page will likely get less attention. In this work, we consider a simple click model which estimates the times that product is observed by a user only as a function of impressions and position. In our click model, the number of *examines* of a query q and a product p is estimated using the following formula:

$$\text{examine}_{q,p} = \sum_i \eta_i \times \text{impression}_i \quad (5)$$

In this equation, we sum over all positions i where impression_i is the number of impression at position i for query q and product p. We weight the number of impressions in the log data by a global position bias term η_i which is fitted empirically. Intuitively, η_i will weight the impression count lower if the position i is high, and it will weight the impression higher if the position i is low.

We then compute the click-through rate as the ratio of clicks and examines and order rate as the ratio of orders and examines. We can also use the examine number for each query product training instance to weight either the point-wise loss or pair-wise loss. This weighting scheme will capture both position bias, and also it will capture the confidence of each training instance, as training instance with more impressions/examines we are more confident on the data, and it will have a higher weight value.

4 EMPIRICAL EVALUATION

The implementation of our product search models is based TF-Ranking [22], which is a recently proposed open source library for solving large-scale ranking problems in Tensorflow [2]. In the following of this section, we first present the dataset used in our work, then we discuss metrics we used in our evaluation, and finally we present our findings by comparing the different product search models presented in this work.

4.1 Dataset

We collected a large query log dataset on shoes segment during a six month window from May 2018 to October 2018 on Walmart.com. Historical data of the extra features such as clicks and orders are collected from the query log six months before May 2018. The dataset is composed of more than 100 million query and product pairs, of which there are more than 1 million unique queries and more than 1 million unique item titles. We randomly split the data into training and validation dataset with the ratio 92.5%:7.5%, and we make sure there are no shared query and product pairs in the training and validation dataset.

4.2 Metrics

The first metric we use to evaluate our model is the standard ranking metric NDCG (Normalized Discounted Cumulative Gain) [14]. We consider NDCG at different positions 3, 5, and 10.

$$\text{DCG} = \sum_{j=1}^{n} \frac{2^{y_j} - 1}{\log_2(1 + r_j)}, NDCG = \frac{1}{N} \sum_{i=1}^{N} \frac{DCG}{DCG_{\pi^*}} \quad (6)$$

Where r_j and y_j is the rank and the score respectively of item j. As score we use the ratio of orders and examines, where examines are defined as in Section 3.6. Note that though we are optimizing for order rate, we did not use order rate as label for training because order data is sparser compared with click data.

For product search we usually want to rank those products which have higher order rate at top, thus this inspired us to consider another metric Pair Accuracy which measures in the validation set that for a query q, of all product pairs p_i and p_j where p_i has a higher order rate compared with p_j, how many of them the model will predict correctly.

We also track some other metrics such as ARP (Average Relevance Position), the result is very similar to NDCG and Pair Accuracy. Thus we skip the discussion of those numbers in this paper.

4.3 Bag-of-Token Model

As expected the basic BOT model which is based solely on the shared tokens between query and product, does poorly on all metrics, this is understandable since the modeling power of this baseline without any transformation is very limited. Thus we use the metric values from BOT as our baseline, and we show only the improvement over BOT for other models on all metrics.

We show our experiment results of the BOT-N models in Table 2. As can be seen from this table, For BOT-N we added in our experiment three dense layers which serve as the neural transformation on the shared token information, and these added transformation layers can improve significantly on the performance over the basic BOT. Adding additional product level features in BOT-N$^+$ will help lift all metrics compared with BOT-N, which is again expected, as based on our experience, the added features are usually highly correlated with the metric we are evaluating.

4.4 Bag-of-Token Model vs. NTM Model

In Table 2, we also show the comparison of the more advanced neural text models with our bag-of-token models. From the table, it is very clear that the added token embedding layer boosts the metrics over the BOT significantly. This matches our intuition as these NTM models has more modeling capacity based on their learning of token embedding representation that can capture semantic similarity relationship between them. Also we can observe that similar to BOT-N vs. BOT-N$^+$, adding additional product features also helps on all the metrics we are tracking.

4.5 Pair Loss vs. Point Loss

We compare the pair-wise loss function and point-wise loss function in Table 3. And as can be easily observed from this table, the pair-wise loss function introduces a better performance compared with point-wise loss function. This should be intuitive as all metrics we are tracking are all based on ranking, it matches the direction of the pair-wise loss function better.

4.6 NTM Model vs. NRM Model

To compare pair loss with point loss, we show in Table 3 the performance comparison results for NTM and NRM. The difference between these two models essentially lie in the fact that whether we perform a complex transformation layer once query and product embedding representation have been obtained. Intuitively, NTM should perform better as the additionally introduced transformation layers can help matching query to its relevant products. However, as discussed in the previous sections, the important benefit of NRM lies in its efficiency gain in practical implementation. As query embedding and product embedding in NRM can be matched with simple operations, we can leverage fast high dimensional approximate nearest neighbor retrieval libraries such as Faiss [15]. As shown in Table 3, as expected, the metric performance with NRM is worse than the NTM models, however, it is not far behind, and still comparable to many models we considered in the experiment. We think NRM with pair loss strikes a good balance between modeling power and empirical performance.

4.7 Head Query vs. Tail Query

We also consider weighted NDCG and weighted pair accuracy metric which are weighted based on query traffic, thus naturally these weighted metrics are more leaning towards head queries, whereas the non-weighted metrics are more leaning towards tail queries. In Table 4, we show a comparison of weighted metrics vs. non-weighted metrics for NTM and NTM/Pair Loss. As can be seen from the table, it is very consistent that most of the case our models have a better performance on weighted metrics compared with the corresponding non-weighted metrics. This can be simply explained by the fact that in the training data, we weight training examples based on impression, thus naturally the model is putting heavier emphasis on the training instances with more query traffic.

5 CONCLUSION AND FUTURE WORK

Inspired by latest works in the domain of neural information retrieval (NIR), we discuss in this work our exploration of various product retrieval models which are trained on search log data. We present a set of lessons learned based on our exploration that can be easily applied to any product search engine.

In the future, we plan to improve our retrieval models by exploring multiple directions. Though letter trigram based tokenization has the benefit of having a small vocabulary and being able to naturally handle mis-typed user queries better, it cannot leverage word-level embedding pre-trained on large text datasets. We are exploring to incorporate the latest text embedding works such as ELMo [23] and BERT [7] into our models. Also while experimenting with our neural retrieval model, we noticed that the underlying neural network structure has heavy impact on the performance of the model, we plan to investigate more on this topic in the future. Finally we also plan to explore the use of other click models to better capture user interaction with the site.

REFERENCES

[1] 2019. Apache Lucene. http://lucene.apache.org.

[2] Martín 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 Gordon Murray,

Model	NDCG@3	NDCG@5	NDCG@10	Pair Acc.
BOT-N	19.84%	19.00%	15.33%	3.83%
BOT-N$^+$	34.65%	32.02%	26.42%	12.80%
NTM	63.18%	58.04%	53.04%	32.65%
NTM$^+$	**63.72%**	**60.77%**	**54.21%**	**33.90%**

Table 2: Bag-of-token model vs. Neural text model.

Model	NDCG@3	NDCG@5	NDCG@10	Pair Acc.
NTM	63.18%	58.04%	53.04%	32.65%
NTM/Pair Loss	**69.36%**	**66.36%**	**59.69%**	**35.67%**
NRM/Pair Loss	45.98%	46.37%	42.94%	32.41%

Table 3: Performance of neural text model vs. neural Retrieval Model

Model	NDCG@3	NDCG@3 Weighted	NDCG@5	NDCG@5 Weighted	NDCG@10	NDCG@10 Weighted
NTM	63.18%	125.87%	58.04%	128.44%	53.04%	114.37%
NTM/Pair Loss	69.36%	88.93%	69.36%	91.74%	59.69%	87.94%

Table 4: Performance of head query vs. tail query

Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* 265–283.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *TACL* 5 (2017), 135–146.

[4] Eliot Brenner, Jun Zhao, Aliasgar Kutiyanawala, and Zheng Yan. 2018. End-to-End Neural Ranking for eCommerce Product Search: an application of task models and textual embeddings. In *eCom.*

[5] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal Sentence Encoder. *CoRR* abs/1803.11175 (2018).

[6] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. *Click Models for Web Search.* Morgan & Claypool Publishers.

[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] Huizhong Duan, ChengXiang Zhai, Jinxing Cheng, and Abhishek Gattani. 2013. Supporting Keyword Search in Product Database: A Probabilistic Approach. *PVLDB* 6, 14 (2013), 1786–1797.

[9] Susan T. Dumais. 2016. Personalized Search: Potential and Pitfalls. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016.* 689.

[10] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. 2018. Mix'n Match: Integrating Text Matching and Product Substitutability within Product Search. In *CIKM.*

[11] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013.* 2333–2338.

[12] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015.* 448–456.

[13] Mohit Iyyer, Varun Manjunatha, Jordan L. Boyd-Graber, and Hal Daumé III. 2015. Deep Unordered Composition Rivals Syntactic Methods for Text Classification. In *ACL.* 1681–1691.

[14] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.* 20, 4 (2002), 422–446.

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

[16] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL.* 1746–1751.

[17] Tie-Yan Liu. 2011. *Learning to Rank for Information Retrieval.* Springer.

[18] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval.* Cambridge University Press.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).

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

[21] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel.* 807–814.

[22] Rama Kumar Pasumarthi, Xuanhui Wang, Cheng Li, Sebastian Bruch, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2018. TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank. *CoRR* abs/1812.00073 (2018).

[23] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL.*

[24] Ruslan Salakhutdinov and Geoffrey E. Hinton. 2009. Semantic hashing. *Int. J. Approx. Reasoning* 50, 7 (2009), 969–978.

[25] Gerard Salton, A. Wong, and Chung-Shu Yang. 1975. A Vector Space Model for Automatic Indexing. *Commun. ACM* 18, 11 (1975), 613–620.

[26] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On Application of Learning to Rank for E-Commerce Search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017.* 475–484.

[27] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014.* 101–110.

[28] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[29] Liang Wu, Diane Hu, Liangjie Hong, and Huan Liu. 2018. Turning Clicks into Purchases: Revenue Optimization for Product Search in E-Commerce. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018.* 365–374.

[30] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017.* 55–64.

[31] Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit Semantic Ranking for Academic Search via Knowledge Graph Embedding. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017.* 1271–1279.