Semantic Search in E-Commerce using User Click-through Data

By Suraj Lekhraj Yadav

WORD COUNT 6196
TIME SUBMITTED 22-JUL-2022 01:53PM
PAPER ID 88695152

Semantic Search in E-Commerce using User Click-through Data

Suraj Lekhraj Yadav

Semantic Search in E-Commerce using User Click-through Data

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Suraj Lekhraj Yadav
[Roll No: CS-2036]

under the guidance of

Dr. Debapriyo Majumdar.
CVPRU

Indian Statistical Institute
Kolkata-700108, India

July 2022

CERTIFICATE

This is to certify that the dissertation entitled **"Semantic Search in E-Commerce using User Click-through Data"** submitted by **Suraj Lekhraj Yadav** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Dr. Debapriyo Majumdar
CVPRU,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I would like to show my highest gratitude to my internal advisor, *Dr. Debapriyo Majumdar*, CVPRU, Indian Statistical Institute, Kolkata and to my external advisor *Surender Kumar* (Senior Principal Data Scientist at Flipkart) for their guidance and continuous support and encouragement. They have motivated me with great insights and innovative ideas throughout the thesis work.

I would also like to thank Praveen Gupta (Senior Data Scientist at Flipkart) for his valuable suggestions.

Last but not the least, I would like to thank all of my friends from MTech CS for their help and support.

Suraj Lekhraj Yadav
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

We study the problem of semantic search in E-Commerce using the User Click-through data such that, for a given user query, show all semantically relevant products to user. Pure keyword based matching fails here due to various reasons like: lack of user query intent matching, misspelled queries etc. In this thesis, we propose a novel idea of computing product-product similarity among clicked products present in a same session along with query-product similarity. We train representation based two tower neural network model. We build and train a representation based two tower neural network model for semantic matching in query-product pair and in product-product pair. In two tower architecture, we explored both Siamese based and Non-Siamese based model. In Siamese based model weights are shared between query tower and product tower, in Non-Siamese based model weights are not shared between query tower and product tower. As query and product information text tend to be shorter in length, we used average pooling with word unigram to capture the short range linguistic pattern. We present better results as compared to Semantic and Lexical baseline models. Both Siamese and Non-Siamese approaches produced eventually same results but Siamese network based model has converged much faster than the Non-Siamese network based model. We present better F1 and Recall score as compared to baseline models.

Contents

Chapter 1

Introduction

In the current era, online shopping has become an integral part of our life. Users start their online shopping by entering a search query text which describes the product as per their understanding. The user experience is worsened when products that are displayed do not match the intent of the search query, and this may harm user's long-term engagement and trust. Therefore, shopping sites try to show the list of relevant products for a given query. There can be problems with user queries like misspelled queries, very short queries etc. Simple lexical matching can be one of the approaches to show the relevant products. In lexical matching we search for the same keywords in product description as present in the user query and return the set of matched products. There are some problems with the lexical matching approach as below:

- Misspelled Queries: As per some web search log [4, 5], 10-15% user queries have spelling mistakes. For example: if the query is 'colege bbag', lexical matching engine will not show any result here. We can use an advanced spell checker method here but if engine architecture takes care of it that will be easier.

- Intent Matching: Product descriptions vocabulary will be very professional in nature as compared to user's query which will be in daily language and this leads to vocabulary gap [24]. Capturing the intent of the query is very important here. For example: let the user want a glucometer but do not know the machine name so he will usually search for 'diabetes test machine'. If we use lexical matching here glucometer product will never be shown here because user query does not contain the word 'glucometer'. In such a scenario the user will go back and search the same term in google and get the correct machine name as glucometer and then search for it again in the shopping portal. Such interruptions are not good for shopping sites because users might go to some other site for a better user experience which affects the business. If we can understand the user's intent relevant products can be shown immediately, thus semantic matching is needed.

1.1 Problem Statement

In this research, we are building a Semantic Search Model using User Click-through data such that for a given user query, list of all relevant semantic products should be shown to the user.

We present neural network based model trained on large amount of user click-through data which captures the semantic representation between query and clicked products. Earlier work [17, 11] has been focused on building query-document or query-product semantic similarity. Our contribution include sessionization of user click-through data and then considering a co-clicked neighbouring products in a same session, then we build a product-product semantic similarity along with the query-product semantic similarity.

Our contribution can be summarized as follows:

- We propose a neural network based model which captures the semantic relationship between given query-product pair.
- With the help of sessionization, we build a sessions of some fixed time interval on a given user click-through data and then we consider the co-clicked products in a same session which helps us to build a semantic similarity between product-product pairs along with query-product pairs.
- We trained and tested this model on large amount of data, we observed better performance as compared to Semantic and Lexical baseline models.

Chapter 2

Related Work

There is a substantial literature on capturing the semantics of queries and relevant documents in Natural Language Processing (NLP). Word2Vec [14] drew a lot of attention for using word embeddings to capture semantic structure. Diaz et al. [7] trained neural word embeddings using concepts from Word2Vec to locate neighbouring words in order to extend the search with synonyms. Finally, the state-of-the-art semantic search models can typically be divided into below three groups based on these latest developments and other significant findings:

- Latent Factor Models: Query and document-level embeddings are learned via non-linear matrix completion techniques without accessing their content.
- Factorized Models/Representation Based Models: Using the content of queries and products convert them to low dimensional word embeddings separately.
- Interaction Models: Create interaction matrices between the query and product content. Then with help of neural networks extract patterns from the interaction matrix.

In order to find semantic ideas, Latent Semantic Analysis (LSA) [6] calculates a low-rank factorization of a term-document matrix. It was expanded and further improved by [1, 15] using concepts in Latent Dirichlet Allocation (LDA) [2]. Huang et al. [11] introduced the Deep Semantic Similarity Model (DSSM) by publishing in the space of factorized models. In DSSM, a deep neural network model is trained end to end using a discriminative loss in order to learn a fixed size representation for the query and products, which was based on LSA and Semantic Hashing [20]. In DSSM architecture, fully connected layers were later replaced by Convolutional Neural Networks (CNNs) [10, 21] and Recurrent Neural Networks (RNNs) [18]. The Deep Relevance Matching Model (DRMM) [9] employs an interaction matrix in order to capture local term matching using neural approaches. This alternative approach, which articulated the

idea of interaction models, has since been effectively expanded by MatchPyramid [19] and other approaches [12, 13, 14, 23, 25].

2.1 Deep Semantic Matching

With increasing popularity in Deep Natural Language Processing approaches, numerous neural network based architectures have been presented in recent years to solve the semantic gap issue brought up by conventional lexical matching. One of these approaches is representation based learning. The representation based models have the typical characteristics of two tower architecture. DSSM [11] is one of the representation based model which has two tower struct. In two tower architecture, both tower can jointly use either siamese or distinct neural netwo in order to create a semantic representation of query and products, then we use a simple matching function like cosine similarity to compute the similarity between query and product. Siamese Neural Network [3], which is often referred to as the twin neural network, is an artificial neural network that applies the same weights on two different types of input vectors to compute equivalent output representation. In case of Non-Siamese or distinct neural network based two tower architecture, query and product have their own distinct tower where weights between them is not shared and they learn their own representation.

This kind of two tower representation based approach has recently been explored in E-Commerce Semantic Product Search [17], here query and product are embedded as vectors using the representation based learning and after that matching is performed in the vector space. One set of stacked neural network layers represents "tower" here, we have such two towers one for query and one for product. This kind of approach favours in production settings, there we can precompute the embeddings for product as the number of products usually be fixed and thus we only calculate the embeddings for search query at real time and it accelerats the real time services.

2.2 Prod2Vec and Meta-Prod2Vec

In our approach, we are considering co-clicked products in a same session and performing product-product similarity among the based on some window size k. This idea is inspired by Word2Vec [16] approach, which is one of the most widely used methods for learning word embeddings from la text corpus using neural networks. Word2Vec captures the semantic structure in such a way that words in the corpus with similar contexts are situated close to each other in the vector space. Based on this, clicked products which share the same session can be semantically related as the queries in that small session time interval are usually semantically related to each other.

Prod2Vec [8] is based on Word2Vec approach, in which we apply Word2Vec algorithm on sequence of clicked products in a session, clicked products in the same session acts as a sentence and individual product as words. To create distributed representations of products, Prod2Vec approach employs the product co-occurrence information which is defined by sequence of purchased or clicked products in a session. However, Prod2Vec approach does not utilizes metadata information of products like product category, brand, color etc. Meta-Prod2Vec [22] leverages the product's metadata as a side information along with Prod2Vec approach. In this along with products their metadata information such as brand, category etc. passed through the network. This is very relevant for scenarios in which product occurrence in a session is very rare and metadata information can be used for semantic matching and recommending relevant products. In our proposed approach we have used product's category as a side information.

Chapter 3

Data

The dataset which we have used here is Flipkart's User Click-through data. It consists of user_id, query_text, clicked_products, unix_timestamp etc. Click-through data is daily logged and one hadoop based file is getting created for each day. There are many types of products present in Flipkart's catalog, for this thesis work we have only considered lifestyle products.

3.1 Sessionization of Data

In sessionization of data, we create sessions each of some fixed time interval t. As one of our approach is Prod2vec, in which we consider the co-clicked products in a same session, we create sessions by dividing user click-through data using some fixed time interval t for a given userid. This is based on the understanding that user may search for certain product with different queries for certain time intervel t and for a given user all those queries within time t may be semantically related and also all those clicked products in a session for these queries can be semantically related. This sessionization is done by aggregating user click-through data with respect to userid and then forming a sessions for a given user using time interval t. In our approach we have considered each session of 10 minutes.

3.2 Data Preparation

We have performed below steps for data preparation:

- Using PySpark merge the hadoop based avro files of given day which contains the user click-through data.
- Select only those records where at least one click is performed .

- Filter to get only lifestyle products as we are only considering lifestyle product for our research.
- Select only required columns from this like query_text, user_id, clicked_products, unix_timestamp.
- After performing above selection, write new file to hadoop based parquet format and load this to pandas for further processing.
- Form t minutes session for each user using the unix_timestamp information. Add new column session_number, which represents session number for a given user.
- Perform group by operation on columns session_number and user_id in order to aggregate queries and clicked products in each session.
- After above group by operation each row will have aggregated queries and their corresponding clicked products in array format.
- Final data format will be array of array in which each list contains query_text, corresponding clicked product_title and k left and k right neighbouring products from the session. Here k is the window size which represents how many left and right co-clicked products we are considering from a session to form a single data instance.

3.3 Data Format and Example

After performing all pre-processing steps our single instance data format will look like as below:

$$[q, p, p_1, p_2, ..., p_{2k}, label]$$

Here q represents query, p represents anchor product title text, p_1 to p_{2k} represent neighbouring products to p with window size k. $label$ represents if given data instance is a positive sample or negative sample, if $label = 1$ then it is a positive sample and if $label = 0$ then it is a negative sample.

We have explain below getting data instances from click-through data:

- Consider that a particular user has searched for two queries within some time t and clicked few products as below:

$$q_1 : p_{11} \quad p_{12}$$

$$q_2 : p_{21} \quad p_{22} \quad p_{23}$$

- After sessionization on time interval t, these two queries belong to same session:

$$[(q_1 : p_{11} \quad p_{12}), \qquad (q_2 : p_{21} \quad p_{22} \quad p_{23})]$$

- Now, We build a query and corresponding clicked product pairs:

$$(q_1, p_{11}) \quad (q_1, p_{12}) \quad (q_2, p_{21}) \quad (q_2, p_{22}) \quad (q_2, p_{23})$$

- Considering window size $k = 3$, we build our data instances as below:

$$[q_1, \quad p_{11}, \quad p_{12}, \quad p_{21}, \quad p_{22}, \quad 1]$$

$$[q_1, \quad p_{12}, \quad p_{11}, \quad p_{21}, \quad p_{22}, \quad p_{23}, \quad 1]$$
$$[q_2, \quad p_{21}, \quad p_{11}, \quad p_{12}, \quad p_{22}, \quad p_{23}, \quad 1]$$
$$[q_2, \quad p_{22}, \quad p_{11}, \quad p_{12}, \quad p_{21}, \quad p_{23}, \quad 1]$$

$$[q_2, \quad p_{23}, \quad p_{12}, \quad p_{21}, \quad p_{22}, \quad 1]$$

- These are the positive data instances and therefore have $label = 1$ at the end of the list. Now, we build negative data instances/samples by replacing the anchor product with some random product from different category and $label = 0$ as below:

$$[q_1, \quad p'_{11}, \quad p_{12}, \quad p_{21}, \quad p_{22}, \quad 0]$$

$$[q_1, \quad p'_{12}, \quad p_{11}, \quad p_{21}, \quad p_{22}, \quad p_{23}, \quad 0]$$
$$[q_2, \quad p'_{21}, \quad p_{11}, \quad p_{12}, \quad p_{22}, \quad p_{23}, \quad 0]$$
$$[q_2, \quad p'_{22}, \quad p_{11}, \quad p_{12}, \quad p_{21}, \quad p_{23}, \quad 0]$$

$$[q_2, \quad p'_{23}, \quad p_{12}, \quad p_{21}, \quad p_{22}, \quad 0]$$

- In this particular example, 5 positive and 5 negative data instances are formed from one session and each one of them will be single data instance input to our neural network model.

Chapter 4

Proposed Approach

Our approach is based on sessionization of user click-through data, representation based two tower architecture and prod2vec with metadata information. We sessionize the user click-through data by forming a session of t minutes for a given user. Within session query and clicked products are aggregated and data instances are formed based on window size k. These formed data instances will be input to our model. We are passing data instance which consist of query, anchor product and co-clicked neighbouring products and getting fixed vector representation of each. In two tower architecture, we are having embedding layer and from there on two separate fully connected feed forward networks. Query and product will be passed through their own network and embedding layer will be comm[20] to both. Figure 4.1 and Figure 4.2 represents the two tower architecture, in this left network represents query tower and right network represents product tower.

4.1 Neural Network Architecture

Our proposed model is based on representation based two tower architecture. We are having two variants of this architecture:

- Siamese based network
- Non-Siamese based network

In Siamese based network, same weights are shared while working on two different types of input vectors. We have two types input text data: query text and product text. So, query and products will pass through their own network but their weights will be shared as shown in Figure 4.1.

In Non-Siamese based network, weights are not shared between query tower and product tower. Query and product will learn their own representation initially and

will converge over the time during training. Figure 4.2 represents the Non-Siamese based two tower architecture.

As shown in Figure 4.1 and Figure 4.2, embedding layer is our first model component which consists of $|V| \times N$ parameters where V represents vocabulary and N is the embedding dimension. We use average pooling after getting word embeddings of query and product from embedding layer. Average pooling requires very less computation which reduces the training time and inference latency. Query and Product both tend be shorter and without long range dependencies and thus average pooling gives better result here. After embedding layer and average pooling, we get query or product dimension as $Batch\_Size \times N$. We used two hidden layers in both query tower and product tower. Finally we get an vector representation from last hidden layer from both query tower and product tower and compute the cosine similarity between them.

Consider Figure 4.1 and Figure 4.2, which represents Siamese Network and Non-Siamese Network respectively, in this architecture after embedding layer we have two set of stacked neural network layers one in left and one in right. Each of these stacked networks are fully connected Multi Layer Perceptrons. Usually one stacked network represents query tower and other stacked network represents product tower.

Let q represents the query, p represents product and they are represented as sequence of tokens. We generate tokens using word unigram based tokenization technique, let q is broken into n tokens and p is broken into m tokens as shown below:

$$q = (t_1, t_2, t_3, ..., t_n)$$

$$p = (t_1, t_2, t_3, ..., t_m)$$

We build a word to index dictionary from our vocabulary which assigns each token in query and product to its corresponding index. Now, each token will be encoded to embedding size N, thus query and product now represented as:

$$q : n \times N$$

$$p : m \times N$$

To get a fixed width representation for both query and product, we apply average pooling on query and product token representation. After average pooling we get both query and product representation as vector of size $1 \times N$, where N is the embedding size. As we are going to work with batch data, so in this case after average pooling we get both query and product representation as vector of size $Batch\_Size \times N$.

Query and product now pass through their own tower which is basically a fully connected Multi Layer Perceptrons. In our architecture we have used two hidden layers say $Hidden\_Layer1$ and $Hidden\_Layer2$. Let the dimensions for hidden layers be as below:

$$Hidden\_Layer1: \quad N \times L_1$$
$$Hidden\_Layer2: \quad L_1 \times L_2$$

For each batch of input data, $Hidden\_Layer1$ outputs vector of size $(Batch\_Size \times L_1)$ and $Hidden\_Layer2$ outputs vector of size $(Batch\_Size \times L_2)$.

From last hidden layer we get the vector representation for query, anchor product and all the co-clicked neighbouring products which we pass it to our loss function for cosine similarity computation.

In Figure 4.1 which represents Siamese Network, weights are shared, this implies that corresponding hidden layers in both query tower and product tower are identical, as the embedding layer is already shared, here we can say that they are practically same network.

Figure 4.2 represents Non-Siamese Network, here we design our network in such a way that embedding layer is common for both query and product but query tower and product tower have their own MLP network and weights are not shared between query tower and product tower. Thus, here both query tower and product tower learn their own weights during training using the same cosine similarity based loss function.

4.2 Loss Function

For semantic matching our objective is to calculate the similarity between two given vectors and make their embeddings closer if positive pair and farther if negative pair.

We are calculating similarity on two types of pairs: query-product pair and product-product pair. In product-product pair, one element will always be our anchor product p.

Consider our input data instance format:

$$[q, p, p_1, p_2, ..., p_{2k}, label]$$

As explained earlier, q represents query, p represents anchor product, p_1 to p_{2k} are neighbouring products and $label$ represents positive or negative sample depending on its value.

In this case, we calculate the cosine similarity between following pairs:

$$(p, q), (p, p_1), (p, p_2), ...(p, p_{2k})$$

Here, k represents the window size, we are considering k left and k right co-clicked neighbouring products in a session for a given anchor product.

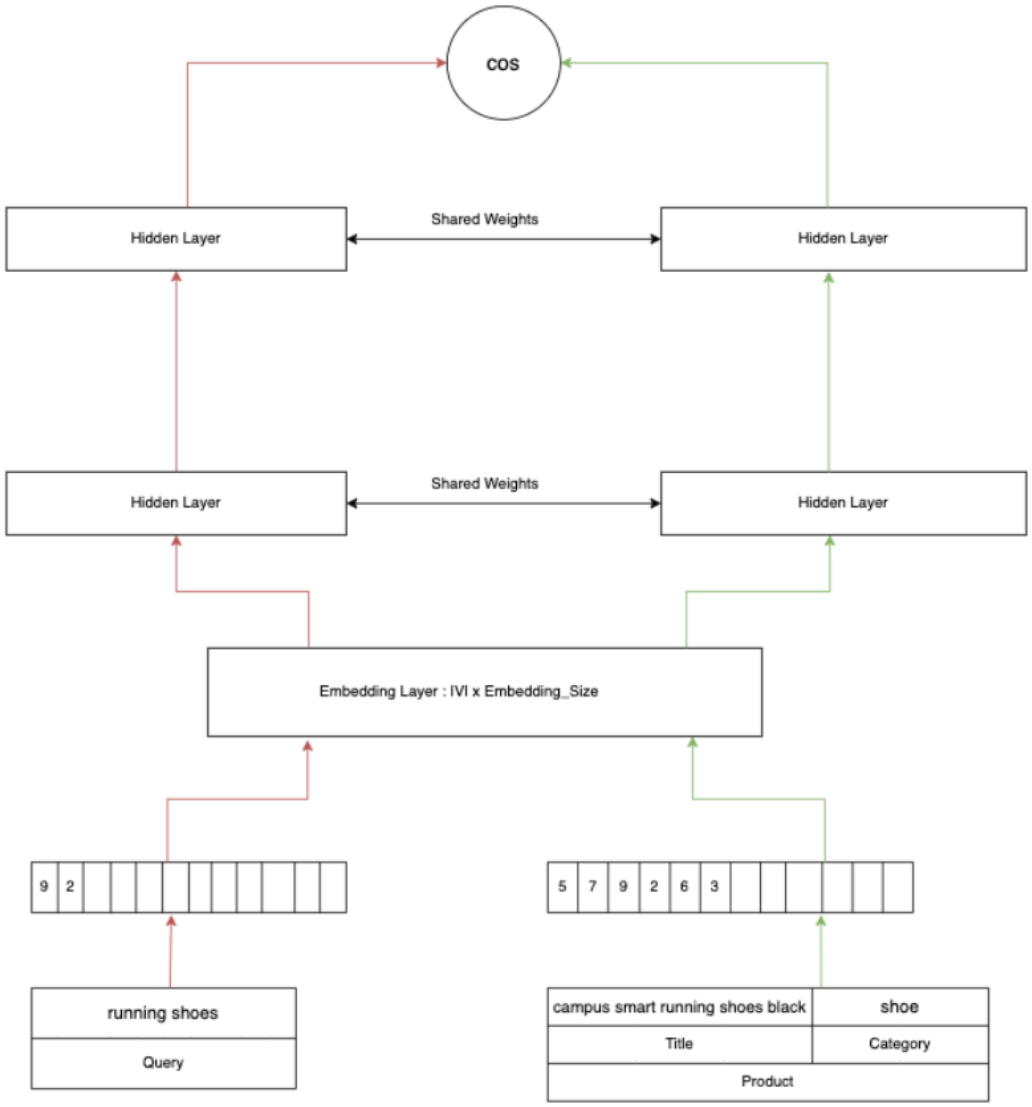

Figure 4.1: Siamese Network

Below expression represents our batch loss function:

$$batch\_loss = MSE(cos(p, q), label) + \sum_{i=1}^{2k} MSE(cos(p, p_i), label)$$

MSE represents mean square error, it is calculated based on cosine similarity between pairs and their corresponding label.

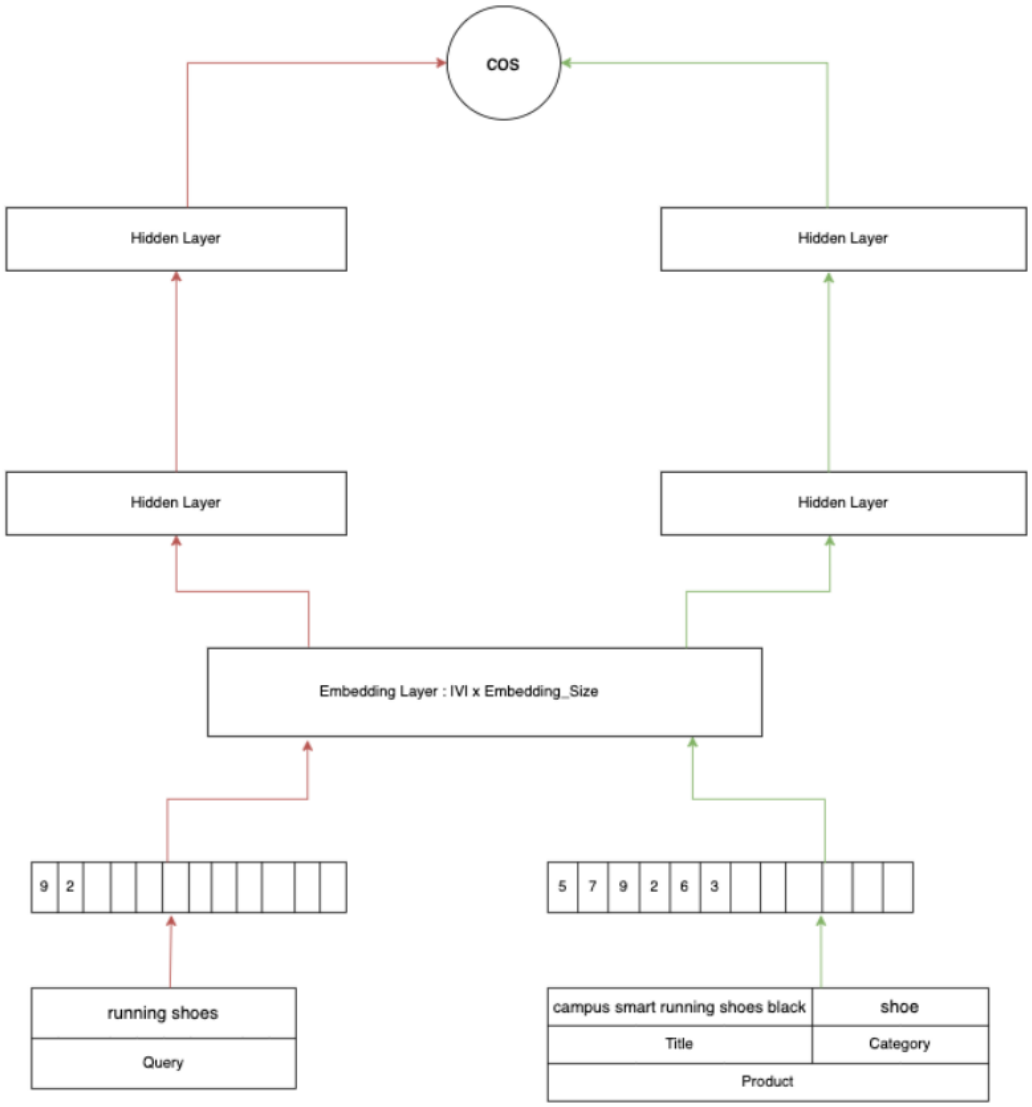

Figure 4.2: Non-Siamese Network

If $label = 1$, then it is a positive sample and we want to maximize the similarity between vectors present in a given pair, If $label = 0$, then it is a negative sample and we want to minimize the similarity between vectors present in a given pair.

4.3 Tokenization

In this section we explain the tokenization technique used in our approach. We are using query and product titles in our model, which will be in form of strings and can be comprised of few words. For example query can be "running shoes" and product can be "campus smart running shoes black". We tokenize these strings to form sequence of smaller components such as words, sub-words or characters. In our model we have used word unigram. We have also explored word bi-grams and character trigrams but it was significantly increasing the vocabulary size and taking huge amount of training time for a single epoch.

4.3.1 Word Unigram

This is a basic form of tokenization technique in which we break the given string into sequence of individual words. For example if word is "campus smart running shoes black" then its corresponding word unigrams are ["campus", "smart", "running", "shoes", "black"].

Chapter 5

Experiments

5.1 Training Dataset

As already explained we are using user click-through data for our model training. In E-Commerce site daily millions of queries are getting searched and thus our training corpus becomes very huge. After applying filters for clicked products and lifestyle products we still have almost 9 Millions records per day. We perform sessionization on this with 10 minutes interval and then consider the neighbouring products for each anchor product with window size as k. In our case we used $k = 3$ and we got almost 50 millions data instances for three days of data where each data instance consists of query, anchor product and neighbouring products.

5.1.1 Positive Samples

Each data instance in our training dataset is an array and last element of this array represent label of instance, if $label = 1$ then it is a positive sample and if $label = 0$ then it is a negative sample. Data instances which we get from user click-through data is a positive sample for us as these are the products which user clicked for a given query and we assume user clicks here as a relevant product.

5.1.2 Negative Samples

Each product belong to some category like 'shoe' , 'necklace' , 'shirt' etc. We generate negative data instances by replacing anchor product in positive data instance with the random product which belongs to some different category. For instance, consider below a positive data instance which we generated from user click-through data, here $k = 3$.

Positive Data Sample : [q, p, $p1$, $p2$, $p3$, $p4$, $p5$, $p6$, 1]

To generate the negative product we replace p with p', where p' is randomly selected from all the products such that its category is different from p, we get Negative Data Sample as below:

Negative Data Sample : [q, p', $p1$, $p2$, $p3$, $p4$, $p5$, $p6$, 0]

5.2 Testing Dataset

We used standard test dataset from Flipkart, which was manually annotated and consist of both relevant as well as non-relevant query-product pairs. It consist of total 741724 records out of which almost 79% are positive samples and 21% are negative samples.

5.3 Experimental Setup

In our model architecture, for both query and product text, we have used common embedding layer which consists of $|V| \times N$ parameters. We fix the embedding dimension to $N = 512$ and our vocabulary size is $|V| = 589936$.

As explained, after embedding layer we use average pooling to get fixed length embeddings for both query and product. After average pooling, for both query and product text we get dimension as $Batch\_Size \times N$, where $Batch\_Size = 1024$, $N = 512$ and thus after average pooling our dimension becomes 1024×512.

We have used two fully connected hidden layers for both query tower and product tower whose dimesnions are like below:

$$Hidden\_Layer1 : \quad 512 \times 256$$

$$Hidden\_Layer2 : \quad 256 \times 128$$

From last hidden layer we are getting vector of size 128 for a given query or product. We are using tanh as an activation function in each layer. We experimented with different hidden layer configuration and $Batch\_Size$ and above setup was optimal with respect to training time and results.

We used Pytorch to implement our model and train the model using Adam optimizer. The hyperparameters of Adam optimizer are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and the learning rate is set to 0.001. The batch size for training is set to 1024. We train our model on a Tesla V100-SXM2-32GB GPU card. Due to high volume of data each epoch took almost 3 hours for almost 100 million data instances. We considered 15 days of data here and at a given time we trained our model on 3 days of data, once done we took next 3 days of data and retrained our model on it and so on.

		S-MS (cut-off = 0.5)	Semantic Baseline (cut-off = 0.5)	Lexical Baseline (cut-off = 0.08)
Good Class [585353] 79%	F1	**0.86**	0.81	0.79
	Precision	0.82	**0.83**	0.84
	Recall	**0.9**	0.79	0.76
Bad Class [156371] 21%	F1	0.34	**0.37**	0.38
	Precision	**0.44**	0.34	0.33
	Recall	0.27	**0.41**	0.44

Figure 5.1: Siamese Network Result with Cut-Off = 0.5

5.4 Results

From Figure 5.1 to Figure 5.4, we have shown results of our model and compared them with Semantic and Lexical baseline models. In Figure 5.1 and Figure 5.2 we have shown results of our Siamese Network (S-MS) and Non-Siamese Network (NS-MS) respectively where cut-off = 0.5. In Figure 5.3 and Figure 5.4 we have shown results of our Siamese Network (S-MS) and Non-Siamese Network (NS-MS) respectively where cut-off = 0.6. Semantic and Lexical are baseline models against which we are comparing our results. Here cut-off represents the threshold at which we decide for relevant or non relevant product. For example, in test dataset for certain sample if cosine similarity between query and product is greater than cut-off value then we predict sample as relevant i.e. 1 else it is not relevant i.e. 0. Lexical Baseline is based on Jaccard Similarity where we compare only common words between query and product text, since it is text matching we use very low cut-off i.e 0.08. Semantic Baseline and our model (S-MS, NS-MS) is neural network based model which look for semantic matching.

We observe that at cut-off = 0.5, both our model performed much better than the baseline models in good class category which is 79% of total test data, our models are having a slightly low F1 score compared to Semantic Baseline in bad class category which is 21% of test data. Here good and bad class represents relevant and not relevant query-product pairs respectively in test data set.

We also observe that even at higher cut-off = 0.6, both of our model performed much better than the baseline models for which scores are computed at cut-off = 0.5. At cut-off = 0.6, both our model performed better than the baseline models in good class category and our models are having similar F1 score in bad class category. Thus, even at higher cut-off our proposed models has produced better results as compared to baseline models.

		NS-MS (cut-off = 0.5)	Semantic Baseline (cut-off = 0.5)	Lexical Baseline (cut-off = 0.08)
Good Class [585353] 79%	F1	**0.86**	0.81	0.79
	Precision	0.82	**0.83**	0.84
	Recall	**0.9**	0.79	0.76
Bad Class [156371] 21%	F1	0.34	**0.37**	0.38
	Precision	**0.44**	0.34	0.33
	Recall	0.27	**0.41**	0.44

Figure 5.2: Non-Siamese Network Result with Cut-Off = 0.5

		S-MS (cut-off = 0.6)	Semantic Baseline (cut-off = 0.5)	Lexical Baseline (cut-off = 0.08)
Good Class [585353] 79%	F1	**0.85**	0.81	0.79
	Precision	0.83	0.83	0.84
	Recall	**0.87**	0.79	0.76
Bad Class [156371] 21%	F1	0.37	0.37	0.38
	Precision	**0.38**	0.34	0.33
	Recall	0.36	**0.41**	0.44

Figure 5.3: Siamese Network Result with Cut-Off = 0.6

		NS-MS (cut-off = 0.6)	Semantic Baseline (cut-off = 0.5)	Lexical Baseline (cut-off = 0.08)
Good Class [585353] 79%	F1	**0.85**	0.81	0.79
	Precision	0.83	0.83	0.84
	Recall	**0.87**	0.79	0.76
Bad Class [156371] 21%	F1	0.37	0.37	0.38
	Precision	**0.38**	0.34	0.33
	Recall	0.36	**0.41**	0.44

Figure 5.4: Non-Siamese Network Result with Cut-Off = 0.6

Chapter 6

Conclusion and Future Work

We have introduced a novel idea for semantic search in E-Commerce by including product-product similarity for a co-clicked neighbouring products in a same session along with the query-product similarity. We looked into user click-through data in E-Commerce and pre-processed it to form a sessionize data. We introduced two architecture in this, one is Siamese network architecture where weights between query tower and product tower are shared, other being Non-Siamese network in which query and product are sharing the embedding layer but we have separate query tower and product tower and weights between them are not shared. We found that though both approaches are giving the same results eventually but Siamese network based model converging much faster than the Non-Siamese network based model. We observed that our results are better than the baseline models by getting better F1 score and better recall as compared to baseline models. We also observed that even at higher relevance cut-off our proposed models have produced better results as compared to baseline models.

In E-commerce, large amount of data is getting generated daily and this data can be very noisy. In our model we used complete daily data for training as data cleansing in this case is very time consuming. In future we can try for data cleansing methods which will be helpful to consider more relevant query product pair based on some threshold and thus initial data will be less noisy. We only tried product category as a metadata information in our model in future we can look for more metadata features like product brand, product parent category, product price etc. In E-Commerce, we look for very low latency systems, to maintain this we used multi-layer perceptron in our approach. We can explore RNN and Transformers based models in future where latency will be high, but results can be better. We did sessionization on the basis of userid, so in future we can utilize this to build models for personalized recommendation to a particular user.

Bibliography

[1] Berry, M.W., Young, P.G.: Using latent semantic indexing for multilanguage information retrieval. Computers and the Humanities 29(6), 413–429 (1995)

[2] Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. Journal of machine Learning research 3(Jan), 993–1022 (2003)

[3] Chicco, D.: Siamese neural networks: An overview. Artificial Neural Networks pp. 73–94 (2021)

[4] Cucerzan, S., Brill, E.: Spelling correction as an iterative process that exploits the collective knowledge of web users. In: Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing. pp. 293–300 (2004)

[5] Dalianis, H.: Evaluating a spelling support in a search engine. In: International Conference on Application of Natural Language to Information Systems. pp. 183–190. Springer (2002)

[6] Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American society for information science 41(6), 391–407 (1990)

[7] Diaz, F., Mitra, B., Craswell, N.: Query expansion with locally-trained word embeddings. arXiv preprint arXiv:1605.07891 (2016)

[8] Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., Sharp, D.: E-commerce in your inbox: Product recommendations at scale. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1809–1818 (2015)

[9] Guo, J., Fan, Y., Ai, Q., Croft, W.B.: A deep relevance matching model for ad-hoc retrieval. In: Proceedings of the 25th ACM international on conference on information and knowledge management. pp. 55–64 (2016)

[10] Hu, B., Lu, Z., Li, H., Chen, Q.: Convolutional neural network architectures for matching natural language sentences. Advances in neural information processing systems 27 (2014)

[11] Huang, P.S., He, X., Gao, J., Deng, L., Acero, A., Heck, L.: Learning deep structured semantic models for web search using clickthrough data. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management. pp. 2333–2338 (2013)

[12] Hui, K., Yates, A., Berberich, K., De Melo, G.: Pacrr: A position-aware neural ir model for relevance matching. arXiv preprint arXiv:1704.03940 (2017)

[13] Hui, K., Yates, A., Berberich, K., De Melo, G.: Co-pacrr: A context-aware neural ir model for ad-hoc retrieval. In: Proceedings of the eleventh ACM international conference on web search and data mining. pp. 279–287 (2018)

[14] Hui, K., Yates, A., Berberich, K., De Melo, G.: Co-pacrr: A context-aware neural ir model for ad-hoc retrieval. In: Proceedings of the eleventh ACM international conference on web search and data mining. pp. 279–287 (2018)

[15] Littman, M.L., Dumais, S.T., Landauer, T.K.: Automatic cross-language information retrieval using latent semantic indexing. In: Cross-language information retrieval, pp. 51–62. Springer (1998)

[16] Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)

[17] Nigam, P., Song, Y., Mohan, V., Lakshman, V., Ding, W., Shingavi, A., Teo, C.H., Gu, H., Yin, B.: Semantic product search. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2876–2885 (2019)

[18] Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., Ward, R.: 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), 694–707 (2016)

[19] Pang, L., Lan, Y., Guo, J., Xu, J., Wan, S., Cheng, X.: Text matching as image recognition. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 30 (2016)

[20] Salakhutdinov, R., Hinton, G.: Semantic hashing. International Journal of Approximate Reasoning 50(7), 969–978 (2009)

[21] Shen, Y., He, X., Gao, J., Deng, L., Mesnil, G.: 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. pp. 101–110 (2014)

[22] Vasile, F., Smirnova, E., Conneau, A.: Meta-prod2vec: Product embeddings using side-information for recommendation. In: Proceedings of the 10th ACM conference on recommender systems. pp. 225–232 (2016)

[23] Wan, S., Lan, Y., Xu, J., Guo, J., Pang, L., Cheng, X.: Match-srnn: Modeling the recursive matching structure with spatial rnn. arXiv preprint arXiv:1604.04378 (2016)

[24] Xiao, R., Ji, J., Cui, B., Tang, H., Ou, W., Xiao, Y., Tan, J., Ju, X.: Weakly supervised co-training of query rewriting andsemantic matching for e-commerce. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. pp. 402–410 (2019)

[25] Yang, L., Ai, Q., Guo, J., Croft, W.B.: anmm: Ranking short answer texts with attention-based neural matching model. In: Proceedings of the 25th ACM international on conference on information and knowledge management. pp. 287–296 (2016)

Semantic Search in E-Commerce using User Click-through Data

ORIGINALITY REPORT

11%
SIMILARITY INDEX

PRIMARY SOURCES

1	library.isical.ac.in:8080 Internet	172 words — 3%
2	arxiv.org Internet	137 words — 2%
3	Shaowei Yao, Jiwei Tan, Xi Chen, Keping Yang, Rong Xiao, Hongbo Deng, Xiaojun Wan. "Learning a Product Relevance Model from Click-Through Data in E-Commerce", Proceedings of the Web Conference 2021, 2021 Crossref	57 words — 1%
4	deepai.org Internet	40 words — 1%
5	link.springer.com Internet	18 words — < 1%
6	Yang Xiao-lin, Wang Jian-cheng, Yang Chu-yuan, Yuan Zun-li. "A New Fast Monte Carlo Code for Solving Radiative Transfer Equations Based on the Neumann Solution", The Astrophysical Journal Supplement Series, 2021 Crossref	16 words — < 1%
7	pt.scribd.com Internet	16 words — < 1%

8	Jiafeng Guo, Yinqiong Cai, Yixing Fan, Fei Sun, Ruqing Zhang, Xueqi Cheng. "Semantic Models for the First-Stage Retrieval: A Comprehensive Review", ACM Transactions on Information Systems, 2022 Crossref	13 words — < 1%
9	Shukla, Ravi. "Online Relay Switching in the Presence of Dynamic Obstacles in Millimeter Wave D2D Communication", Indian Statistical Institute - Kolkata ProQuest	13 words — < 1%
10	acikbilim.yok.gov.tr Internet	13 words — < 1%
11	www.sybase.com Internet	12 words — < 1%
12	"Advances in Information Retrieval", Springer Science and Business Media LLC, 2018 Crossref	10 words — < 1%
13	"Natural Language Processing and Chinese Computing", Springer Science and Business Media LLC, 2018 Crossref	9 words — < 1%
14	repository.tudelft.nl Internet	9 words — < 1%
15	"Advanced Data Mining and Applications", Springer Science and Business Media LLC, 2018 Crossref	8 words — < 1%
16	"Chinese Computational Linguistics", Springer Science and Business Media LLC, 2019 Crossref	8 words — < 1%

17	"PRICAI 2018: Trends in Artificial Intelligence", Springer Science and Business Media LLC, 2018 Crossref	8 words — < 1%
18	Guy Lev, Gil Sadeh, Benjamin Klein, Lior Wolf. "Chapter 50 RNN Fisher Vectors for Action Recognition and Image Annotation", Springer Science and Business Media LLC, 2016 Crossref	8 words — < 1%
19	aclanthology.org Internet	8 words — < 1%
20	export.arxiv.org Internet	8 words — < 1%
21	hal.archives-ouvertes.fr Internet	8 words — < 1%
22	icon2021.nits.ac.in Internet	8 words — < 1%
23	itlab.uta.edu Internet	8 words — < 1%
24	studentsrepo.um.edu.my Internet	8 words — < 1%
25	Tripathi, Yashaswi. "Aspect Based Sentiment Analysis in Text Reviews", Indian Statistical Institute - Kolkata ProQuest	7 words — < 1%
26	"Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data", Springer Science and Business Media LLC, 2017	6 words — < 1%

Crossref

27 "Natural Language Understanding and Intelligent Applications", Springer Nature, 2016 — 6 words — < 1%
Crossref

EXCLUDE QUOTES	ON	EXCLUDE SOURCES	OFF
EXCLUDE BIBLIOGRAPHY	ON	EXCLUDE MATCHES	OFF