POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

Master's thesis

JOINT MULTI MODAL QUERY-DOCUMENT REPRESENTATION LEARNING

Maciej Mościcki, 132290

Supervisor
dr hab. inż. Mikołaj Morzy prof. PP

POZNAŃ 2022

Tutaj będzie karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Streszczenie

Klasyczne systemy wyszukiwania informacji oparte na odwróconym indeksie oraz dopasowaniu leksykalnym sprawdzają się dobrze w przypadku zapytań korzystających ze słownictwa identycznego do tego obecnego w szukanych dokumentach. Systemy te mają natomiast problem z dopasowaniem zapytań do dokumentów zawierających słownictwo synonimiczne oraz słownictwo w innych formach fleksyjnych. Odpowiedzią na te problemy są alternatywne systemy wyszukiwania oparte o reprezentowanie zapytań oraz dokumentów w przestrzeni ukrytej. W systemach tych zarówno dokumenty jak i zapytanie reprezentuje się w przestrzeni ukrytej, a proces wyszukiwania sprowadza się wyszukiwania najbliższych sąsiadów w tej przestrzeni. Tworzenie reprezentacji zapytań oraz dokumentów we wspólnej przestrzeni można zrealizować z pomocą modeli uczenia maszynowego opartych o sieci neuronowe.

W pracy zaproponowano podejście do uczenia oraz ewaluacji modeli służących do tworzenia reprezentacji zapytań oraz dokumentów. Wykorzystując zbiory treningowy oraz ewaluacyjny, oparte o historyczne wyszukiwania z komercyjnej wyszukiwarki, zbadano jak dobór funkcji straty, architektura sieci, modalność cech oraz sposób tokenizacji tekstu wpływają na jakość tworzonych reprezentacji. Zaproponowano również sposób na ewaluację modeli pod kątem odporności na literówki we frazie wyszukiwania.

Z przeprowadzonych badań wynika, że dobór funkcji straty oraz jej hiperparametrów ma duże znaczenie dla jakości trenowanych modeli. Najlepszą funkcją straty okazała się funkcja *softmax cross-entropy loss*. Badania pokazały również, że dodanie do modelu cechy o innej modalności niż tekst, pozytywnie wpływa na jakość modelu. Ewaluacja różnych metod tokenizacji dowiodła, że wykorzystanie tokenizacji opartej o trójki znaków wraz z tokenizacją opartą na słowach, zwiększa odporność reprezentacji na literówki względem tokenizacji opartej jedynie o słowa. Zysk ten jest jednak znikomy, co sugeruje, że w komercyjnej wyszukiwarce korektę literówek należy przeprowadzić przed wyszukiwaniem. Model wytrenowany z najlepszymi parametrami porównano z wyszukiwarką wykorzystującą klasyczny algorytm BM25. Model osiągnął znaczną przewagę we wszystkich metrykach ewaluacyjnych, udowadniając dobrą jakość nauczonych reprezentacji. Na koniec dokonano empirycznej analizy zapytań oraz zwracanych dokumentów, która pokazała, że model potrafi dopasować dokumenty do zapytań na podstawie dopasowania semantycznego.

Contents

Chapter 1

Introduction

1.1 Limitations of classical retrieval systems

Information retrieval systems based on lexical matching and inverted index (e.g., based on *Lucene*) perform well with queries with exact text matches in a document corpus. These systems, however, lack the ability to perform semantic matching between queries and documents. To circumvent this problem, Information Retrieval engineers have to maintain carefully curated lists of synonyms (e.g., containing information that *photovoltaic* is synonymous to *solar*).

Lexical matching is also vulnerable to inflectional forms of words (e.g., *draw, drawing, drawer*). Stemming and lemmatization can reduce words to a common base form, but these techniques are hard to apply to the Polish language.

Furthermore, lexical matching performance is heavily reduced when queries are misspelled. To tackle misspelled queries, fuzzy matching can be used, but it has to be tuned carefully not to degrade the results' relevance.

1.2 Embedding Based Retrieval

In recent years, alternative retrieval systems based on embeddings are gaining popularity. In these systems, queries and documents are represented in the latent space, and the retrieval task is formulated as a nearest neighbors search in that space.

The advantages of such systems are:

- ability to perform semantic matching between queries and documents,
- partial immunity to inflection and misspelled queries,
- ability to include personalization context in the query.

These properties are advantageous to both search engine users - providing them with more relevant results as well as search engineers - freeing them of the need to create synonyms lists, tuning fuzzy matching, and query preprocessors.

1.3 Representation learning

The process of learning representations of queries and documents in the common latent space is called representation learning. One of the approaches for learning representations is to use neural networks. In this approach, query and document features are fed into neural networks called *encoders* to create their respective representations. Features fed to those models can be of

different modalities - e.g., text and categorical. There are many techniques used to train encoders, and evaluating them is crucial in order to achieve the best results.

1.4 Goal and scope of the thesis

This thesis aims to design, implement and compare the effectiveness of different techniques used to create representations of queries and documents in the same latent space. Specifically, the author's original contribution is:

- designing and implementing the training loop for optimizing selected metrics relevant to the search domain,
- designing and implementing different architectures for query and document encoders - including the ability to create representations from features with different modalities (e.g., text and categorical features),
- implementing different loss functions,
- implementing different text tokenization techniques,
- comparing the effectiveness of different model architectures and techniques with respect to selected metrics,
- designing a method to compare models' ability to handle misspelled queries,
- comparing models' ability to handle misspelled queries,
- conducting an empirical analysis of semantic matching between queries and documents.

1.4.1 Thesis structure

The structure of the thesis is the following. Chapter 2 explains the theoretical framework of embedding-based retrieval. Chapter 3 presents evaluation metrics used to evaluate model performance and introduces the training routine, datasets, and baseline model. Chapter 4 discusses experiment results and key findings on training encoders. Chapter 5 contains conclusions and future work ideas.

Chapter 2

Theoretical Framework

Traditional retrieval based on an inverted index and term matching is vulnerable to query-document term mismatch. Term matching fails to capture semantic matching between queries and documents [1].

Neural Information Retrieval can be used to capture the semantic matching between queries and documents. In this approach, neural networks generate representations of queries and documents that capture their semantics [2]. The process of learning the representations (embeddings) in the latent space is called representation learning [3]. With queries and documents represented as embeddings, the retrieval problem is turned into nearest neighbors search in the latent space [4].

Representation learning can be done in both supervised, and unsupervised settings [2]. In the unsupervised approach, representations are learned straight from text without labels. In the supervised approach, clickthrough data can be used as a source of user-labeled positive query-document pairs [5].

The model architecture widely adopted in the industry, in both search and recommendations, is called two-tower or dual encoder architecture [4, 6, 7, 8]. In this architecture, there are separate models for query and document, which learn query and document representations, respectively.

The common approach to training two-tower networks is to define the similarity function between queries and documents. Cosine similarity [6, 9] and inner product [4, 7, 8] are the two most commonly used. The loss function is then defined so that by minimizing it, the model pushes queries closer to the positive (relevant) documents in the latent space while simultaneously pushing away queries and negative (irrelevant) documents. Popular loss functions include triplet loss [6, 8, 9] and cross-entropy softmax loss [4, 7]. Negative query-document pairs are needed to avoid folding [10]. Given that clickthrough datasets contain only positive (clicked) query-document pairs, there are three main approaches to obtaining negative documents:

- batch negatives - given the batch of positive query-document pairs, for each pair, treat every other document in the batch as a negative [7],
- random negatives - for each batch of positive query-document pairs, randomly sample negative documents from the whole document pool [4, 6],
- mixed negatives - a mix of two previously mentioned techniques [8, 11].

When using text as an input feature to the encoder, different tokenization options can be considered. The simplest tokenization method is the word unigram tokenization [9]. Using character trigram tokenization can make the representation robust to typos and inflectional forms [6, 9].

Word bigram tokenization allows the model to capture token ordering information lost when average pooling of token embeddings is used [9]. Out-of-vocabulary tokens can be assigned to the fixed token ID, or hashing trick can be used to assign them to the range of token IDs [12].

Two-tower networks can be trained with multi-modal input features. This allows for including user context (personalization) in the query embedding [4, 8] as well as makes complex matching scenarios, such as query-image matching, possible [13].

Chapter 3

Methodology

3.1 Notation

The following notation is introduced for the rest of this chapter:

- q - query,
- d - document,
- d^+ - relevant (positive) document,
- d^- - irrelevant (negative) document,
- $id(d)$ - identifier of given document,
- T_D - training dataset - dataset of positive query-document pairs, $T_D = \{(q_1, d_1^+), (q_2, d_2^+), ..., (q_n, d_n^+)\}$,
- E_Q - evaluation queries dataset - dataset of positive query-document-id pairs, $E_Q = \{(q_1, id(d_1^+)), (q_2, id(d_2^+)), ..., (q_n, id(d_m^+))\}$,
- E_D - evaluation documents dataset - dataset of documents, including documents referenced in E_Q, $E_D = \{d_1, d_2, ..., d_j\}$,
- B - training dataset split into batches,
- Q_m - $f(q)$ query embedding,
- D_m - $g(d)$ - document embedding,
- $S(Q_m, D_m)$ - similarity function between query and document,
- $\mathcal{L}$ - loss function,
- $\mathcal{I}_\mathcal{D}$ - similarity search index.

3.2 Evaluation metrics

To evaluate model performance, two Information Retrieval metrics are selected - *Mean Recall at K* and *Mean Reciprocal Rank at K*.

Given a single query from query dataset Q, a set of relevant documents for that query, $R = \{r_1, r_2, ..., r_N\}$, and a list of top K documents returned by model for a given query, $D = \{d_1, d_2, ..., d_K\}$, *Recall@K* is defined as:

$$Recall@K = \frac{\sum_{i=1}^{K} d_i \in R}{N} \tag{3.1}$$

Mean recall at K is calculated as an average Recall@K over the whole query dataset and referred to as Recall@K during the rest of this work. Recall@K can be thought of as a percentage of relevant documents retrieved (e.g., 0.4 = 40%). Recall@K metrics are compared using percentage points throughout this work.

Given $rank = \min_i d_i \in R$, $RR@K$ for a given query is defined as

$$RR@K = \begin{cases} \frac{1}{rank}, & \text{if } rank \leq K \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

Mean reciprocal rank at K is defined as an average RR@K over the whole dataset and referred to as $MRR@K$ for the rest of this work.

3.3 Training and evaluation algorithm

The proposed training and evaluation algorithm for a single epoch is presented in Algorithm 1.

Algorithm 1 Training epoch

procedure TRAINING EPOCH
  for batch b in B **do**
    Sample a batch of negative documents D^-
    Compute query embeddings - $Q_m(q)$ for all queries in b
    Compute document embeddings - $D_m(d)$ for all positive documents in b as well as all negative documents from D^-
    Compute similarities between query and document embeddings - $S(Q_m, D_m)$
    Compute the loss function $\mathcal{L}$ given query-document similarities
    Perform backpropagation and update models' parameters
  end for
  Compute document embeddings - $D_m(d)$ for all documents in E_D and index them in $\mathcal{I}_\mathcal{D}$
  for $(q, id(d_i^+))$ in E_Q **do**
    Compute query embedding - $Q_m(q)$
    Perform nearest neighbour search using $Q_m(q)$ and $\mathcal{I}_\mathcal{D}$
    Compute evaluation metrics given result set $D = \{id(d_1), id(d_2), id(d_k))\}$ and relevant document id, $id(d_i^+))$
    Save computed metrics values
  end for
  Compute average value of evaluation metrics over whole E_Q
end procedure

3.4 Implementation details

The negative sampling strategy used in this work is batch negative sampling, i.e., for every batch b sampled from B, document $d_i \in b$ is used as a negative for $\forall(q_j, d_j^+), i \neq j$. Cosine similarity (Equation 3.3) is used as a similarity function between query and document embeddings.

$$S(Q_m, D_m) = cos(Q_m, D_m) = \frac{Q_m \cdot D_m}{||Q_m|| ||D_m||} \tag{3.3}$$

Batch size is empirically set to 64 as the best trade-off between quality and training speed. The optimizer used is Adam [14] with a learning rate of 0.01.

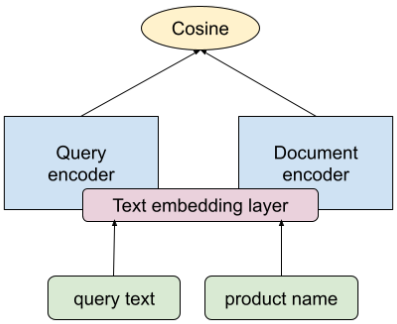


FIGURE 3.1: Baseline model

Models were trained using a single Tesla T4 GPU. The training loop was implemented using PyTorch [15]. FAISS [16] was used to create a similarity search index and perform nearest-neighbor queries.

3.5 Datasets

In this work, models were trained and evaluated on a proprietary dataset with clickthrough data from an e-commerce search engine. The training dataset consisted of 32M query-document pairs. The evaluation queries dataset consisted of 30k queries. The evaluation documents dataset consisted of 1.1M documents. Input features considered in this work are presented in Table 3.1.

TABLE 3.1: Features used in this work

Model	feature name	feature type	example
Query	search phrase	text	"dog toy"
Document	product name	text	"violet frisbee for the dog"
Document	product category id	categorical	"1234" ("dog toys")

Textual features were ASCII folded and lowercased. Punctuation marks and other non-alphanumeric characters were removed.

3.6 Baseline model

The proposed baseline model is a two-tower neural network. Both towers take textual features as input. Textual features are tokenized with the word unigram tokenizer. The token dictionary is computed before training and consists of 150k most popular tokens. Tokenized features are passed through an embedding layer. The embedding layer is shared between query and document features, allowing token matching to occur even before training. The embedding layer has a dimensionality of 200k x 256. 150k embeddings are assigned to tokens from the token dictionary, and 50k embeddings are used for out-of-vocabulary tokens, which are assigned to them using the hashing trick. Token embeddings are reduced using mean pooling and passed through a feed-forward network with a single hidden layer of size 128 with the Relu activation function.

Chapter 4

Experiments

4.1 Loss function

To evaluate different model architectures and tokenization techniques training loop must be completed with the best loss function. In this section, two popular loss functions: triplet margin loss and softmax cross-entropy loss, are evaluated.

4.1.1 Triplet margin loss

Given a dataset of N triplets (q_i, d_i^+, d_i^-) triplet loss is defined as

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \max\{0, D(q_i, d_i^+) - D(q_i, d_j^-) + m\} \tag{4.1}$$

where $D(u, v)$ is a "distance function" between vectors u and v. Considering that batch negatives are used, the loss function computed over a batch b of (q_i, d_i^+) pairs is defined as

$$\mathcal{L}(b) = \frac{1}{|b| * (|b| - 1)} \sum_{i=1}^{|b|} \sum_{\substack{j=1 \\ j \neq i}}^{|b|} \max\{0, D(q_i, d_i) - D(q_i, d_j) + m\} \tag{4.2}$$

With cosine used as a similarity function, the distance function is defined as

$$D(q, d) = 1 - cos(Q_m, D_m) \tag{4.3}$$

Parameter m is a *margin* hyperparameter representing the minimum value by which d_+ and d_- must be separated for the loss function to be 0.

Results

Several models were trained with triplet margin loss with different margin values. The results are presented in Table 4.1.

Table 4.1: Evaluation metrics for models trained with triplet margin loss with different margin

m	Recall@10	Recall@50	Recall@100	MRR@10	MRR@50	MRR@100
1.0	0.0057	0.0178	0.0270	0.0021	0.0027	0.0028
0.5	0.2670	0.4906	0.5894	0.1233	0.1339	0.1353
0.3	**0.2800**	**0.5097**	**0.6115**	**0.1308**	**0.1417**	**0.1431**
0.1	0.2406	0.4719	0.5721	0.1065	0.1176	0.1190
0.05	0.2284	0.4593	0.5637	0.0996	0.1106	0.1121

As shown in Table 4.1. tuning the margin loss has a significant impact on the model performance. The model trained with margin=0.3 performed the best across all evaluation metrics. Interestingly, the default margin value for PyTorch implementation is 1.0 [17], meaning that the model trained with a margin set to 0.3 outperforms the baseline by around 28 p.p. for the Recall@10 metric.

4.1.2 Softmax cross-entropy loss

As described in [18], retrieval can be treated as a multi-class classification problem. In this approach the likelihood of selecting a document d_i from document pool $D = \{d_1, d_2, ..., d_n\}$ given a query q, and similarity function $S(q, d)$ is defined as

$$P(d_i|q) = \frac{exp(S(q, d_i))}{\sum_{j=1}^{|D|} exp(S(q, d_j))} \tag{4.4}$$

The cross-entropy loss for a single (q, d_i) pair then becomes

$$\mathcal{L} = -\log(P(d_i|q)) \tag{4.5}$$

Considering that batch negatives are used, the loss function computed over a batch b of (q_i, d_i^+) pairs is defined as

$$\mathcal{L}(b) = \frac{1}{|b|} \sum_{i=1}^{|b|} -\log\left(\frac{exp(S(q_i, d_i))}{\sum_{j=1}^{|b|} exp(S(q_i, d_j))}\right) \tag{4.6}$$

Results

The results for the model trained with the softmax cross-entropy loss are presented in Table 4.2.

Table 4.2: Evaluation metrics for model trained with softmax cross-entropy loss

Recall@10	Recall@50	Recall@100	MRR@10	MRR@50	MRR@100
0.1119	0.2579	0.3402	0.0480	0.0548	0.0560

Following the work of [7], temperature parameter τ was introduced to improve model performance. The similarity function $S(q, d)$ then becomes

$$S(q, d) = \frac{cos(Q_m, D_m)}{\tau} \tag{4.7}$$

The results for models trained with different temperature values are presented in Table 4.3. Note that the loss function with temperature 1.0 is equivalent to that without the temperature parameter.

Table 4.3: Evaluation metrics for models trained with softmax cross-entropy loss with different temperature

τ	Recall@10	Recall@50	Recall@100	MRR@10	MRR@50	MRR@100
1.0	0.1119	0.2579	0.3402	0.0480	0.0548	0.0560
0.5	0.1818	0.3629	0.4622	0.0816	0.0902	0.0916
0.2	0.3731	0.5833	0.6682	0.1986	0.2088	0.2100
0.1	**0.4018**	**0.6230**	**0.7081**	**0.2087**	**0.2194**	**0.2206**
0.05	0.3625	0.6026	0.6947	0.1805	0.1921	0.1934

As shown in Table 4.3., introducing the temperature parameter has a significant impact on the model performance. The best model trained with $\tau = 0.1$ outperforms the model trained

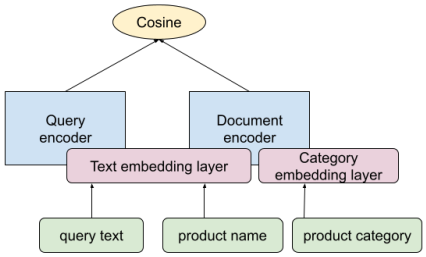


FIGURE 4.1: Multi-modal model

without τ by 29 p.p. for the Recall@10 metric. This means that the domain of the cosine function $cos(x) \in [-1, 1]$ combined with softmax heavily restricts the model performance.

4.1.3 Conclusions

From experiments conducted with both the triplet margin loss and the softmax cross-entropy triplet loss, it is observed that the model performance is very sensitive to their hyperparameters. Thus, tuning both of these loss functions is necessary to maximize the evaluation metrics.

TABLE 4.4: Evaluation metrics for model trained with different loss functions

Loss function	Recall@10	Recall@50	Recall@100	MRR@10	MRR@50	MRR@100
Triplet margin $m = 0.3$	0.2800	0.5097	0.6115	0.1308	0.1417	0.1431
Softmax cross-entropy $\tau = 0.1$	**0.4018**	**0.6230**	**0.7081**	**0.2087**	**0.2194**	**0.2206**

Table 4.4. summarizes the experiments with different loss functions. The best model trained with softmax cross-entropy loss outperforms the best model trained with triplet margin loss by 12 p.p. for the Recall@10 metric. Softmax cross-entropy loss with $\tau = 0.1$ is thus chosen as a loss function for the rest of this work.

4.2 Feature modality

As described previously, one of the strengths of neural information retrieval is the ability to incorporate different feature sets in query and document representations. Additionally, features can be of different modalities.

To test the effect of adding a feature of a different modality than text, a categorical feature (product category id) is added to the document model. The categorical feature is passed through an embedding layer. Its embedding is then concatenated with text embedding and passed on to the feed-forward network. The categorical feature has a total of 23k unique values. Its embedding size is empirically set to 32, as no gains from greater dimensionality are observed. The architecture being evaluated is visualized in Figure 4.1.

4.2.1 Results

Table 4.5. shows the results of the model with categorical feature (multi) and the baseline model (single).

Table 4.5: Evaluation metrics for models trained with features of different modalities

Modality	Recall@10	Recall@50	Recall@100	MRR@10	MRR@50	MRR@100
Single	0.4042	0.6315	0.7130	0.2101	0.2211	0.2222
Multi	**0.4229**	**0.6454**	**0.7280**	**0.2234**	**0.2342**	**0.2354**

The multi-modal model outperforms the model with only textual features across all evaluation metrics. Adding the categorical feature to the model increases the Recall@10 metric by 1.8 p.p. Multi-modal model is used for subsequent experiments.

4.3 Network depth and width

Having selected the multi-modal model as a new baseline model, the influence of encoders' width and depth on the evaluation metrics is examined. Namely, it is tested how the text embedding dimensionality and depth and width of hidden layers affect the evaluation metrics. The model architecture follows the one proposed by [18].

4.3.1 Results

Table 4.6 presents the evaluation metrics for models with different depths and widths. Model architecture is described by the width of each layer separated by a hyphen, the first layer being the text embedding layer. The dimensionality of the categorical feature embedding layer is fixed to 32 and omitted from the model name for simplicity.

Table 4.6: Evaluation metrics for models trained with different network depth and width

Layers	Recall@10	Recall@50	Recall@100	MRR@10	MRR@50	MRR@100
128-64	0.4060	0.6320	0.7160	0.2127	0.2236	0.2248
256-128	0.4183	0.6460	0.7277	**0.2194**	**0.2310**	**0.2320**
256-128-64	0.3862	0.6214	0.7035	0.1932	0.2046	0.2058
512-256	**0.4227**	**0.6544**	**0.7373**	0.2190	0.2304	0.2316
512-256-128	0.4127	0.6473	0.7296	0.2099	0.2212	0.2224

As shown in Table 4.6., decreasing the dimensionality of the text embedding layer and the hidden layer (128-64) reduces Recall@10 by 1.2 p.p. compared to the baseline model (256-128). Stacking another hidden layer (256-128-64) on top of the baseline model further degrades its performance (Recall@10) by 2.0 p.p. Interestingly, the model with two hidden layers (256-128-64) falls behind the model with one hidden layer (128-64) and a text embedding layer of half its size. This might point to the bigger model overfitting the training data.

Increasing the dimensionality of text embedding and hidden layers (512-256) improves the baseline (256-128) Recall@10 metric by 0.4 p.p. MRR metrics, however, are basically identical. This means that although the baseline model fails to find the document more frequently when it does find the documents, it ranks them higher than the larger model. Furthermore, the model with two hidden layers (512-256-128) once again performs worse than the model with just one hidden layer (512-256).

To summarize, adding a second hidden layer degrades models' performance. The baseline model (256-128) outperforms the smaller (128-64) model. It also achieves results comparable to the larger model (512-256) with a 35% shorter training time. Thus, the baseline model is kept for the subsequent experiments.

4.4 Tokenization

Word unigram tokenization - i.e., splitting the text on whitespaces is very simple and effective. However, as noted previously, the alternative approach - char n-gram tokenization is said to have an interesting property of being robust to typos / inflectional forms. The two techniques can also be combined. A summary of these approaches is presented in Table 4.7.

Table 4.7: Search query "silver fork" after tokenization

tokenization	tokens
word unigram	["silver", "fork"]
char trigram	["sil", "ilv", "lve", "ver","r f", " fo", "for", "ork"]
word unigram + char trigram	["silver", "fork", "sil", "ilv", "lve", "ver","r f", " fo", "for", "ork"]

In this experiment, two tokenizers producing token dictionaries with the same size are examined

- word unigram - 150k word tokens + 50k OOV tokens
- word unigram + char trigram - 110k word tokens + 40 char trigram tokens + 50k OOV

Apart from measuring the impact on evaluation metrics, it would be beneficial to measure the tokenizers' robustness to typos. However, to do that, one would have to have the dataset with misspelled queries which could be challenging and costly to obtain. To solve this issue, the algorithm for simulating typos using the dataset without typos is proposed.

4.4.1 Typo generation algorithm

For the sake of simplicity, it is assumed that queries can be misspelled in three ways:

- finger slip - selecting the character adjacent (on the keyboard) to the correct character,
- character transposition - switching order of two adjacent (in the word) characters,
- character removal - character is omitted from the word.

Typo types are showcased in Table 4.8.

Table 4.8: Different typos for the phrase "shower gel"

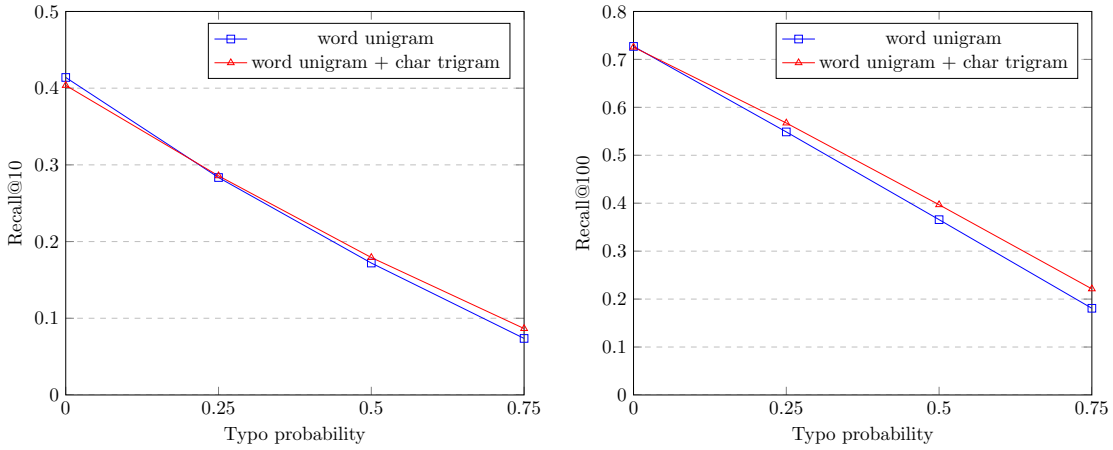
typo	result	explanation
finger slip	"shower hel"	character "h" used instead of character "g"
character transposition	"showre gel"	order of characters "e" and "r" switched
character removal	"showe gel"	character "r" removed

Algorithm for generating typos is presented in Algorithm 2. The finger slip algorithm is inspired by [19] and adapted for smartphone keyboards. The algorithm has one configurable parameter - p, the probability of a typo occurring in each token. Typo generation algorithm is applied to each query in evaluation queries dataset E_Q with $p \in \{0.25, 0.5, 0.75\}$, effectively producing three new evaluation datasets containing queries with a different number of typos.

Algorithm 2 Typo generator

```
procedure TYPO GENERATOR(query, p)
    for word in query do
        if len(word) < 2 then
            continue                    ▷ Words shorter than 2 characters are not modified
        end if
        r ← rand()
        if r > p then                   ▷ Decide with given probability whether the word will have a typo
            continue
        end if
        t ← rand()
        if t < 0.5 then                 ▷ Finger slip has a highest probability of 0.5
            fingerSlip(word)
        else if t < 0.75 then           ▷ Character removal has a probability of 0.25
            removeChar(word)
        else                            ▷ Character transposition has a probability of 0.25
            transposeChar(word)
        end if
    end for
end procedure
```

FIGURE 4.2: Recall for different tokenizers and typo probabilities



4.4.2 Results

Figure 4.1. shows the results of evaluating models with different tokenizers on datasets with different query typo probabilities. It is clear that the performance of both models degrades when the probability of typos increases. However, the decline of the model with the char trigram tokenizer is slightly less sharp. Regarding the Recall@10 metric, the char trigram tokenizer's performance is 1 p.p. worse for queries without typos and 1 p.p. better for queries with 75% chance of typo per word. Char tokenizer's advantage is more visible for the Recall@100 metric. While both tokenizers have basically identical performance for queries without typos, for queries with a 75% chance of typos, the char trigram tokenizer outperforms the word unigram tokenizer by 4 p.p.

While technically, the evaluation confirmed that adding a char trigram tokenizer improves robustness to misspelled queries, the gains are so small that they do not seem to justify a more complex tokenizer setup. Perhaps specialized training setup or different tokenization schemes are needed to observe noticeable typo robustness.

4.5 Comparison to classical retrieval

Having selected the best loss function, feature set, model architecture, and tokenization, the model can be compared to the classical information retrieval algorithm - BM25 [20]. To do that, the evaluation documents dataset is indexed in *Elasticsearch*. The index is then queried with queries from the evaluation queries dataset. Evaluation metrics are calculated on the result set.

4.5.1 Results

Model comparison to BM25 is presented in Table 4.9. As shown in the table, the model outperforms BM25 across all evaluation metrics, having Recall@10 10 p.p. higher and Recall@100 20 p.p. higher than the classical information retrieval algorithm.

Table 4.9: Evaluation metrics for models trained with features of different modalities

	Recall@10	Recall@50	Recall@100	MRR@10	MRR@50	MRR@100
BM25	0.3220	0.4646	0.5241	0.1948	0.2018	0.2026
Model	**0.4229**	**0.6454**	**0.7280**	**0.2234**	**0.2342**	**0.2354**

One reason why the model is superior to BM25 might be because of the training data. Since the model was trained on a clickthrough dataset from a commercial search engine, it could have learned the fuzzy matching and synonyms configured in the search engine. Elasticsearch with BM25, on the other hand, was constrained to raw lexical matching between queries and documents. It is nevertheless a good model quality indicator that the model was able to gather the knowledge from the rules configured by search and relevance engineers.

4.6 Model strengths and weaknesses

A manual study was conducted to find model strengths and weaknesses. Different query types were issued to assess the model performance. Key findings are summarized in this section.

4.6.1 Semantic matching

The empirical analysis shows that the model is, in fact, able to perform semantic matching between queries and documents. The model is able to highly rank the products with tokens synonymous with the ones used in the query. Furthermore, the model is able to perform matching between queries and documents with no overlapping tokens. Examples are gathered in Table 4.10., product names are simplified for readability.

Table 4.10: Semantic matching between queries and documents

query	matched product
aerobars	triathlon bike cockpit
photovoltaic panel	solar panel
vr goggles	vr glasses
wireless speaker	portable speaker
cardigan	loose sweater

4.6.2 Omittance of redundant tokens

Another strength of the model is that it is able to effectively ignore redundant tokens in the search query. A query like "iphone xr abc" would yield zero results in traditional search engines.

With embedding-based retrieval, relevant results for the query are returned. Table 4.11 presents examples of such cases.

Table 4.11: Omitting redundant tokens in search query

query	matched product
iphone xr abc	iphone xr 128 GB
tourist backpack zzzzz	tourist backpack 40 l

4.6.3 Sensitivity to misspelled queries

Despite the promise of being robust to misspelled queries, the model with the char trigram model is still susceptible to misspelled input. Mistyping one character in a query leads to an irrelevant set of products being retrieved. Issuing the query for "wadrobe" instead of "wardrobe" results in physics books being returned. Results for misspelled "bottle" query are mainly car parts. To make matters worse, in classical retrieval, results for mistyped queries would most likely be empty. In embedding-based retrieval, k results are always returned because there are always k documents closest to the query in the embedding space. To not show irrelevant results to the users, one would need to tune the similarity threshold above which documents are returned, e.g., returning only documents with cosine similarity greater than 0.5.

Chapter 5

Conclusions and future work

5.1 Conclusions

In this work, a general training algorithm for training two-tower models was proposed. Different loss functions were examined, pointing to conclusions that their hyperparameters are extremely important to tune. It was also observed that softmax cross entropy loss is an especially efficient loss function. Furthermore, it turned out that adding a feature of a different modality than text to the model is beneficial for evaluation metrics. To evaluate different tokenization techniques algorithm for generating mistyped queries was proposed. It was then discovered that while char trigram tokenization is more robust to mistyped queries, its absolute performance is disappointing. This leads to the conclusion that for production systems, spelling correction is better done before retrieval. Additionally, the best model was compared to the BM25 algorithm, exceeding its performance across all metrics, proving the good quality of the learned representation space. Lastly, the model's strengths and weaknesses were examined, with the key finding that the model is, in fact, able to perform semantic matching.

5.2 Further work

To further improve the model performance, following development directions can be considered.

- Inspecting other negative sampling techniques - in this work, batch negative sampling was used. It should be examined whether using random negatives or mixed negative sampling improves evaluation metrics.
- Evaluating personalization features - as shown in other works, adding user context to the query can be beneficial to model performance. Experiments with features such as gender or purchase history should be conducted.
- Extending the models with more multi-modal features. The ability to include features of different modalities goes beyond text and categorical features. Incorporating visual features, such as image embeddings, could lead to the model being able to match text to visual artifacts such as styles or textures. This could improve performance for queries concerning highly visual assortment - such as fashion or decor products.

Bibliography

[1] Hang Li and Jun Xu. Semantic matching in search. *Foundations and Trends in Information Retrieval*, 7(5), 2013. ISSN 15540677. doi: 10.1561/1500000035.

[2] Bhaskar Mitra and Nick Craswell. An introduction to neural information retrieval, 2018. ISSN 15540677.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 2013. ISSN 01628828. doi: 10.1109/TPAMI.2013.50.

[4] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao Ming Wu, and Qianli Ma. Embedding-based Product Retrieval in Taobao Search. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2021. doi: 10.1145/3447548.3467101.

[5] Po Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *International Conference on Information and Knowledge Management, Proceedings*, pages 2333–2338, 2013. ISBN 9781450322638. doi: 10.1145/2505515.2505665.

[6] Jui Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. Embedding-based Retrieval in Facebook Search. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2553–2561. Association for Computing Machinery, 8 2020. ISBN 9781450379984. doi: 10.1145/3394486.3403305.

[7] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *RecSys 2019 - 13th ACM Conference on Recommender Systems*, 2019. doi: 10.1145/3298689.3346996.

[8] Han Zhang, Songlin Wang, Kang Zhang, Zhiling Tang, Yunjiang Jiang, Yun Xiao, Weipeng Yan, and Wen Yun Yang. Towards Personalized and Semantic Retrieval: An End-to-End Solution for E-commerce Search via Embedding Learning. In *SIGIR 2020 - Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020. doi: 10.1145/3397271.3401446.

[9] Priyanka Nigam, Vihan Lakshman, Choon Hui Teo, Yiwei Song, Weitian Ding, Hao Gu, Vijai Mohan, Ankit Shingavi, and Bing Yin. Semantic product search. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. doi: 10.1145/3292500.3330759.

[10] Google. Folding, Recommendation Systems Course. URL `https://developers.google.com/machine-learning/recommendation/dnn/training`.

[11] Ji Yang, Xinyang Yi, Derek Zhiyuan Cheng, Lichan Hong, Yang Li, Simon Xiaoming Wang, Taibai Xu, and Ed H. Chi. Mixed Negative Sampling for Learning Two-tower Neural Networks in Recommendations. In *The Web Conference 2020 - Companion of the World Wide Web Conference, WWW 2020*, 2020. doi: 10.1145/3366424.3386195.

[12] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th International Conference On Machine Learning, ICML 2009*, 2009.

[13] Yiqun Liu, Kaushik Rangadurai, Yunzhong He, Siddarth Malreddy, Xunlong Gui, Xiaoyi Liu, and Fedor Borisyuk. Que2Search: Fast and Accurate Query and Document Understanding for Search at Facebook. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2021. doi: 10.1145/3447548.3467127.

[14] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[16] Jeff Johnson, Matthijs Douze, and Herve Jegou. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3), 2021. ISSN 23327790. doi: 10.1109/TBDATA.2019.2921572.

[17] PyTorch Contributors. TripletMarginWithDistanceLoss, PyTorch documentation. URL `https://pytorch.org/docs/stable/generated/torch.nn.TripletMarginWithDistanceLoss.html`.

[18] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys 2016 - Proceedings of the 10th ACM Conference on Recommender Systems*, 2016. doi: 10.1145/2959100.2959190.

[19] Dan Melton. Misspeling generator. URL `https://gist.github.com/danmelton/183313`.

[20] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4), 2009. ISSN 15540669. doi: 10.1561/1500000019.

Poznan University of Technology
Faculty of Computing and Telecommunication
Institute of Computing Science

Typeset using LaTeX.