University of Glasgow | School of Computing Science

A web search engine based on deep learning neural net models

Joseph Moukarzel

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

September 7th, 2018

Abstract

Search engines rank documents based on how the words in the query match those in the text. Existing models (e.g. BM25) may be combined with "stemming" to reduce any mismatch of words between query and document, thus increasing effectiveness. After that, other scoring models, such as proximity measures or PageRank, may be used to re-rank retrieved documents, to further increase effectiveness. They can be either employed as static score modifiers, or in a Learning To Rank (L2R) setting, allowing the retrieval engine to automatically learn how to combine these features in order to increase retrieval effectiveness. In this project, we use and implement new and existing deep neural networks to learn to match queries with documents and then use the learned models to re-rank retrieved documents. As proposed by previous work [4], the training data used is the output of an unsupervised ranking model, like BM25, obtained from running queries from the AOL against the Robust04 homogeneous collection. We study the effectiveness of each model by varying the size of the training set, labeling with and without query expansion and using stemmed and unstemmed document representation. Finally, we report on the effectiveness of each model as well as on the general idea of using neural network models in a weak supervision set to address the problem of semantic mismatch.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: **Joseph Moukarzel** Signature: [illegible]

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Dr. Craig MacDonald and Dr. Jeff Dalton, for giving me this opportunity to work on such an instructive and contemporary project. They were an essential guide to me throughout the course of this project. Under their guidance I was given the opportunity to extend my knowledge and broaden my horizons in the field of information retrieval and deep neural networks.

I would also like to thank Karl and Constantinos who agreed to proof-read this report and provide feedback on its comprehensibility.

Finally, I would like to thank all the school of computing science teachers, especially those of the data science curriculum who greatly contributed into making me who I am today, whether on the professional or ethical side.

Contents

Chapter 1

Introduction

1.1 Traditional Information Retrieval (IR)

Traditionally, when search engines rank documents in relation to a query, they match terms in the query to those in the document. For example, a simple retrieval model would simply count the occurrence of each query term in a document, and then sum up. The resulting number would be the score of each document. The engine then considers the highest scoring documents as being the most relevant to the query. When matching, a search engine might run into multiple forms of a word, such as “university”, and “universities”. These words are treated independently by the engine, potentially losing a match. For this reason, an engine can employ stemming on the query and document terms. Stemming reduces word variants to a single root. In this case, if we use the porter stemmer both words are reduced to “universiti”. By using stemming, two words that were originally treated as different by the search engine become equivalent [9], allowing the engine to retrieve potentially relevant documents. In addition to stemming, search engines can also employ query expansion, a technique that aims at reformulating the query based on some feedback strategy to enhance retrieval effectiveness. The three main expansion feedback strategies are the use of explicit feedback from a user, such as suggesting additional query terms to the initial query and allowing the user to choose which ones to add, implicit feedback like click tracking, and user history, and finally pseudo-relevance feedback, which does an initial retrieval based on the user query, and then uses the retrieved set to add terms to the initial query based on the most occurring terms in the top retrieved documents.

1.2 Machine learning and IR

More recently, learning to rank (L2R) [17], an approach that attempts to integrate machine learning in Information Retrieval has shown promising results. This approach consists of building a machine learning model, commonly a “forest” of decision trees, that learns to combine multiple document features in order to increase retrieval effectiveness. L2R is commonly used in a re-ranking setting, where some unsupervised model like BM25 or PL2 returns the top k documents for a query, and then the L2R model re-ranks these k documents according to the provided features. L2R models do not usually incorporate the textual content of a document, such as the document’s body, as a feature, but rather rely on meta-information like the document’s PageRank score, and the proximity of query terms in the document. Despite the promise of L2R in ad-hoc retrieval and the success of deep learning in speech recognition, Natural Language Processing (NLP), and image recognition, deep learning-based machine learning models have not been historically popular in retrieval. This has been partly due to the community considering that the retrieval task is fundamentally different from NLP, where deep learning has been successfully used, and partly due to the limited training data available for such a task.

1.3 Representational approaches in IR

Recently, many attempts at developing unsupervised models that try to learn query and document representations directly from the collection or from word embeddings, in order to match them, have had some success. These attempts are based on the Distributional Semantic Model (DSM), which is a model that learns how to associate words with fixed-dimensional vectors also known as embeddings. For example, a 4-dimensional representation of the word cat might be $\left[0.3, 0.1, 0.22, 0.01\right]$, while the words dog, and terrier could be represented as $\left[0.21, 0.12, 0.03, 0.4\right]$ and $\left[0.26, 0.08, 0.09, 0.43\right]$, respectively. Vectorial operations can be applied on them. For instance, $cos(\overrightarrow{cat}, \overrightarrow{dog}) = 0.47$, while $cos(\overrightarrow{dog}, \overrightarrow{terrier}) = 0.98$, thus capturing the idea that dog and terrier are closer semantically than dog and cat. These embeddings can be trained from a large collection using neural networks [21], or can be obtained from sources, such as the Stanford NLP initiative - Global Vectors for Word Representation (GloVe). However, these models work in a fully unsupervised setting, and without a training function that accurately models the problem, their effectiveness is limited.

This led to the creation of supervised models to train neural networks for the task of ad-hoc retrieval. However, the training data used was either small or biased [14]. For example, a research at Microsoft proposed the "Learning Deep Structured Semantic Models for Web Search using Clickthrough Data model" [14]. This model assumes that only, and all clicked documents are relevant. This does neither consider mis-clicks nor that some queries do not need documents to be clicked, as some of the advanced search engines can display answers to queries on the result page, without the users clicking on a document. Other models, such as the "Deep Relevance Matching Model for Ad-hoc Retrieval" [11], use human labeled data to train their networks. They acknowledge that the number of available queries with relevance judgements for such a task is limited. This problem is often mentioned and is attributed to the labeling task being time consuming and expensive [36, 33].

Given that these labels might not always be available, especially in large quantities, Deghani et al. [4] proposed the idea of weak supervision, i.e. to create automated training labels generated from an unsupervised ranking technique, and then train the models based on these labels. They showed that trained models could significantly outperform the model used to generate the labels if enough training examples are used.

1.4 Project purpose & outline

In this project, our main objective is to reproduce the results of Deghani's experiment and to try to integrate this idea into Terrier [19, 27, 25, 26] under the form of re-ranking. The following chapters are structured as follows: Chapter 2 deals with presenting a survey of work in the field that led to the idea of weak supervision, in order to formulate the requirements and objectives in a clear way. Chapter 3 describes the high level system architecture, it also provides implementation details of some parts of the system that are judged to be pertinent to the context. Chapter 4 specifies how the proposed system can answer the objectives of the project, and describes a series of experiments to do so. The results of the experiments are presented and analyzed in order to draw conclusions concerning the original problems. Finally, chapter 5 provides a conclusion for the project its implications and possible future work, as well as a reflection on what I learned.

Chapter 2

Background, Analysis & Requirements

2.1 Semantic matching

In traditional IR, documents are primarily retrieved by matching terms in the query with those of a document. For example, equation 2.1 shows how a well known retrieval model, BM25 [31], leverages this idea.

$$Score(Q, D) = \sum_{i=1}^{r} = \frac{f(D, q_i)(k_1 + 1)}{f(D, q_i) + k_1(1 - b + b\frac{|D|}{avg\_dl})} IDF(q_i) \tag{2.1}$$

$f(D, q_i)$ indicates the frequency of query term q_i in document D. b and k_1 are empirical parameters used to set the importance of the document size and frequency of apparition of term q_i. avg_dl is the average document size in the collection, and $|D|$ is the size of document D. $IDF(q_i)$ is the Inverse Document Frequency of query term q_i, which aims at reducing the score contribution of terms that appear a lot in the collection, because they are considered to be non-discriminative [18]. This model and many others, rank documents according to the exact lexical matches between the query and the document.

However, lexical matching is often not enough, even when combined with stemming, because the same meaning can be conveyed by using an entirely different vocabulary or a different linguistic style. For example, a traditional search engine receiving a query containing the acronym 'U.S.' might not return some relevant documents containing the terms 'USA', or 'United States' if they do not contain specifically the term 'U.S.'. The problem that is referred to here, is that of semantic matching. It has been an active area of research in NLP for the last two decades, and due to the success that it has had, IR researchers recently started considering semantic matching as a replacement or as a complement to traditional retrieval.

2.1.1 Early days of semantic matching in IR

Unsupervised semantic matching with pre-trained representations: The first attempt at semantic representation in IR came under the form of using pre-trained representations to learn an unsupervised semantic matching model. In this approach, each word in a query and in a document are represented using the embeddings of that word.
Having a vector that represents a word is a good start, but what is needed, is a way to represent queries and documents as vectors. A simple technique is to represent a document or a query by taking the maximum or the average of the representation of its composing words across each embedding dimension. While this technique might seem primitive, it has been extensively explored by many works [22, 41] and going beyond it, is not a trivial task. Attempts at doing so have not been scarce, however:

Dual Embedding Space Model (DESM) [24] moves from word embeddings to document embeddings by attempting to represent the document as a centroid of its OUT vectors, where an OUT vector is the result of the product of an IN (word vector) with the learned weights matrix. Cosine similarity is used between the OUT vectors of the document and the IN vectors of the query to produce the matching score [24]:

$$DESM_{IN-OUT}(q,d) = \frac{1}{q}\sum \frac{\vec{v}_{t_{q,IN}}^{T}\vec{v}_{t_{d,OUT}}}{||\vec{v}_{t_{q,IN}}^{T}||\ ||\vec{v}_{t_{d,OUT}}||} \tag{2.2}$$

Other similar models include the Neural Translation Language Model (NLTM) [43] which relies on term embeddings similarity to compute a term $->$ term transition probability sequence P(t$_q$/$_d$) to maximize the query likelihood given a document, and the Generalized Language Model (GLM) [10] where we model the process of query generation, i.e. find the model (document) that maximizes the likelihood of a query being generated by that document:

$$\begin{aligned}&maximize \quad P(D/Q) = \tfrac{P(Q/D)P(D)}{P(Q)} = \ Rank(P(D)P(Q/D))\,, \\ &where \quad P(Q/D)\ can\ be\ approximated\ by \\ &P(Q/D) = \textstyle\prod_{q_i \in D} P(qi/D)\end{aligned} \tag{2.3}$$

Learning unsupervised representations for semantic matching: In the previous approach, the main question was "How do we go from word embeddings to document embeddings in an information retrieval task?", in this approach, however, the main focus is to learn unsupervised query/document representations directly for IR tasks. Techniques such as Latent Semantic Analysis (LSA), Probabilistic Latent Semantic Analysis (PLSA), and Latent Dirichlet Allocation (LDA) have been used to address this issue with some success [3, 30, 6]. However these models work in a fully unsupervised setting and without a training function that accurately models the ranking problem, the results are not very accurate and barely comparable to standard BM25 and TF-IDF models [14]. From this finding came the idea of using deep neural networks in a supervised setting to train models for re-ranking, which was reinforced by Hinton et al [12] who showed that the semantic meaning found in documents and queries can be extracted through the use of deep neural networks in the form of auto-encoders. In accordance with this idea, Van Gysel et al. developed the Neural Vector Spaces [37], where document and query terms are represented directly from the document collection by using gradient descent. Their optimization objective is based on 3 ideas, (a) documents with a similar vocabulary and linguistic style end up close in the latent space, (b) high term frequency words are neglected, based on Luhn's hypothesis [18], (c) nearby words have close vectorial representations. This technique significantly outperforms other latent models by up to 40% [37]. Its major problem is that both its time and space complexities are largely affected by the number of documents in the collection.

With this approach's evolution, it has become clear that using neural networks to learn how to represent text and to address the semantic mismatch problem has had its success. There remains, however, the problems of finding the most appropriate network structure and training function and potentially defining the training data in the case of a supervised setting.

2.1.2 Learning to match models - a modern approach

While the previous approaches were concerned with "accurately" modeling queries and documents, here the main concern is relevance ranking and question answering. The main difference is that now the problem can be more easily formulated as a supervised objective by defining a document-query pair and associating a label with it (e.g. D1 Q1 label). Techniques that follow this concept can be divided into 2 categories, (a) Representation-based models, where the aim is to start by building a fixed-dimensional size vector representation for each document and the query separately and then perform query-document matching within the latent space that represents them. This means that the query and the document do not meet until their representation is mature. (b) Interaction-based models, where computing the interaction between query and document terms is usually done as a first step using a similarity matrix, and then transforming this interaction into a score between the document and the query.

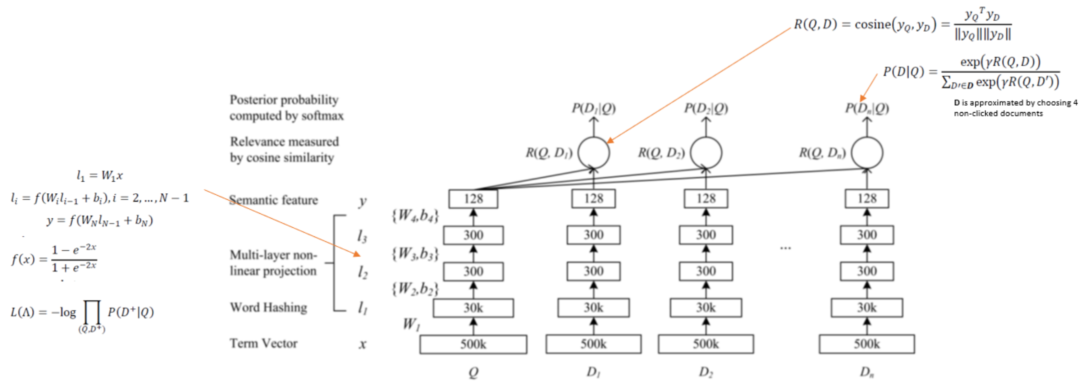
This approach is used in this project, and the next few pages are dedicated to exploring some of the proposed techniques and classifying them as (a) representation-based or (b) interaction-based. Many of the techniques explained below are used in the experiments of this project.

Representation-based models

DSSM & CDSSM: Learning Deep Structured Semantic Models for Web Search using Click-through Data [14] (CDSSM is its convolutional variant). Figure 2.1 shows how the model starts by representing a query and documents by applying word hashing [12] to their terms and then sending them through non-linear projection layers and finally ranking the documents using their cosine similarity with the query. The optimization problem is formulated as maximizing the document likelihood given a query P(D/Q). Originally developed to be used with click-through data, this a representative approach since queries and documents do not meet in the network until their representations are fully mature. Its main advantages are that the click through data is readily available and there is no need for human-made labels for training, and its main disadvantages are that it assumes that only and all clicked documents are relevant, and does not consider mis-clicks and that not all queries need documents to be clicked (e.g. weather, news, game score update, ...). It can be easily modified to use relevance judgements or non-binary score for training instead of click-through data as it tries to minimize the pairwise hinge loss function [32] shown in equation 2.4.

$$L(y;\theta) = \frac{1}{|l|}\sum_{i=1}^{|l|} max\{0, 1 + sign(score_{(q,d_2)_i} - score_{(q,d_1)_i})\} \tag{2.4}$$

Where l is the size of the batch per training iteration. The $score$ operator gives a document-query pair a relevance (the higher the more relevant a document is to a query). For this equation, it is assumed that document d_1 is more relevant than document d_2, hence the sign of the difference would be negative, and the maximum would be 0, reducing the loss value. This loss function allows the model to focus on ranking documents correctly instead of scoring them, as it only takes into account the score of documents in relation to each other.



$R(Q,D) = \text{cosine}(y_Q, y_D) = \frac{y_Q^T y_D}{\|y_Q\|\|y_D\|}$
$P(D|Q) = \frac{\exp(\gamma R(Q,D))}{\sum_{D' \in \mathbf{D}} \exp(\gamma R(Q,D'))}$
D is approximated by choosing 4 non-clicked documents
Posterior probability computed by softmax
Relevance measured by cosine similarity
$l_1 = W_1 x$
$l_i = f(W_i l_{i-1} + b_i), i = 2, \ldots, N-1$
$y = f(W_N l_{N-1} + b_N)$
$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$
$L(\Lambda) = -\log \prod_{(Q,D^+)} P(D^+|Q)$
Semantic feature
Multi-layer non-linear projection
Word Hashing
Term Vector
y
l_3
l_2
l_1
x
$\{W_4, b_4\}$
$\{W_3, b_3\}$
$\{W_2, b_2\}$
W_1
$P(D_1|Q)$
$P(D_2|Q)$
$P(D_n|Q)$
$R(Q, D_1)$
$R(Q, D_2)$
$R(Q, D_n)$
128
300
300
30k
500k
...
Q
D_1
D_2
D_n

Figure 2.1: Architecture of DSSM model, adapted from [14].

ARCI: Convolutional Neural Network Architectures for Matching Natural Language Sentences [13]. The main idea is to start with two matrices, one representing a query and one a document. Each row is the embedding of a word in that document/query. Figure 2.2 shows how the model successively applies convolutions and max-pooling kernels to each matrix to finally have a fixed size vector for each matrix that are then sent through a multi-layer perceptron for matching. The learning data is made up of human-labeled pairs of relevance judgements (e.g. D1 Q3 label), but non-binary labels can be used as well, since the optimization problem can be defined as minimizing the pairwise hinge loss, shown in 2.4. The convolutional layers try to capture semantic meaning by summarizing together nearby words, and the max-pooling tries to keep only the strongest signals in order to avoid overfitting. Its main advantage is that it is simple and the number of parameters to learn is not very high, thus reducing the number of training data required to achieve good results. The main problems of this technique are that it applies zero-padding to the query in order to increase its size to a fixed length, and it applies the same number of operations to a query, which is short, and to a document which is long, thus creating representational bias.

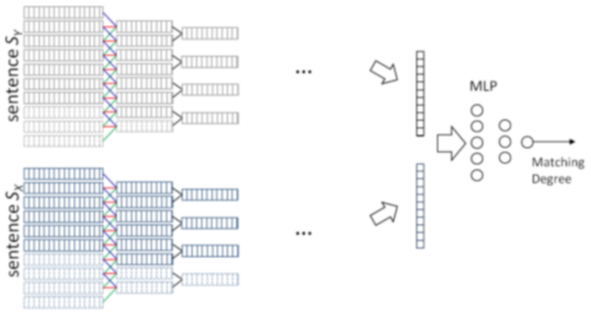


Figure 2.2: Architecture of ARCI model, taken from [13].

Interaction-based models

MatchPyramid (MP): Text Matching as Image Recognition [28]. Figure 2.3 shows how the model starts by representing the query and a document as 2 independent matrices where each row represents the embeddings of a word, then it applies a similarity operator between each element of the query matrix and each element of the document matrix to construct a similarity matrix. The work proposes the use of cosine because it is known in IR for its normalization factor that reduces the importance of document and query size on the score. Afterwards, a series of convolutions and max-pooling layers are successively applied on the joint query-document representation much in the same way as image recognition networks. Finally, the output of the last pooling layer is sent through a multi layer perceptron for matching.
While the idea of convolution and max-pooling to capture term interaction and to keep the strongest signals has been explored by representative models [13], the novelty here is that these operations are not independently applied on the query and on the document, but rather, on their similarity matrix. This model performs well in practice, and significantly outperforms the representative DSSM, and ARCI approaches on paraphrase identification, and paper citation matching. The model in itself

was not was not developed for ad-hoc retrieval, but the ideas it introduces can be used by models specifically developed for retrieval and ranking.

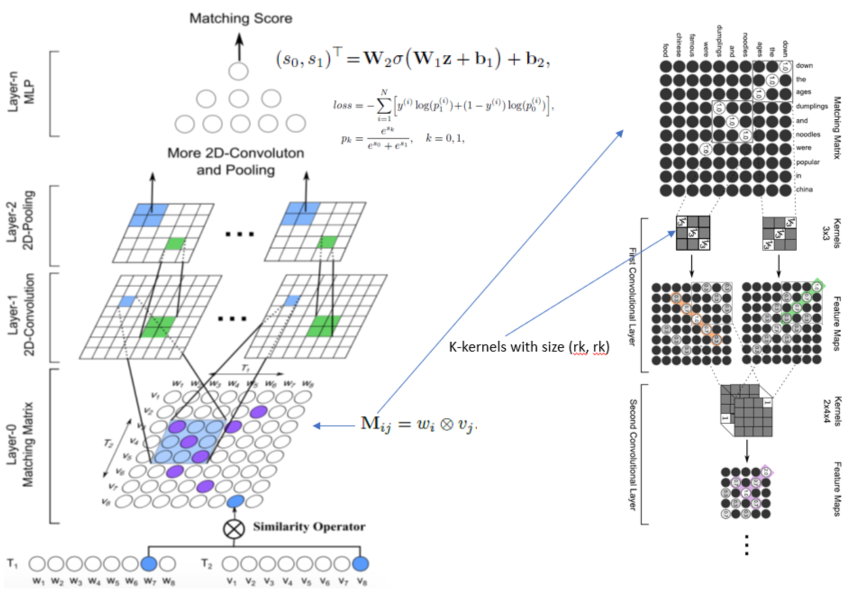
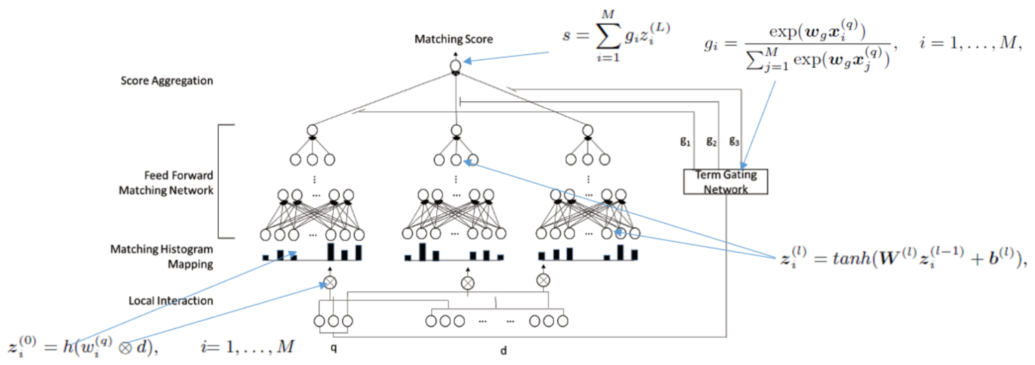

Matching Score
$(s_0, s_1)^{\top} = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1) + \mathbf{b}_2,$
Layer-n MLP
More 2D-Convoluton and Pooling
Layer-2 2D-Pooling
Layer-1 2D-Convolution
Layer-0 Matching Matrix
$\mathbf{M}_{ij} = w_i \otimes v_j.$
Similarity Operator
K-kernels with size (rk, rk)
Matching Matrix
Kernels 3x3
First Convolutional Layer
Feature Maps
Kernels 2x4x4
Second Convolutional Layer
Feature Maps

Figure 2.3: Architecture of MP, adapted from [28].

DRMM: A Deep Relevance Matching Model for Ad-hoc Retrieval [11]. As with other interaction-based models, at first, the model computes the cosine similarity between each term in the query and each term in the document. These values are fed into a matching mapping histogram, because the length of queries/documents is not fixed. This replaces the idea of a similarity matrix since the authors argue that the notion of relevance matching is fundamentally different than the semantic matching found in NLP tasks, as such, a similarity matrix that computes the similarity between each word in the document and each word in the query term would introduce positional bias and noise, thus making it unsuitable for information retrieval [11]. In addition, the similarity matrices used in other interaction-based models favour long queries and documents over shorter ones since they introduce zero-padding to make them into a fixed length vector. Figure 2.4 shows how the matching histograms group local interactions according to different levels of signal strengths : [(1,0.5), (0.5,0), (0, 0.5), (0.5, 1), (1, 1)]. The outputs of the histograms are fed into simple feed forward matching network to learn the matching weights. In parallel to that, a term gating network is fed the query terms and learns a matrix of importance for each term based on each term's TF or IDF scores, and the final score is a sum of the output of the feed forward matching network and the term gating network. Like previous models, the objective function tries to minimize the pairwise hinge loss. The main advantage of this architecture is that it deals with the problem of representational bias, and its main drawback is that it categorically considers the position of terms as being irrelevant, even though it has been shown to be a useful measure, especially in re-ranking [2].

DRMM_TKS: A variant of DRMM that uses kernel trees to capture non-linearity [42]. Aims at improving DRMM, by replacing the matching histograms with a non-linear operation: max-pooling to keep the top-k strongest signals. In more detail, the document and the query are represented as sequences of embedding vectors, represented as $d = [\vec{w}_1^d, \vec{w}_2^d, ..., \vec{w}_n^d]$ and $q = [\vec{w}_1^q, \vec{w}_2^q, ..., \vec{w}_m^q]$. As opposed to its predecessor, the model does construct an (n x m) similarity matrix M for the query-document pair, by computing $d(q)^T$. The values of this matrix are used as input to the dense



Matching Score
$s = \sum_{i=1}^{M} g_i z_i^{(L)}$
$g_i = \frac{\exp(\boldsymbol{w}_g \boldsymbol{x}_i^{(q)})}{\sum_{j=1}^{M} \exp(\boldsymbol{w}_g \boldsymbol{x}_j^{(q)})}, \quad i = 1, \ldots, M,$
Score Aggregation
Feed Forward Matching Network
g1 g2 g3
Term Gating Network
Matching Histogram Mapping
$\boldsymbol{z}_i^{(l)} = tanh(\boldsymbol{W}^{(l)} \boldsymbol{z}_i^{(l-1)} + \boldsymbol{b}^{(l)}),$
Local Interaction
$\boldsymbol{z}_i^{(0)} = h(w_i^{(q)} \otimes d), \quad i = 1, \ldots, M$
q
d

Figure 2.4: Architecture of DRMM, adapted from [11].

layer of the network. Then based on a softmax operation, the top-k best signals are selected and fed into the following layers. This operation is useful for short queries and documents that are zero-padded to a fixed fixed length vector, i.e. it insures that the padded zeros don't propagate to the score computation. The output layer computes the final matching score between the query and the document.

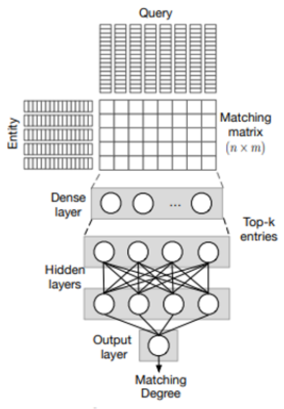


Figure 2.5: Architecture of DRMM-TKS, adapted from [42].

KNRM: End-to-End Neural Ad-hoc Ranking with Kernel Pooling [39]. The model starts by building two matrices, one for the query and one for the document where each row is a word embedding. After that a similarity matrix is constructed by computing cosine similarity between each query term and each doc term, i.e. each row in the resulting matrix represents a term's resemblance with each word in the document. The novelty of this model is that it applies K kernels to each row, so each kernel's size is (1xembedding size). Each kernel outputs a number. As such for each row the result is a K-dimensional vector, i.e. one element from each kernel. The sum of the elements in each vector represents the soft term frequency for the corresponding query term in that document. After summation, the soft counts are placed in a new vector, which is fed to a single layer perceptron that finally outputs the score for that document. For example if the query term contains the word "dog" and the document contains only the word "terrier", and assuming the similarity between "dog" and "terrier" is 0.7, then soft count would output that "dog" appears 0.7 times in the document, allowing to capture the semantic meaning that "terrier" is highly related to "dog". The training objective is defined as minimizing the pairwise hinge loss, just like the previous models.

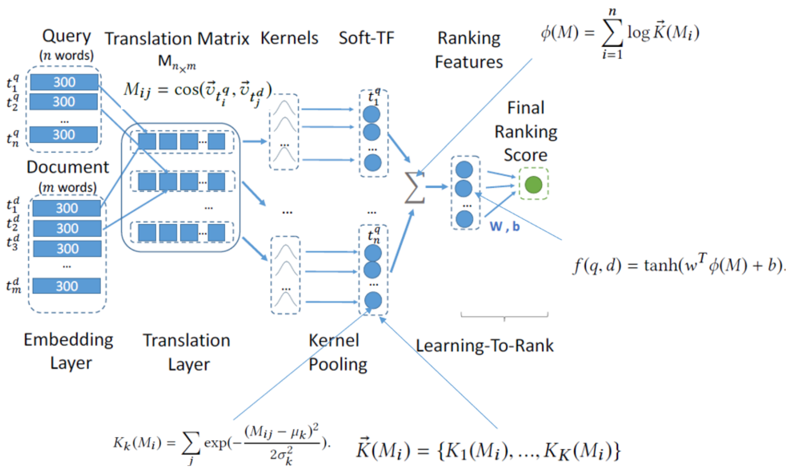

Query
(n words)
Translation Matrix
$M_{n \times m}$
Kernels
Soft-TF
Ranking Features
$\phi(M) = \sum_{i=1}^{n} \log \vec{K}(M_i)$
$M_{ij} = \cos(\vec{v}_{t_i^q}, \vec{v}_{t_j^d})$
t_1^q
t_2^q
t_n^q
300
Document
(m words)
t_1^d
t_2^d
t_3^d
t_m^d
Final Ranking Score
w, b
$f(q,d) = \tanh(w^T \phi(M) + b)$
Embedding Layer
Translation Layer
Kernel Pooling
Learning-To-Rank
$K_k(M_i) = \sum_j \exp(-\frac{(M_{ij} - \mu_k)^2}{2\sigma_k^2})$
$\vec{K}(M_i) = \{K_1(M_i), \ldots, K_K(M_i)\}$

Figure 2.6: Architecture of KNRM, adapted from [39].

DUET: Learning to Match Using Local and Distributed Representations of Text for Web Search [23]. Figure 2.7 shows how the duet model regroups both representation and interaction based models: the local model aim to capture exact matches, by sending the query and the document through a interaction featurizer, a convolution layer follow by a series of fully connected layers, and the distributed model first learns low-dimension representation of the documents and queries by independently applying word hashing [12] to the query and document, and are then sent through multiple layers of convolutions, dense, and pooling to finally produce a similarity matrix based on the Hadamard product that is then sent through a series of fully connected layers. Its main advantage is that it tries to bring the best from each kind of representation. The principle disadvantage is that the number of parameters to learn can easily reach hundreds of millions, thus requiring a lot of training data.

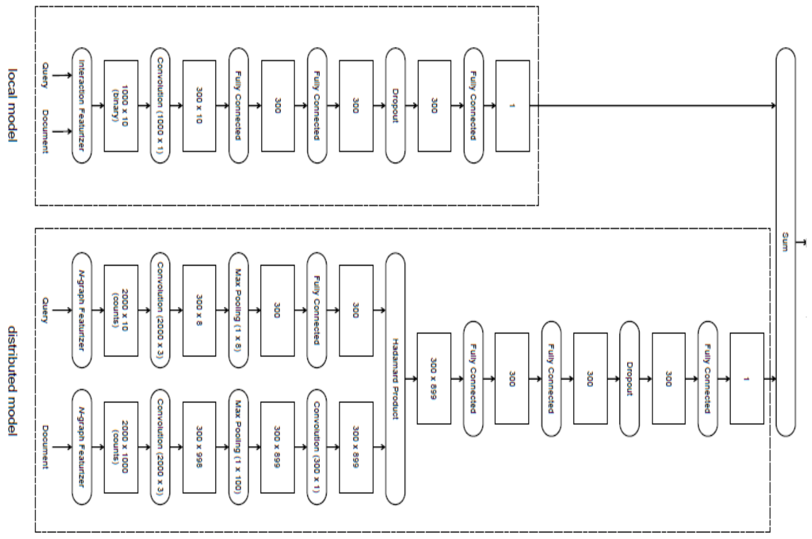


Figure 2.7: Architecture of DUET, adapted from [23].

More models have been proposed in the literature and studied during the course of this project, but will only be cited to avoid clutter:

PACRR: A Position-Aware Neural IR Model for Relevance Matching [15]. Applies kernels of different size to the similarity matrix to simulate n-gram matches and incorporates IDF to take into account term specificity.

ARCII: A variant of ARCI where a similarity matrix is computed before the convolutions are applied [13], making it an interaction-based model.

ANMM: Ranking Short Answer Texts with Attention-Based Neural Matching Model [40]. Based on building a similarity matrix between the query terms and the document terms and then applying a value-shared weighting scheme network that captures similar regularities in both texts followed by dense layers to learn the matching.

2.1.3 Weak supervision

The explored models rely on human-made relevance judgements for training and testing. The labeling task, however, is time-consuming and expensive [36, 33]. Some works [38, 1] proposed the use of Crowdsourcing to accomplish this task, and they showed that it can be a feasible alternative to professional assessors, if the system is sourcing the right crowd and implements quality control measures such as honeypots. These constraints alongside the steps of the creation of a Crowdsourcing pipeline, (a) planning the content for each task and the number of tasks, (b) defining task configuration and result management, (c) identifying the best platform for the task and the audience, (d) defining an aggregation strategy, can be burdensome, making Crowdsourcing less practical.
To remediate to the problem of missing supervised signals, Deghani et al. propose the "Neural Ranking Model With Weak Supervision" [4] that uses labels generated from an unsupervised model like BM25 to create the training data. They showed that by increasing the number of training instances up to a few million and by finding optimal parameter settings for their network, it is possible to achieve effectiveness metrics that go beyond the ones achieved by the unsupervised model. This is mostly attributed to the model being able to capture semantic matches that go beyond standard lexical matching through the use of non-linear operations in the network. The main advantages in such a technique are that, it's free, it's fast, and the number of labels that can be obtained is only bound by the size of the collection itself.
The network that they use is quite simple and only consists of a few dense layers. What is of interest to this project is the ability to learn from weak signals, specifically BM25. A criticism for training with BM25 would be that the trained model would be biased towards BM25. However, the labels don't necessarily have to be obtained from BM25. They can be generated from any unsupervised model or a combination of multiple models, making this approach a framework for generating training data. For this project, however, we only use BM25 for label generation to establish a working prototype and a proof of concept.

2.2 Requirements Gathering

The principal aim of this project is to design a system that allows us to implement and evaluate neural re-ranking and to integrate them in Terrier so they can be used during retrieval. One of the main focuses of the project is the reproduction of the results achieved by Deghani et al [4], using their proposed model and the following models described earlier in this section: CDSSM, ARCI, ARCII, DRMM, KNRM, aNMM, DUET, MP and DRMM_TKS. A secondary aim of this project is to compare the different neural models implemented to decide which one is the most suitable for re-ranking with weak supervision.

The following paragraphs include a summary of the functional and non-functional requirements expressed in terms of their MoSCoW weighting.

2.2.1 Functional requirements

- *R1-MUST:* The system must allow the user to specify a trained neural network model to be used during re-ranking via a simple command or in a configuration file.
- *R2-MUST:* The system must allow the user to design and train their own neural-network, so it can be used during re-ranking.
  - *R2.1:* The system must allow the user to provide it with queries and documents in a unique and specific format for training.
  - *R2.2:* The system must allow the user to specify the training labels as binary or float (for weak supervision).
  - *R2.3:* All the commonly used functionalities, layers, and operations (convolutions, pooling, averaging, mapping, etc...) in designing and developing neural networks must be accessible to the user.
- *R3-SHOULD:* The system could allow the user to specify the importance of a neural model during re-ranking in the form of a coefficient.
- *R4-COULD:* The system would be able to allow the user to use a trained neural network in a learning to rank (L2R) setting.

2.2.2 Non-functional requirements

- *R5-MUST:* The system must be able to handle millions of queries and documents during training and thousands during re-ranking.
- *R6-SHOULD:* The system must be able to quickly re-rank the provided documents (order of milliseconds for a thousand of documents). Keeping the retrieval time to a minimum is always a concern in information retrieval.
- *R7-SHOULD:* The system should be portable to Windows and Linux.
- *R8-SHOULD:* Given that these models transform text into vectors and matrices of floats for processing, the system should be able to run them on a graphics card. In fact, GPUs are specifically designed to efficiently handle large numerical arrays and can lead to speed ups of several orders of magnitude [34].
- *R9-COULD:* The system could be able to read and write multiple file formats, including compressed files due to the large size that corpora can reach (Hundreds of GBs).

In the next section, we propose a system that addresses these requirements. After that, we use the system to run retrieval experiments and we report on the effectiveness of each model in the system.

Chapter 3

Design & Implementation

In order to fulfill the requirements of the project, a suitable framework for developing and using the aforementioned deep learning models (and others) is needed. We decide to use MatchZoo (MZ) [8], a keras-based with tensorflow backend framework for implementing text matching neural networks. MZ is a good choice for this project because it provides implementations for many models: DSSM, CDSSM, ARCI, ARCII, MV-LSTM, DRMM, KNRM, aNMM, DUET, MatchPyramid and DRMM_TKS, some of which have been discussed in the previous chapter It also allows the addition of new models easily. The usage of tensorflow and Keras addresses requirements R8, and partially R6, by allowing the models to run on GPU, increasing the speed by orders of magnitude. Also, the usage of Terrier and MZ allows the system to be portable to many platforms including Windows and Linux, addressing requirement R7.

Since Terrier is in Java, and MZ is in python, a pipeline to ensure the interaction between the two systems is needed. The pipeline should be able to transfer data from the Terrier index to MZ to train the models. On retrieval, it should also allow Terrier to use one of these trained models to re-rank documents. The following section is dedicated to explaining the architecture of the system.

3.1 System architecture

In order to describe the architecture of the pipeline between Terrier and MZ, we first provide a description of the structure of MZ, its training and prediction processes.

3.1.1 Structure of MZ

MZ offers an abstraction over the models whether for training or using the models. This abstraction allows users to add new models in the form of a pair of files (one containing the code and the other containing the configuration of the neural network), and run them without modifying any code in the framework. This abstraction allows us to directly address requirement R2.3. However, in order to achieve it, MZ requires some specific input files.

Input of MZ

`corpus_preprocessed`**:** Contains the mapping of each document and query to a vector of integers where the first integer indicates the size of the document/query and where each successive integer indicates the ID of a word present in the document. The word IDs are indicated in the order that they occur in the document to provide a more accurate representation.
`word_dict`**:** Contains a mapping of each word to a unique ID that are needed in `corpus_preprocessed` and `word_stats`.
`word_stats`**:** Indicates the total number of times a word occurs in the collection, the number of documents it appears in, and its Inverted Document Frequency (IDF) score.

DocID or QueryID	Size	WordIDs
Q1	1	87837
D1	7	78 1 43 5464 8 21 78
D2	254	778 97 2352 245 ...
Q2	3	1 9 87
D3	254	23 353 2351 1 ...
D4	168	24 24 32 23342 ...
Q3	5	34 2455 8475 3442 864

Table 3.1: Table showing the format of `corpus_preprocessed`.

Word	WordID
rules	0
monday	1
regulation	2

Table 3.2: Table showing the format of `word_dict`.

WordID	Freq	DocFreq	IDF
0	55935	30222	2.245
1	47881	33482	2.400
2	68034	29509	2.049

Table 3.3: Table showing the format of `word_stats`

Score	QueryID	DocID
48.068	Q1725	D225779
47.652	Q1725	D239504
16.801	Q1728	D172302

Table 3.4: Table showing the format of `relation_train`

`relation_train`: Essentially, this file contains the training data. Each line contains a Document-Query pair to which is associated a score. For example, the score could be obtained using an unsupervised algorithms such as BM25.
`relation_valid`: Same format as the previous file, however, this one one is used to tune the hyperparameters of the model, as is normally done with validation data.
`relation_test`: Same format as the previous file. This file contains the Document-Query pairs which we would like to re-rank based on the learned models.
`Embedding file`: Specifies the mapping of each WordID to a low dimensional vector (e.g. 50, 300). This file can be normalized by removing the mean of each dimension.

WordID	Representation
0	-0.126155 -0.121950 -0.044587 -0.055263 ...
1	0.207413 0.136026 0.200087 0.239905 ...
2	0.139437 -0.069490 -0.229085 -0.025190 ...

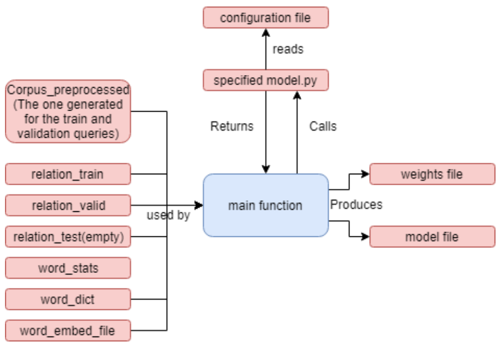
Table 3.5: Table showing the format of the embed file.

This standard and specific formatting of files allows us to directly address requirement R2.1.

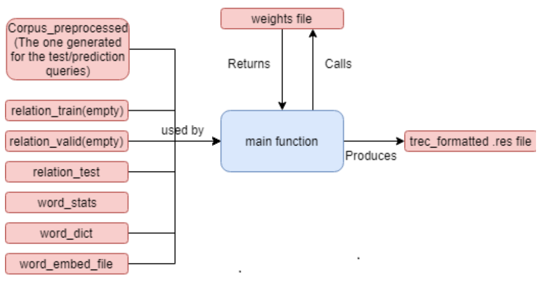
Output of MZ:

Figure 3.1 shows a high level view of how MZ uses the files described in this section. It also shows how for each trained model, MZ outputs a weights file, containing the weights of the layers in the model. This file is used when doing predictions. After doing a prediction (alternatively called test) run, MZ outputs a .res file in TREC format with the new scores for each Document-Query pair. A

TREC result file is defined line-by-line with each line containing the following information quadruplet $\{$Topic, Iteration, Document, Relevancy$\}$, where in our case, a topic is a query, relevancy is the score for each query-document pair, and iteration is the feedback iteration and it's not used. The resulting .res file can be evaluated in Terrier by using the appropriate command.



(a) Training phase diagram.



(b) Testing/Predicting phase diagram.

Figure 3.1: High level overview of MZ's training and predicting flow diagram.

Integration/usage of the trained models in Terrier:

The output weights file of a trained model in MZ can be used to do predictions as specified in Figure 3.1b, however, it is not practical for a user to manually copy/paste data files from Terrier's directory to MZ's directory for predictions, and then from MZ to Terrier for evaluation. As such, a way to automate the prediction process without requiring intervention from the user is necessary. The following provides attempts at doing such a task.

Keras to DL4J: Deep Learning For Java (DL4J) is a Java Library for training and using deep neural networks on the JVM. It also provides functionalities to import models that were constructed in Keras and saved to a weights file. The class that does the work is **KerasModelImport**. The *importKerasSequentialModel* function allows to import models that are constructed via the Keras sequential API, as **MultiLayerNetwork** objects, and the *importKerasModel* function allows to import models that are constructed via the Keras functional API, as **ComputationGraph** objects. In our case, the models are built using the functional API, so the aim is to import them as **ComputationGraph** objects. As specified earlier in this chapter, after a model is done training in MZ, its weights file is saved to disk. However, as it turns out, the most recent versions of DL4J (1.0.0-beta, 1.0.0-alpha and the more stable 0.9.1) only allow imports from Keras 1.x, and since we are using Keras 2.0.2, the imports do not work. A downgrade to Keras 1.x, would remove some of functionalities that are needed to construct the networks, such as the Dot and Concat layers (classes), and others. There are open issues for this on github [20, 35, 7, 29], but most solutions propose downgrading from keras 2.x to keras 1.x, which is not something feasible here.

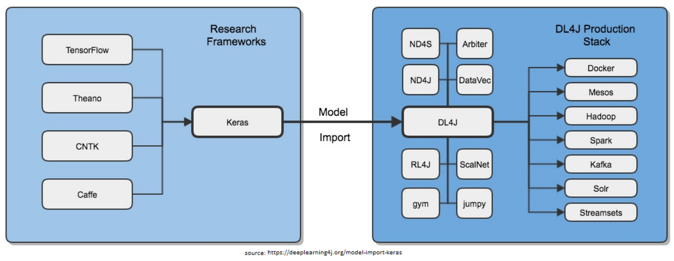


Figure 3.2: DL4J model import architecture [5].

Integration with inter-process communication (IPC): Since a direct integration is not possible at the moment, the alternative is to create a python server, within MZ, which loads a specified trained model in memory and awaits messages from a Terrier retrieval run. When it receives the messages, it sends them through the specified network, recuperates the output, and sends that output to the retrieval instance.

3.2 Implementation

The implementation of the pipeline is divided into two modules, (a) the first provides a list of functionalities allowing Terrier to provide MZ with the list of files it needs for training, and (b) the second provides a server in MZ which loads a model and waits for inputs from Terrier for re-ranking. In this section, we explain how these modules are implemented.

3.2.1 Input files generation

In order to provide the needed files to MZ, Terrier first needs to index the collection of documents using the standard functionalities offered. While the indexing can be done with or without stemming, it must record the position of terms in documents.

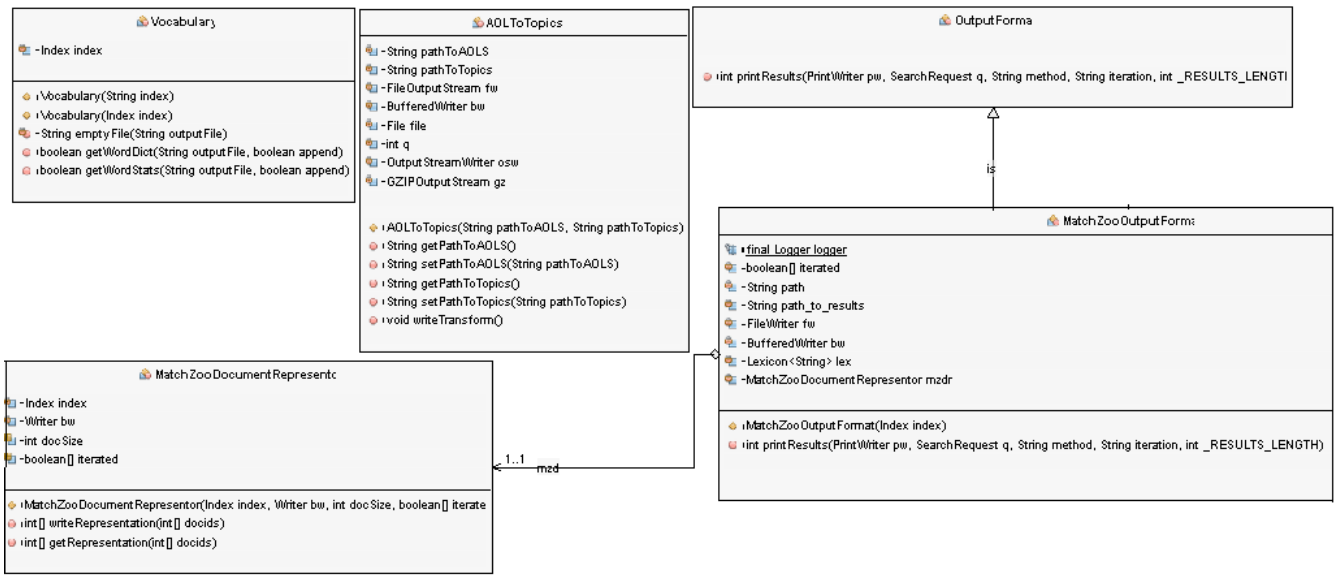

Vocabulary
-Index index
Vocabulary(String index)
Vocabulary(Index index)
-String emptyFile(String outputFile)
boolean getWordDict(String outputFile, boolean append)
boolean getWordStats(String outputFile, boolean append)
AOLToTopics
-String pathToAOLS
-String pathToTopics
-FileOutputStream fw
-BufferedWriter bw
-File file
-int q
-OutputStreamWriter osw
-GZIPOutputStream gz
AOLToTopics(String pathToAOLS, String pathToTopics)
String getPathToAOLS()
String setPathToAOLS(String pathToAOLS)
String getPathToTopics()
String setPathToTopics(String pathToTopics)
void writeTransform()
OutputForma
int printResults(PrintWriter pw, SearchRequest q, String method, String iteration, int _RESULTS_LENGTI
is
MatchZooOutputForm
final Logger logger
-boolean[] iterated
-String path
-String path_to_results
-FileWriter fw
-BufferedWriter bw
-Lexicon<String> lex
-MatchZooDocumentRepresentor mzdr
MatchZooOutputFormat(Index index)
int printResults(PrintWriter pw, SearchRequest q, String method, String iteration, int _RESULTS_LENGTH)
MatchZooDocumentRepresentc
-Index index
-Writer bw
-int docSize
-boolean[] iterated
MatchZooDocumentRepresentor(Index index, Writer bw, int docSize, boolean[] iterate
int[] writeRepresentation(int[] docids)
int[] getRepresentation(int[] docids)
1..1
mzd

Figure 3.3: Architecture allowing the provision of MZ with the input files.

Figure 3.3 provides a list of classes and functionalities that generate the required input for MZ to train a model. The following is dedicated to explaining how these classes are used during the input files generation.

AOLToTopics: This class allows the transformation of line-by-line queries from the AOL file to queries in a topical format to be used by Terrier. A topical format identifies queries by a number and attaches to each query some potential metadata. Usually a topic is defined with the quadruplet {Number, Title, Desc, Narr}, where Number is a unique query identifier for this list of queries, Title is the content of the query (e.g. University of Glasgow famous alumni), Desc is an optional tag describing the query briefly (e.g. Who are some famous individuals that graduated from the university of Glasgow), and Narrative is also an optional tag describing the query further by specifying what relevant documents would look like (e.g. A relevant document will contain references to, or biographies of notable graduates from the University. For example, relevant documents could contain a list of people who have been awarded notable rewards like the Nobel prize). The last two tags are used by human annotators to label documents in a collection as being relevant or not to the query.
`word_stats` & `word_dict`: The class **Vocabulary** provides functionalities that iterate through the Terrier lexicon and outputs these 2 files. In more detail, after having specified the index or the path to the index to the constructor, one can invoke the following functions to get these files:

- *getWordDict*(String outputFile, boolean append). This function writes the word dictionary to a destination specified by outputFile. The user can also specify whether they want to append or override the existing one.
- *getWordStats*(String outputFile, boolean append). This function writes the word statistics to a destination specified by outputFile. The user can also specify whether they want to append or override the existing one.

The `corpus_preprocessed`, `relation_train` and `relation_valid` are generated together. The class **MatchZooOutputFormat** implements the Terrier **OutputFormat** class to produce them. For each query that is fed into it, the representation for that query and for each document in its result set is written to `corpus_preprocessed`, without repetition. This is achieved by using the class **MatchZooDocumentRepresentor** to which is specified the index, a writer handle, the maximum document representation size, and an array of booleans that specifies if a document was already written to the corpus file and that is updated each time a document is written to the corpus file.

- The function *writeRepresentation*(int[] docids) in **MatchZooDocumentRepresentor** does the actual boolean verification and writing to the file, given a specified list of document IDs or query IDs.

The output destination is in TERRIER_HOME\var\res. Terrier allows to write these files as .GZip files, reducing their size on disk considerably. To make MZ compatible with .GZip files, we modified the way it processes files by testing the extension of its input files, and reading them accordingly. This change fulfilled requirement R9. Also, MZ was only able to handle binary scores/relevance judgements, i.e. the relation files it expected were assumed to have the following format: {1/0 QueryID DocumentID}. However, since one of the aims of the project is the usage of weak supervision with neural networks, we had to modify MZ's pairwise loss computation to handle arbitrary score values, allowing us to address requirement R2.1.
In addition to the Java classes and functions we provide, MZ has a script that generates the word embeddings file from the `word_dict` file. The embed file, referred to in MZ as `embed_glove_dxx`,

where xx refers to an embedding size (e.g. 50 or 300) is generated by calling on the gen_w2v.py script provided by MZ. gen_w2v.py is fed the pre-trained word embeddings such as the gloVe embeddings (glove.840B.300d) and the word_dict file. The output is a file that contains the embedding of each word.

3.2.2 IPC implementation

As shown in figure 3.4, we provide the **NeuralScoreModifier** (NSM) class which extends the Terrier interface **DocumentScoreModifier**. This gives the system access to the retrieved result set of each query, and allows the change of the score of each document. Figure 3.5 shows how we use this class to re-rank documents by changing their scores. For each query for which we retrieve the relevant documents, the **NeuralScoreModifier** instance contacts a python server that has the model in memory, and consecutively sends the query, the retrieved result set and their representation in terms of WordIDs. We modified MZ to be able to read the required data straight from memory instead from the files described in the previous section. The server sends each query-document pair in the neural network and returns a new score to the NSM which then modifies the score of each query-document pair. The new score can replace the old score, or can be combined with the old score by using a coefficient specified in the Terrier properties file under the name neural.modifier.alpha. Equation 3.1 shows how the score is updated for each query-document pair, allowing us to address requirement R3.

$$\begin{aligned} score[Document_i, Query_j] \leftarrow \alpha * newScore[Document_i, Query_j] \\ +(1-\alpha) * score[Document_i, Query_j] \end{aligned} \tag{3.1}$$

At first we had developed the process communication as Datagram packets (UDP), however, seeing as the server might not necessarily be started on the same machine, the messages might be lost or may not arrive in order, causing the run to fail. For that reason, we consider to send the messages in packets via TCP/IP or to use REST APIs. REST APIs have the main advantage of being easily extensible. One can easily add more attributes to be sent without breaking backward compatibility. They can also be consumed by different applications and not necessarily by this specific server. On the other hand, a TCP implementation has the advantage of being faster, but loses the extensibility and compatibility provided by a REST API. In our case, we are mostly concerned with retrieval speed and do not foresee a change in the required attributes that we send over the network. For this reason, the implementation of the IPC is done via TCP/IP.

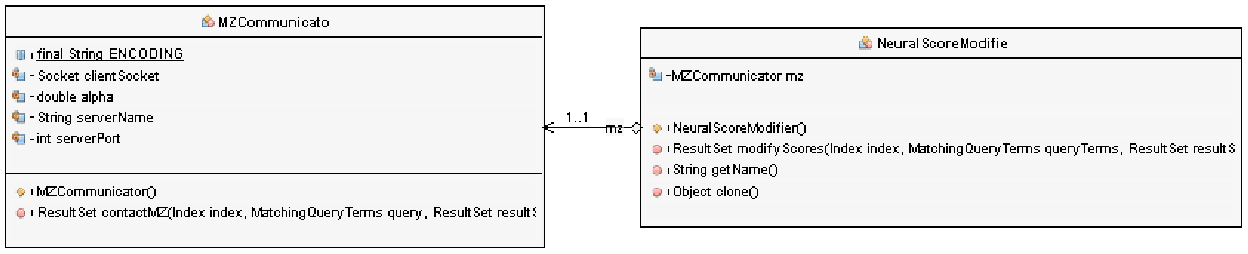


Figure 3.4: Architecture allowing the integration of a neural model into Terrier.

3.2.3 Example of a neural model implementation

To make ideas more concrete, the following code snippet and its explanation show how we implement the Neural Ranking Model With Weak Supervision (NMWS), as a combination of a model file and a configuration file in MZ. We choose to show this specific model's implementation because it's the first model that was designed with the intent of using weak supervision. The NMWS model consists of a series of dense (fully connected layers) with dropout to prevent overfitting.

`nmws.config` - configuration file

```
Specification of the output weights file:
     weights_file: examples/Robust/weights/nmws.Robust.444.50.
        false.254.weights
Specification of the embedding and vocabulary size:
     embed_size: 50,
     vocab_size: 520424,
Specification of the maximum query and document length:
     text1_maxlen : 10,
     text2_maxlen: 254,
Specification of the training configurations:
     learning_rate: 0.001,
     batch_per_iter: 5,
     query_per_iter: 50,
     batch_size: 100,
     relation_file: ./data/Robust/relation_train.txt,
     model_path: ./matchzoo/models/,
     model_py: nmws.NMWS,
     layer_count: [3],
     layer_size: 64
     dropout_rate: 0.3,
     train_embed: false
```

`nmws.py` - model file

```
#Get the maximum query and document lengths
query = Input(name=query, shape=(self.config[text1_maxlen],))
doc = Input(name=doc, shape=(self.config[text2_maxlen],))
#Retrieve the embedding
embedding = Embedding(self.config[vocab_size], self.config[embed_size],
    weights=[self.config[embed]], trainable = self.embed_trainable)
#Get query and document embedding representations
q_embed = embedding(query)
d_embed = embedding(doc)
#Concatenate the query and document
psi = Concatenate(axis=1) ([q_embed, d_embed])
#Add dropout and dense layers
for layer_count in self.config[layer_count]:
    psi = Dropout(rate=self.config[dropout_rate])(psi)
    psi = Dense(self.config[layer_size], activation =sigmoid)(psi)
#Final layer consists of a single neuron to obtain a single score for each document
out_ = Dense(1)(psi)
model = Model(inputs=[query, doc], outputs=[out_])
return model
```

Depending on the parameters provided for the variable 'embedding', the input of the network should be equivalent to equation 3.2 proposed by Deghani et al. [4]:

$$\begin{aligned}
&\psi(d,q) = [\odot_{i=1}^{|q|}(E(t_i^q), W(t_i^q))||\odot_{i=1}^{|d|}(E(t_i^d), W(t_i^d))] \\
&with, \\
&\odot_{i=1}^{|n|}(E(t_i^n), W(t_i^n)) = \textstyle\sum_{i=n}^{n} \hat{W}(t_i)E(t_i) \\
&\hat{W}(t_i) = \frac{e^{W(t_i)}}{\sum_{j=1}^{n} e^{W(t_j)}}
\end{aligned} \tag{3.2}$$

Where E denotes the embedding function, $\hat{W}$ denotes the weight or the importance of each term in the vocabulary. If set to true in the configuration file, then the model will attempt to learn term importance and use it during predictions, if set to false, then the weight of each term is one. $\odot$ creates a single weighted vector for each term and each query based on the embeddings of their terms, and on the weight of each term. For example, an n-term query or document would be at first represented by an (n, embed size) matrix. $\odot$ would map it to an (1, embed size) vector. Finally, ψ concatenates the query with a document and the output is sent into the network. In addition, the model is also able to modify the provided embeddings by setting the configuration parameter "embed_trainable" to true.

3.3 Usage of the system for re-ranking

The user starts the server with a model specified on the command line by specifying the configuration file of the model as follows:

```
python matchzoo/Server.py - -model_file examples/Robust/config/drmm_tks_Robust.config
```

Then, the user starts a Terrier retrieval run using the following command:

```
trec_terrier.bat -r -q -Dtrec.model=BM25 -Dtrec.topics=Path/To/Topics
```

With the Terrier configuration file containing the following properties:

Property	Description	Default value
matching.dsms	Fully qualified DocumentScoreModifier class name	No re-ranking model
neural.server.name	Host name where the python server is started	localhost
neural.server.port	Host port where the python server is started	6776
neural.modifier.alpha	Coefficient of the Re-ranking model	1

Table 3.6: Configuration properties required for neural re-ranking with Terrier and MZ.

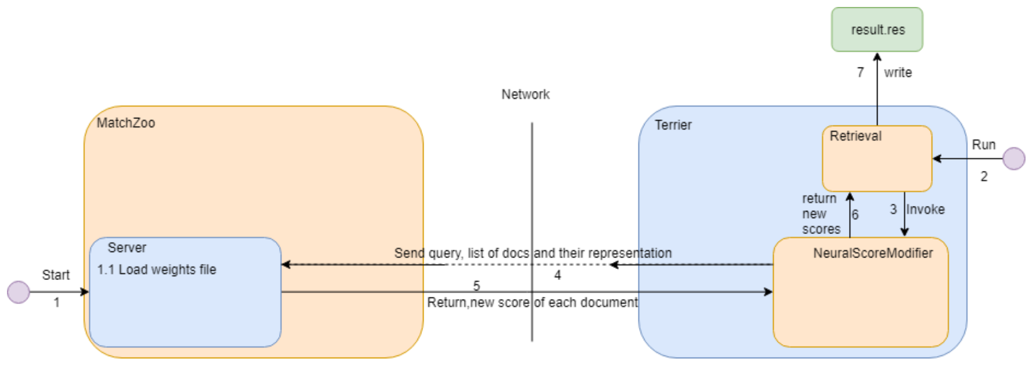


Figure 3.5: Terrier-MatchZoo retrieval IPC.

This simple interface allows a user to specify a re-ranking model with its coefficient, allowing us to address requirement R1.

3.3.1 Usage of the system in L2R

A more general case where re-ranking can be used is L2R, where the system automatically learns how important a feature is and learns to combine multiple features. Terrier provides a full interface for L2R. The difference with the previously mentioned re-ranking is that not all queries with relevance judgments are used for testing. In fact, after having trained a model in MZ using weak supervision and obtained its weights file, the user divides the labeled queries into training/validation/testing. The training and validation data are then used by Terrier as features to train and tune the parameters of a model (usually a random forest). The user then does a retrieval run on the test set and specifies the learned model as a document re-ranker. In a sense, this simulates the automatic tuning of the alpha parameter.

3.4 Summary

In this section, we provide an overview as well as implementation details of the architecture that (a) allows the provision of MZ with the required data for training, and of (b) the architecture that allows Terrier to communicate with MZ during retrieval in order to re-rank documents. These two modules constitute our integration system/pipeline: The training data provided to MZ is obtained by iterating through the indexed collection in Terrier and outputting the required files. During retrieval, the re-ranking is achieved by sending the query, and the documents to a python server that loads a neural model in memory. The server sends the received information in the network and outputs a score which is sent back to Terrier to either replace the old document score with the new one, or combine both scores.

In the following chapter, we propose research questions that we would like to address, then we use this system to run retrieval experiments that treat the research questions.

Chapter 4

Evaluation & Experiments

In the next section, first, we propose the research questions we would like to address, then we provide an experimental setup to address each one of these questions, and finally we show the results of each experiment and discuss these results in relation to the corresponding research question.

4.1 Experimental questions

The idea of using weak supervision is to replace human-made relevance judgements in case they are absent or limited, therefore, we propose to study RQ1 to evaluate the 'quality' of labels generated from an unsupervised model, BM25. We hypothesize that labels generated from weak supervision can train models to be as effective as models trained on human-made labels. We also hypothesize that the usage of stemming and/or query expansion to generate the labels has a significant effect on the quality of the trained models.

- RQ1: Do models trained with human-made relevance judgments significantly outperform the same models trained using labels obtained from weak supervision, specifically BM25?
  - RQ1.1: Does the provision of training labels from BM25 with query expansion yield significantly better models than models provisioned with BM25 labels without query expansion?
  - RQ1.2: Does the provision of training labels from BM25 with document stemming yield significantly better models than models provisioned with BM25 without document stemming?

All of the mentioned models except NMWS were originally designed to work in a full-ranking setting rather than in a re-ranking setting. In full-ranking, a model does not only rank the top k retrieved documents for each query, but ranks every document in the collection for each query. Since our aim is to use them in re-ranking, we propose to study RQ2 to compare how these models would fare in both full-ranking and re-ranking scenarios.

- RQ2: For which neural model does re-ranking outperform full ranking? Is one approach markedly better than the other?

To have a significant impact on retrieval effectiveness, models trained with weak supervision should be able to outperform the models used to generate the labels. One of the main hypotheses of this project is that this is possible. We propose to study RQ3 in order to answer to this question.

- RQ3: Can the proposed neural models provide a statistically significant increase to the MAP in re-ranking over BM25, if provided with training data labeled from BM25? Is that possible in under 100k training and validation queries?

We hypothesize that the neural models address the semantic mismatch problem. That is, neural models are able to understand that two words are semantically related without being explicitly provided with such information. In order to test this hypothesis we propose to study RQ4. By analyzing the queries that are most affected by the re-ranking procedure, we might be able to find semantic patterns that networks are sensitive to.

- RQ4: Which queries benefit the most from the weak supervision re-ranking approach? Which ones are degraded the most? Are their any remarkable characteristics?

4.2 Experimental design

Since one of the purposes of the project is to reproduce the results of Deghani et al. [4], we use the same collection of documents, training and evaluation queries. The document collection we use is the Robust04 homogeneous track which consists of 500k news articles. We also use part of the 36 million queries provided from the AOL (without relevance judgements) and 250 queries with relevance judgements. The embeddings files used are the Stanford GloVe 6B tokens, 400K vocab, 50d and the Stanford GloVe 840B tokens, 2.2M vocab, 300d. Some of the terms that appear in the queries (e.g. 'ahswbfbshsbv') are bound to be inexistent in the embeddings file, and are considered as Out Of Vocabulary (OOV). The system simply ignores these terms. This certainly creates bias in the trained models, affecting the effectiveness of retrieval. However, this is not of concern to us since the removed terms (e.g. 'ahswbfbshsbv') aren't likely to be used in practice in a useful way.

From the models discussed in chapter 2, and those offered by MZ, the following are the ones used in most of the of the experiments of this project: CDSSM, ARCI, ARCII, MVLSTM, DRMM, KNRM, ANMM, DUET, DRMM_TKS and NMWS (which we implemented in MZ).

An experiment is designed for each research question. The parameters used for BM25 are k_1= 1.2, and b = 0.75. These are the default values provided by Terrier, and they are observed to work well in practice. For any of the following experiments, the configuration options in table 4.1 are used unless indicated otherwise. Also, in the following experiments, the notion of 'betterness' is defined as a significant improvement in the Mean Average Precision (MAP) between two models.

Model	Document size*	Embedding size	Update embedding?	Loss
CDSSM	254	50	No	Rank (pairwise hinge loss)
ARCI	254	50	No	Rank (pairwise hinge loss)
ARCII	254	50	No	Rank (pairwise hinge loss)
MVLSTM	254	50	No	Rank (pairwise hinge loss)
DRMM	254	300	No	Rank (pairwise hinge loss)
KNRM	254	50	No	Rank (pairwise hinge loss)
ANMM	254	300	No	Rank (pairwise hinge loss)
DUET	254	50	No	Rank (pairwise hinge loss)
DRMM_TKS	254	300	No	Rank (pairwise hinge loss)
NMWS	254	50	No	Rank (pairwise hinge loss)

Table 4.1: Configuration settings of the different neural models.

*Document size indicates the first n terms of the document that are used.

4.2.1 Experiment 1: Full-ranking with relevance judgements vs full-ranking with BM25 generated labels

The setup

Full-ranking with relevance judgements:

- Divide the 250 queries with relevance judgement into training, validation, testing in a 65-17.5-17.5 fashion.
- Train the model using the training data and tune the hyparameters based on the validation data: dropout ratio, number of hidden layers, number of kernels, size of kernels, size of hidden layers (where applicable).
- Predict the relevant documents of each test query and evaluate the result set against those in the qrels.

Full-ranking with BM25 generated labels:

- Divide the 250 queries into training, validation, testing in a 65-17.5-17.5 fashion.
- Do a retrieval run in Terrier on the training and validation queries to return the relevant set and the score of each document.
  - In order to provide answers for question RQ1.1 and RQ1.2, multiple end-to-end runs, training and evaluation are made with different options: with/without query expansion and with/without document stemming.
- Train the model using the training data and tune the hyparameters based on the validation data: dropout ratio, number of hidden layers, number of kernels, size of kernels, size of hidden layers (where applicable).
- Predict the relevant documents of each test query and evaluate the result set against those in the qrels.

The aim: This experiment would tell us how well these models perform when trained on a small size data. It will also tell us how best to generate the labels for weak supervision (query expansion vs no query expansion, stemming vs no stemming). In addition, the comparison of the two full-ranking approaches will tell us how better or worse the weak supervision labels are when compared to relevance judgements.

4.2.2 Experiment 2: Full-ranking vs re-ranking

The setup

- Divide the 250 queries into training, validation, testing in a 65-17.5-17.5 fashion.
- Do a retrieval run in Terrier on the training and validation queries to return the relevant set and the score of each document.
- Train the model using the training data and tune the hyparameters based on the validation data: dropout ratio, number of hidden layers, number of kernels, size of kernels, size of hidden layers (where applicable).

- Use BM25 to retrieve the first 1000 relevant document of each query in the test set using Terrier.
- Re-rank these documents using the learned models. The alpha parameter is set to 1, meaning that the new rank of each document from the top 1000 retrieved set is entirely decided by the neural model.
- Evaluate using the qrels file for the test queries.
- Compare the effectiveness metrics to those obtained by full-ranking (same results as experiment 1)

The aim: This experiment allows us to discuss if re-ranking a retrieved set is better than full-ranking.

4.2.3 Experiment 3: Re-ranking with BM25 generated labels vs BM25 ranking

The setup

- Retrieve the relevant documents according to BM25 with stemming and with query expansion for a different number of queries from the AOL: 444, 2000, 10 000, and 100 000 queries
- Train each model with 444, 2000, 10 000, and 100 000 queries. So each model has 4 weights files, each with different settings.
- For some value of alpha, re-rank the top 1000 retrieved documents for each of the 250 queries with relevance judgements.
- Evaluate using the qrels file for the 250 queries, using each of the 4 weights files.
- Repeat re-ranking with multiple values of alpha and plot the results.

The aim: This experiment would allow us to discuss if re-ranking a result set retrieved by BM25 with a trained model provides an enhancement to effectiveness, as well as how the size of the training set affects this enhancement. This is the realistic scenario for which we originally intended the usage of trained models. This experiment would also allow us to discuss how important the neural re-ranking is by varying the model's importance via the alpha parameter.

4.2.4 Experiment 4: Qualitative analysis - queries benefited, hurt and their characteristics

The setup

- Generate the query by query performance from Terrier for the BM25 with query expansion and stemming result set and for the BM25 with query expansion and stemming + neural re-ranking for the best model from experiment 3.
- Retrieve the list of queries that are either improved or hurt by a minimum relative precision of 25% and ± 0.1 precision difference.
- Report on patterns for the hurt/benefited queries.

The aim: Establish the qualitative effect of re-ranking in addressing the semantic mismatch problems.

4.3 Results

Some of the terms are abbreviated to reduce the size of the tables:
-BM25L = BM25-generated labels.
-QE = query expansion.
-STE = document stemming.

4.3.1 Experiment 1: Full-ranking with relevance judgements vs full-ranking with BM25 generated labels

Model	Human-made labels	BM25L, no QE, no STE	BM25L, QE, no STE	BM25L, QE, STE
CDSSM	0.0904	0.0798	0.0835	0.0927
ARCI	0.0816	0.0773	0.0860	0.0773
ARCII	0.0841	0.0804	0.0860	0.0792
MVLSTM	0.0951	0.0739	0.0854	0.0859
DRMM	0.1739	0.1263	0.0876	0.0826
KNRM	0.0923	0.0839	0.0846	0.0823
ANMM	0.2162	0.1278	0.1502	0.1863
DUET	0.1002	0.1115	0.1274	0.1362
DRMM_TKS	0.1634	0.1435	0.1672	0.1750

Table 4.2: This table reports the MAP of multiple neural-net based ranking algorithms trained on queries with labels obtained differently. For the first results column, the labels are obtained from human-made judgements, for the second, the labels are obtained from BM25 without query expansion and without document stemming. For the third, the labels are also obtained from BM25 but with query expansion and without stemming. For the last column the labels are obtained from BM25 with query expansion and with document stemming.

A t-test analysis on the MAP values of each model trained with BM25 labels without query expansion and without stemming against the same models trained with human-made labels shows that the result is significant at $p < .05$. DUET aside, the results are better with relevance judgement training than with using the BM25 labels WITHOUT query expansion from the UNSTEMMED index to train/validate in a full-ranking setting.

A t-test analysis on the MAP values of each model trained with BM25 labels with query expansion and without stemming against the same models trained with human-made labels shows that the result is NOT significant at $p < .05$. Aside from DRMM and ANMM, the results are not better with relevance judgement training than with using the BM25 labels WITH query expansion from the UNSTEMMED index to train/validate in a full-ranking setting.

A t-test analysis on the MAP values of each model trained with BM25 labels with query expansion and with stemming against the same models trained with human-made labels shows that the result is NOT significant at $p < .05$. Aside from DRMM, the results are not better with relevance judgement training than with using the BM25 labels WITH query expansion from the STEMMED index to train/validate in a full-ranking setting.

Discussion

The usage of labels generated from BM25 can provide the same training quality as labels obtained from human assessors (relevance judgements). This is especially marked when using query expansion and document stemming. We attribute this improvement to the fact that all of these architectures, except DRMM(TKS), zero-pad queries to a fixed size length of 10 words, thus causing great representational bias which is harmful to small queries. By using query expansion, small queries are padded to a length of 10 words with useful words, giving the model access to more information. On the other hand, stemming does provide an impact, but it is not statistically significant enough, meaning that neural networks, in our case DRMM(TKS), KNRM, and DUET are able to capture morphological variations of a word without the help of stemming.

To conclude, this experiment shows that training with relevance judgements is not better than training with weak supervision even for a small number of queries, if the training labels are provided with query expansion. While this conclusion is reached with a full-ranking setting, we assume that it is also true for re-ranking in future experiments. For this reason, in future experiments, the labels are obtained from BM25 in combination with query expansion and document stemming.

4.3.2 Experiment 2: Full-ranking vs re-ranking

Model	Full-ranking. BM25L, QE, STE	Re-ranking. BM25L, QE, STE
CDSSM	0.0927	0.0391
ARCI	0.0773	0.0315
ARCII	0.0792	0.0294
MVLSTM	0.0859	0.0268
DRMM	0.0826	0.0689
KNRM	0.0823	0.0759
ANMM	0.1863	0.0499
DUET	0.1362	0.0474
DRMM_TKS	0.1750	0.1099
NMWS	N/A	0.0380

Table 4.3: Table showing the MAP of the same models trained with the same data. When the models are used in full-ranking, they are significantly better than when they are used for re-ranking.

Discussion

By setting alpha=1, equation 3.1 can be summarized as follows: $score[Document_i, Query_j] \leftarrow newScore[Document_i, Query_j]$, with newScore being the value attributed to document i from one of the neural models. With this value, all the neural models tested are significantly better when used for full-ranking. The results of this experiment do not give us information as to why the models are performing much worse in re-ranking. It can be that the training size is not large enough, or that the alpha parameter should be smaller than 1. That is why, in the next experiment, we explore the importance of these variables alongside comparing re-ranking against BM25 on its own.

4.3.3 Experiment 3: Re-ranking with BM25 generated labels vs BM25 ranking

While in the previous experiments the training and validation data were limited in size (250 queries for training, validation and testing), here we do the training with, 444, 2000, 10 000, and 100 000 queries from the AOL and test on the 250 queries with relevance judgements. However, given the time, and resource constraints, as well as the unpromising results of most models, this experiment is restrained only to DRMM_TKS which according to table 4.3, shows more promise than others.

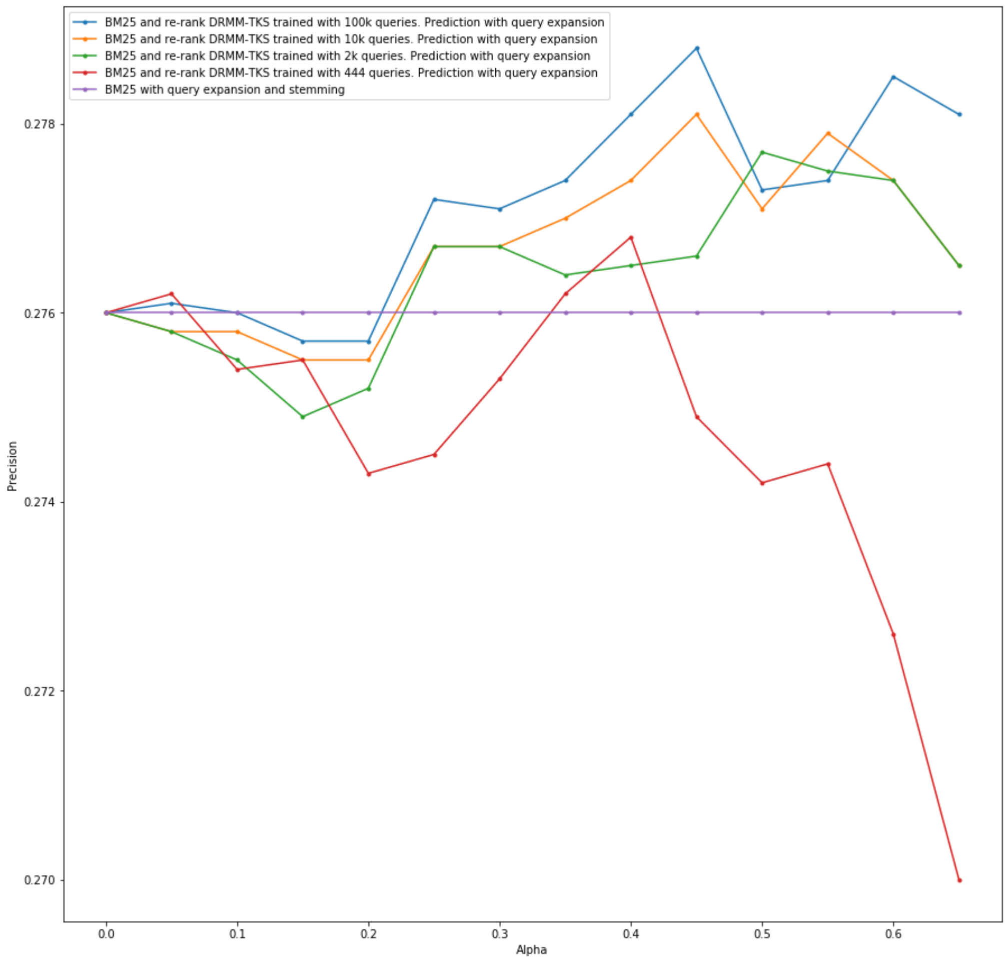


Figure 4.1: Precision of BM25 alone, and of BM25 with re-ranking with DRMM_TKs model trained with different number of queries from the AOL. Multiple measurements are done for different alpha coefficients. The plotting stops at 0.65 because for higher values, the precision significantly drops to very low values, rendering the oscillations between 0.2 and 0.5 unnoticeable.

Discussion

Figure 4.1 shows that training with larger data markedly improves effectiveness, and does not reach saturation with the increase of the training size. The leaps of effectiveness observed at alpha=0.45

compared to BM25 (flat-line) show promise, however the improvement is not statistically significant at $p < 0.05$. The precision significantly drops to low values starting from alpha=0.5, reaching values similar to the one in table 4.3, meaning that in its current state, the network cannot be used on its own to re-rank the results of every query effectively. We attribute this to the difference in query types found in the AOL, used for training, and the 250 queries used for testing. In fact, most of the queries in the AOL are either navigational, or pornographic (more than 50% of the queries). These themes are significantly different from the 250 queries using for testing, which are mostly made of news queries. The issue is also amplified by the fact that the maximum training data we can use is 100k queries. This is minimal compared to the 6.5 million queries used by Deghani et al, when they proved a statistical significance to the results. The reason why the experiment is limited to just 100k queries is MZ's inability to handle more than that even after being deployed on a Google Cloud instance with over 50 GB of RAM and a Tesla K80 GPU. We made many attempts to fix the memory issues among which a successful memory leak detection reported on GitHub [16]. However, the fixes were not enough to go beyond 100k queries.

4.3.4 Experiment 4: Qualitative analysis - queries benefited, hurt and their characteristics

The best performing model in the previous experiment is BM25 + DRMM_TKs trained with 100k queries and alpha=0.45. It has a precision of 27.9% compared to BM25 with query expansion and stemming, which has 27.6%. Table 4.4 shows the queries that are markedly improved or harmed by using the BM25 + DRMM_TKs trained with 100k queries and alpha=0.45 compared to simple BM25 with query expansion.

Benefited	Harmed
Risk of Aspirin	Mexican Air Pollution
Airport Security	declining birth rates
killer bee attacks	
lyme disease	
Turkey Iraq water	
timber exports Asia	
U.S. invasion of Panama	
term limits	

Table 4.4: Queries markedly improved or harmed by using the BM25 + DRMM_TKs trained with 100k queries and alpha=0.45 compared to simple BM25.

Discussion

The marked improvement of query 'U.S. invasion of Panama' suggests that the model was able to capture the semantic meaning carried by 'U.S.'. In fact, by looking at the retrieved set before and after re-ranking, we notice that some documents containing 'united states', 'USA', and other variants were swapped to higher ranks. For the rest of the improved queries, it is noticeable that their terms have higher frequencies in the training set than those that are harmed. For example, in the 100k training data, 'Mexican' appears only 10 times, but 'Airport' and 'Security' occur more than 50 times each. This suggests that the network was able to learn better representations for these terms during training. We attribute the lack of significant improvement using these models to both the 'small' training data size and to the discrepancies in themes in the training and test collections.

Chapter 5

Conclusion

5.1 Requirements satisfaction & achievements

The system proposed in Chapter 3 addresses the requirements stated in Chapter 2. In particular, the system provides Terrier with the ability to use neural re-ranking via a simple command line/ configuration file interface, where the user can specify the model that they wish to use, as well as the weight of that model. We were also able to successfully integrate weak supervision in MZ to train models from weak labels instead of binary relevance judgements. The system also allows users to implement, train and deploy their own neural models in re-ranking, full-ranking and L2R. At the time of writing this report, the system is usable on Windows as well as on Ubuntu, and is not just a prototype with academic ends. As far as I know, this constitutes the first deployable open-source attempt at integrating neural networks in re-ranking and L2R in a major retrieval platform like Terrier.

During the experimental process, we were able to partially address the semantic mismatch problem, by showing that some of the proposed models perform comparably with and without query expansion and stemming. By analyzing how DRMM_TKs re-ranked queries, we were able to see that the model did in some cases capture semantic relations (e.g. U.S. and united states). Also, by separately training the same neural models with binary labels and weak supervision, and evaluating them on the same test set, we were able to show that weak supervision is a viable alternative to human-made binary relevance assessments.

5.2 Challenges

One of the main challenges faced during the course of the project is the inability of MZ to handle large data sets for training. The main issue is MZ's assumption that all the data would be in memory when training. This assumption went unnoticed when running experiments 1 and 2, which only use 250 queries. However, it turned out to be problematic when the data set was increased. For example, training with 100k queries requires about 150 GB of memory, this number would grow to approximately 9000 GB had we used 6 million queries. This is usually addressed by streaming the data from disk and training with batches using Stochastic Gradient Descent (SGD). While MZ does use SGD to perform the training, the system still could not read from disk. We attempted to modify the code to perform disk streaming, but all attempts were met with failures from other modules within MZ. MZ also suffered from a memory leak issue, which we reported [16] on GitHub and was accepted by the developers.

Other minor issues faced include the generation of the input files from Terrier to train models in MZ. In fact, the indexing in Terrier has to provide positional information of the terms in each document since this information is needed when generating the corpus file.

5.3 Future work

Following up on the challenges, it would be interesting and surely rewarding to reproduce experiments 3 and 4 with larger data, to more properly address the problem of semantic mismatch. I propose to train DRMM_TKs with 6 million queries and test on the same 250 queries. To address the problem of large data, I suggest to add a new model to MZ which is dedicated to streaming data from the disk, or alternatively to implement a new, simple framework, designed only for this task. The 6 million queries should be sampled in such a way to have a more balanced thematic distribution of queries. This would ensure that the models would not be biased towards a few themes that depend on the training set. With this training size that is 60 times larger than the largest data set used in this project, we hope that the results would either validate the idea that neural re-ranking with supervision can improve effectiveness or would invalidate it by showing that the effectiveness is saturating at some statistically insignificant threshold.

5.4 Reflection

I have worked on developing software before, but this was my first real contact with a research project. Throughout the course of this project, I learned how to choose research questions and how to better formulate and test hypotheses. I can say that this was an interesting experience for me and that it has peaked my interest in doing research. On the technical side, I was already familiar with neural networks and retrieval after having taken the corresponding courses during my curriculum. However, the project allowed me to apply the theoretical knowledge I had acquired to a useful and practical project. I was able to understand and address the complexities of building a useful neural network and the delicacy of evaluating effectiveness in Information Retrieval.

Bibliography

[1] Omar Alonso. *Can we get rid of TREC assessors? Using Mechanical Turk for relevance assessment*. 2009.

[2] Ronan Cummins and Colm O'Riordan. Learning in a pairwise term-term proximity framework for information retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 251–258. ACM, 2009.

[3] George Dahl, Alex Acero, Dong Yu, and Li Deng. *Context-dependent, pre-trained deep neural networks for large vocabulary speech recognition*. IEEE Transactions on Audio, Speech, and Language Processing, 2012.

[4] Mostafa Deghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and Bruce Croft. *Neural Ranking Models wih Weak Supervision*. SIGIR17, Shinjuku, Tokyo, Japan, 2017.

[5] dl4j. Keras import overview. `https://deeplearning4j.org/docs/latest/keras-import-overview`.

[6] Susan Dumais, George Furnas, Thomas Landauer, Richard Harshman, and Scott Deerwester. *Automatic cross-linguistic information retrieval using latent semantic indexing*. In AAAI-97 Spring Sympo-sium Series: Cross-Language Text and Speech Retrieval, 1997.

[7] Eduramiba. Unsupported keras layer type UpsampleLike. `https://github.com/deeplearning4j/deeplearning4j/issues/5028`, 2017.

[8] Yixing Fan, Liang Pang, JianPeng Hou, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. *MatchZoo: A Toolkit for Deep Text Matching*. arXiv preprint arXiv:1707.07270, 2017.

[9] William Frakes and Ricardo Baeza-Yates. *Information retrieval: Data structures & algorithms*. Prentice hall, 1992.

[10] Debasis Ganguly, Dwaipayan Roy, Mandar Mitra, and Gareth JF Jones. *Word embedding based generalized language model for information retrieval*. Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval, ACM, 2015.

[11] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. *A deep relevance matching model for ad-hoc retrieval*. Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, 2016.

[12] Geoffrey Hinton and Ruslan Salakhutdinov. *Semantic hashing*. SIGIR Workshop Information Retrieval and Applications of Graphical Models, 2007.

[13] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. *Convolutional neural network architectures for matching natural language sentences*. 2014.

[14] 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*. Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, ACM, 2013.

[15] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. *Pacrr: A position-aware neural ir model for relevance matching*. arXiv preprint arXiv:1704.03940, 2017.

[16] Joseph94m. make_pair_iter pair_generator. `https://github.com/faneshion/MatchZoo/issues/273`, 2018.

[17] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[18] Hans Peter Luhn. The automatic creation of literature abstracts. *IBM J. Res. Dev.*, 2(2):159–165, April 1958.

[19] Craig Macdonald, Richard McCreadie, Rodrygo LT Santos, and Iadh Ounis. From puppy to maturity: Experiences in developing terrier. *Proc. of OSIR at SIGIR*, pages 60–63, 2012.

[20] Menion93. Import model from Keras 2.0. `https://github.com/deeplearning4j/deeplearning4j/issues/4124`, 2017.

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[22] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. *Learning to match using local and distributed representations of text for web search*. Proceedings of the 26th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2017.

[23] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. *Learning to match using local and distributed representations of text for web search*. Proceedings of the 26th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2017.

[24] Bhaskar Mitra, Eric Nalisnick, Nick Craswell, and Rich Caruana. *A Dual Embedding Space Model for Document Ranking*. CoRR, 2016.

[25] Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Christina Lioma. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

[26] Iadh Ounis, Gianni Amati, Plachouras Vassilis, Ben He, Craig Macdonald, and Christopher Johnson. Terrier Information Retrieval Platform. In *Proceedings of the 27th European Conference on IR Research (ECIR 2005)*, volume 3408 of *Lecture Notes in Computer Science*, pages 517–519. Springer, 2005.

[27] Iadh Ounis, Lioma Christina, Craig Macdonald, and Vassilis Plachouras. Research directions in terrier. *Novatica/UPGRADE Special Issue on Web Information Access, Ricardo Baeza-Yates et al. (Eds), Invited Paper*, 2007.

[28] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, and Xueqi Cheng. *A study of matchpyramid models on ad-hoc retrieval.* arXiv preprint arXiv:1606.04648, 2016.

[29] Pharnoux. Unsupported keras layer type Model. `https://github.com/deeplearning4j/deeplearning4j/issues/3313`, 2017.

[30] John Platt, Kristina Toutanova, and Wen-Tauh Yih. *Translingual doc-ument representations from discriminative projections.* EMNLP, 2010.

[31] Stephen Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. *Okapi at TREC-3.* Proceedings of the Third Text REtrieval Conference, Gaithersburg, USA, November 1994.

[32] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. *Are loss functions all the same?* Neural Comp., vol. 16, no. 5, pp. 10631076, 2004.

[33] Parnia Samimi and Sri Devi Ravana. Agreement of relevance assessment between human assessors and crowdsourced workers in information retrieval systems evaluation experimentation.

[34] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. *CoRR*, abs/1608.07249, 2016.

[35] Skeydan. Unsupported keras layer type Conv2D. `https://github.com/deeplearning4j/deeplearning4j/issues/3886`, 2017.

[36] Stanford. Assessing relevance. `https://nlp.stanford.edu/IR-book/html/htmledition/assessing-relevance-1.html`.

[37] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. *Neural vector spaces for unsupervised information retrieval.* 2017.

[38] Jeroen Vuurens and Alfred de Vries. Obtaining high-quality relevance judgments using crowdsourcing. *IEEE Internet Computing*, 16(5):20–27, Sept 2012.

[39] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. *End-to-end neural ad-hoc ranking with kernel pooling.* Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2017.

[40] Liu Yang, Qingyao Ai, Jiafeng Guo, and W Bruce Croft. *aNMM: Ranking short answer texts with attention-based neural matching model.* Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, 2016.

[41] Hamed Zamani and W Bruce Croft. *Estimating embedding vectors for queries.* Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval, ACM, 2016.

[42] Shuo Zhang and Krisztian Balog. *On the fly table generation.* 2018.

[43] Guido Zuccon, Bevan Koopman, Peter Bruza, and Leif Azzopardi. *Integrating and evaluating neural word embeddings in information retrieval.* Proceedings of the 20th Australasian document computing symposium, ACM, 2015.