Relevance Ranking for Web Search

João Lages
INESC-ID / Instituto Superior Técnico
Lisboa, Portugal
joao.lages@tecnico.ulisboa.pt

Joao Paulo Carvalho
INESC-ID / Instituto Superior Técnico
Lisboa, Portugal
joao.carvalho@inesc-id.pt

***Abstract*— Relevance ranking is a core problem of Information Retrieval which plays a fundamental role in various real world applications, such as search engines. Given a query and a set of candidate text documents, relevance ranking algorithms determine how relevant each text document is for the given query. This degree of relevance allows them to rank the text documents and perform actions such as returning the best matching documents for the query. As in other machine learning and computational intelligence disciplines, deep learning techniques have recently achieved state of the art results by successfully capturing relevance matching signals between query-textual document pairs. This paper focuses on the PositionAware Convolutional-Recurrent Relevance Matching approach. On a first phase, it reimplements the original work, reproduces the published results and performs a number of additional experiments that identify potential model limitations. On a second phase, it explores possible model improvements based on deep learning techniques such as soft self-attention and deep transfer learning. Experiments on the well-known TREC Web Track data show that it is possible to obtain small improvements over the original model and point to a number of limitations of the general approach due to the information bottlenecks involved.**

Keywords—Relevance Ranking, Text Retrieval, Natural Language Processing, Deep learning

I. Introduction (*Heading 1*)

Document relevance ranking is a core problem in Information Retrieval (IR) and plays a fundamental role in popular search engines such as Google, Bing, Baidu, or Yandex. In traditional Web search, the query consists of only few terms but the body text of the documents can range vastly in length. A ranking model aims at evaluating the relation between a query and different text documents, assigning higher scores to documents that are more relevant to the input query. In the absence of click information, the raw body text can be a useful signal to determine the relevance between a query-document pair. Traditional machine learning and computational intelligence algorithms for document relevance ranking relied on handcrafted features to encode interactions between queries and documents. This feature engineering work was usually time-consuming, incomplete and over-specified, which largely hindered further development of these approaches.

In recent years, new algorithms have been reported in the area of Information Retrieval [1][2], that apply deep neural networks for this purpose. As it happened in areas such as vision or speech processing, the potential of deep learning methods to advance state of the art retrieval quality has attracted a lot of attention. The newly introduced neural IR models differ from previous approaches in that they model the interactions between query and document directly based on the raw text, without the need for manual feature engineering. However, unlike many classical IR models, these new deep learning based approaches are data-hungry, requiring large scale training data before they can be employed.

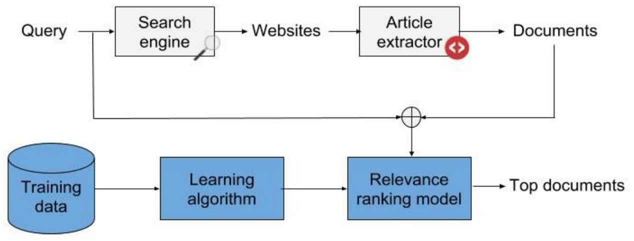


Fig. 1. Overview of a common Web retrieval system. This work is within the shaded components.

This work is motivated by this recent success of neural IR to relevance ranking, as well as the large improvements in performance attained in other areas of NLP by employing deep learning techniques applied directly to raw text. With the objective of improving the performance of prior neural relevance ranking models, this work considers only data in the form of text and no other information about either the query or document. A model is therefore developed with the intention of reranking search results, reordering the previously extracted texts by their descending relevance score to the given query. This ranking model is trained using a supervised machine learning approach, using a set of querydocument pairs labeled by human annotators.

II. Related Work

Deep learning applied to ad-hoc retrieval provides a new way of thinking about the problem as a general text matching task, i.e., query matches document. According to Guo et al. [3], these approaches can be mainly categorized into representation focused models and interaction focused models. The representation focused models, commonly referred to as semantic matching models, try to embed both queries and documents in a low-dimensional space with a neural network, and then conduct matching between the two vectors. Interaction focused models first build the local interactions between the two texts and use deep learning to learn the more complicated interaction patterns for matching. This type of algorithms can be also referred to as relevance matching models.

Salakhutdinov and Hinton [4] introduced one of the first deep neural models for ad-hoc retrieval, the Semantic Hashing model. The algorithm is a deep auto-encoder, trained on unlabeled document corpus, that represents words in onehot vectors and uses binary hidden units to encode those documents so that they can be posteriorly quickly retrieved using a hash function. The model is first pre-trained layer-bylayer and then trained further end-to-end for additional tuning. Given a search query, a corresponding hash is generated and the relevant candidate documents that match the same hash vector are retrieved. A standard IR model (e.g., BM25) can then be employed to rank between the selected documents.

More recently, Huang et al. [5] applied siamese networks to ad-hoc retrieval. Their Deep Semantic Similarity Model (DSSM) trains on query and document title pairs where both the pieces of texts are represented as bags-of-charactertrigraphs. The DSSM architecture consists of two deep models - one for the query and the other for the document - with all fully-connected layers and cosine distance as the choice of similarity function in the middle. This architecture was later improved by Shen et al. [6] by making use of convolutional layers.

This use of Convolutional Neural Networks (CNNs) encouraged posterior interaction-focused models [7], although with still worse results than traditional IR approaches. Still, the Deep Relevance Matching Model (DRMM) [3] was able to significantly outperform them by modeling query-document interactions using matching histograms. Mitra et al. [8] introduced a state-of-the-art model that makes use of both local interactions and distributed representations, arguing that a combination of the two approaches is preferred.

Subsequently, Xiong et al. [9] and Hui et al. [10] published results on interaction-focused models that would take as inputs only the similarity matrices between the query and document terms, t_q and t_d, respectively. While the former method applied RBF kernels to that matrix, the latter used multiple convolutions with different kernel sizes along with pooling layers. This last model, the Position-Aware Convolutional-Recurrent Relevance Matching model (PACRR), will be analyzed in detail in the following section.

III. Baseline Model

The baseline framework adopted in this work is based on the PACRR model [10]. Being a relevance matching model, the core concept of this architecture is to construct first a word similarity matrix between the query and the document words and use convolutional neural networks to account for soft pattern matches involving one or more adjacent words (n-grams). These signals are then non-linearly combined to produce a single query-document relevance score. The result is a non-linear function mapping a query document pair to a relevance scalar value. Following the usual neural network approach, this model can be trained end-to end using gradient descent techniques. An overview of the architecture excluding the pre-processing step can be seen in Figure 2.

A. *Pre-processing and Word Embeddings*

On a first step, a vector representation for each word in the input query and document must be obtained. For this purpose, the raw text is stripped of irrelevant components (e.g., HTML tags) and tokenized into words. Stop-word removal is also applied to both document and query, i.e., removing common English words like ’and’, ’or’ and ’the’. Finally, following the standard procedure in other Natural Language Processing tasks, each different word in the vocabulary is assigned a real-value vector representation termed word embedding. In principle, word embeddings can be trained using the available document query data as any other parameters of the model. In practice, this demands large amounts of data to prevent overfitting. For this reason, pre-trained embeddings are utilized for each word and fixed during training. Pre-trained embeddings can be seen as a simple form of transfer learning where word representations are trained from another task. Following [10], 300-dimensional Word2Vec CBOW embeddings [11,12] are computed from the whole text corpus, comprising documents and queries. CBOW is trained by predicting the center word of a word window giving its surrounding context and negative sampling is used to speed up training time.

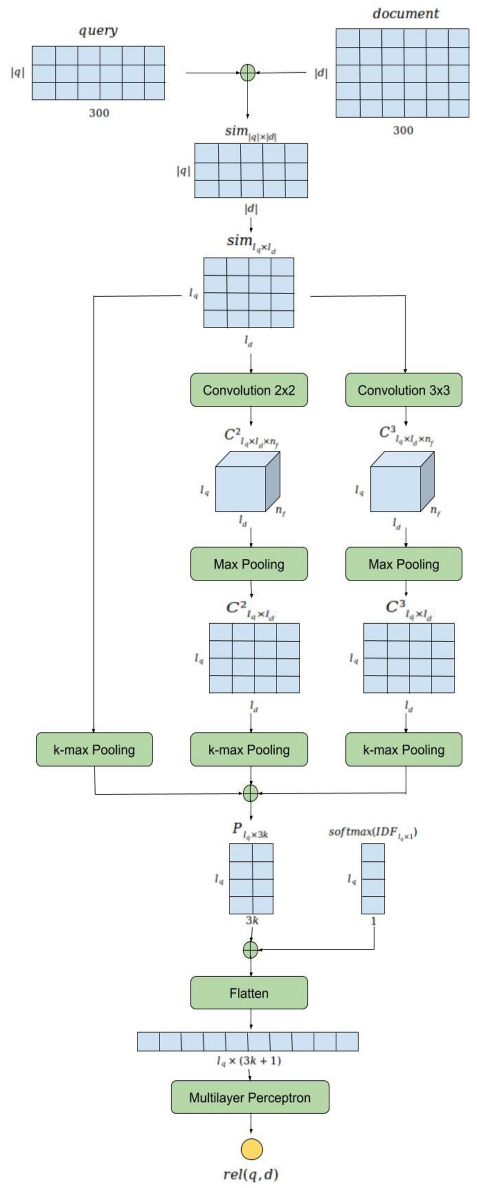


Fig. 2. Overview of the baseline model.

Thenceforth, query and document terms (t_q and t_d) are represented by 300-dimensional pre-trained vectors. Since queries and documents may have an arbitrary number of words, the resulting query and document representation has variable size. This presents an additional difficulty for the sub-sequent model stages that need to dynamically adapt to these sizes. To solve this problem, query $|q|$ and document $|d|$ sizes are cropped to fixed maximum values l_q and l_d, respectively. This is achieved by selecting the first l_q or l_d terms for query and document, respectively, when either $|q|$ or $|d|$ exceed the maximum fixed dimension. If the actual dimension of the query or document is lower than the desired, then the remaining vectors are padded with zeros.

B. Distance Matrix and Relevance Matching Components

Word distance is obtained by applying the cosine similarity between document and query vectors. This yields the query-document similarity matrix $sim_{lq \times ld}$, which is then reshaped to have a 3D shape ($l_q \times l_d \times 1$), and two 2D convolutional layers are applied to it with different kernel sizes, 2×2 and 3×3, corresponding to bi-gram and trigram matching, respectively. The original matrix accounts for unigram matching.

Each convolutional layer applies n_f different filters to its input, where n_f is a hyperparameter. At the end of this step we end up with two 3D matrices, $C^2_{l_q \times l_d \times n_f}$ and $C^3_{l_q \times l_d \times n_f}$, as well as the initial 2D similarity matrix $sim_{lq \times ld}$. In order to capture the strongest similarity signals for each query term, max pooling is performed over the filter dimension n_f to keep only the strongest signal from the n_f different filters.

The model then captures the top-k matching signals, for each query term and for each of the 3 matrices, by applying k-max pooling layers. This layer is implemented in such a way that, for each row, the higher k values are retained, keeping the same number of matching signals for each query term t_q. Previous to inputing the matching signals to the multilayer perceptron, the query IDF vector, $IDF_{lq \times 1}$, is passed through a *softmax* layer for normalization, such that a vector of probabilities is joined with the matrix P thereafter.

Additionally, a vector containing the IDF score of each query term t_q is also computed, $IDF_{lq \times 1}$. This vector has length l_q and will be passed directly to the last layers of the model as we can see in Figure 2.

The matrix P along with $IDF_{lq \times 1}$ are then flattened into a single vector of size $l_q \times (3k+1)$ and passed to a feed-forward network with two hidden layers and a final single neuron that produces the final relevance score, $rel(q,d)$.

C. Training Phase

To train a neural network in a supervised manner, it is necessary to repetitively feed it train data pairs (x,y) in order to minimize a certain loss function. In our case, since the final objective is to rerank documents, we are not interested in predicting a $\hat{y}$ label, as that does not allow proper ranking. The neural model needs to distinguish relevant documents from less relevant (but likely not completely non-relevant).

The most common approach for neural ad-hoc retrieval is to train pairwise with a set of documents D^+ and a set of documents D^- for the same query q, where D^+ are more relevant than D^-. This allows the model to distinguish, and therefore to rank, relevances between documents. In this architecture only a single relevant document d^+ is used and the number of D^- documents is a parameter that was fixed to 6. So this means that, for each training sample the model described in Figure 2 is actually repeated seven times, with a loss function gathering all the outputs as in (1).

$$L(q, d^+, D^-) = -\log \frac{e^{rel(q,d^+)}}{e^{rel(q,d^+)} + \sum_{d^- \in D^-} e^{rel(q,d^-)}} \tag{1}$$

IV. Proposed Changes

The previously described framework has some strong assumptions and simplifications that we explore in this work. In this section, we will go through some of these problems and suggest a modification, always with the objective of enhancing a particular building block of the original model that may cause an information bottleneck, i.e., loss or disregard of relevant data.

A. Lack of regularization

Deep neural networks contain multiple non-linear hidden layers that allow them to learn very complicated relationships between their inputs and outputs. However, these functions are non-convex and therefore the gradient descent algorithm does not give us a formal guarantee of convergence to the optimal solution and overfitting can also occur. The original architecture pictured in Figure 2 has no kind of regularization mechanisms applied, making the neural network subjective to overfitting. In order to prevent this and make the results more consistent, many methods have been developed, that are now common practice in deep learning. These include stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization or the simple dropout mechanism [15]. In this way, we present a way of regularizing the architecture by adding dropout layers after different blocks with trainable weights, that is, following the CNNs (after the max pooling operation) and the hidden layers of the MLP.

B. Parameterless pooling operations

Reducing the dimensionality of 3D and 2D matrices is a fundamental part of the baseline model represented in Figure 2. Although using parameterless pooling operations solves this issue without the introduction of new trainable variables, they can also be detrimental to the model's performance since the network is not considering a lot of information that was removed by an aggressive pooling function like maximum. Considering this a limitation, we propose changes to the two pooling layers used in the baseline model, as depicted in Figure 3.

Even though max pooling has been a standard operation when applied to images, we argue that a non-linear dimensionality reduction might be more suitable when dealing with similarity matrices. Without increasing the learnable parameters of the network too much and motivated by its success in computer vision [13][14], a 1×1 convolution is proposed to replace the max pooling layer, which allows the neural model to learn a more suitable function to reduce 3D matrices into 2D without imposing the max operation, as it was done previously.

Moreover, intuitively, the application of k-max pooling layers seems like a necessary step that allows us to employ a parameterless way of reducing the matrices size to something that is feasible for an MLP to handle. Notwithstanding, choosing only the top k entries of each row is a strong assumption that might be an impediment for the neural network to learn a better representation of the problem, and consequently improving its performance.

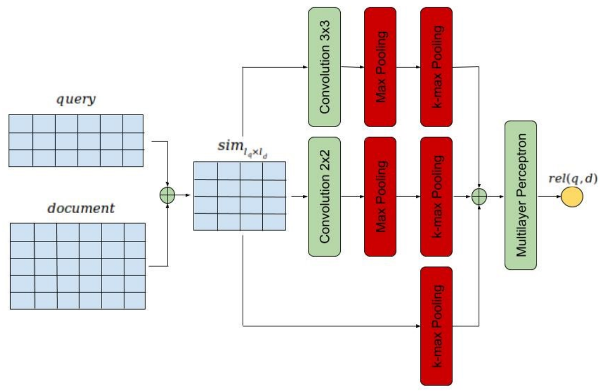


Fig. 3. Simplified baseline model: pooling layers marked in red.

Inspired by the soft attention mechanism and its recent success in other areas, we propose an architecture that allows the reduction of each row of the similarity matrices, with size l_d, into a single score. This score is obtained with:

$$z(x) = softmax(v \cdot \tanh(E_{q_x} W_q + E_d W_d)) \bullet x \tag{2}$$

This mechanism has three variables that are going to be learned while training: the vector v with dimension $(1,\sigma)$ and matrices W_q and W_d, both with size $(300,\sigma)$, where σ is a hyperparameter. The width 300 of the matrices is equal to the length of the Word2Vec embeddings used.

With the final objective of reducing the row's size from l_q to 1, the proposed attention mechanism takes as inputs a row of the similarity matrices x, the embeddings of the document terms E_d and the query term embedding E_{qx} associated with the given row x, as each row contains the interactions of a query word with all documents words. Note that, per matrix, this layer is applied l_q times, one for each row. The intuition behind this attention is to perform a weighted average of the whole row, so that the information of it can be summarized into a single score. This is accomplished by first using the embeddings to learn a more suitable representation of the query term $E_{qx}W_q$ and the document terms E_dW_d. Those representations are then joined, by summing elementwise the vector $E_{qx}W_q$ with all the rows of E_dW_d, and passed through an activation function, tanh. At this point, the output of the non-linearity has shape (σ,l_d) and the vector v will be responsible to learn a linear transformation to reduce it to the same length of x, l_d. *softmax* is then applied to normalize this final vector and the final score is obtained by calculating the dot product with the row x in case.

C. Static Similarity Matrix

In this architecture we consider the input $sim_{lq\times ld}$ to be the biggest information bottleneck, since documents and queries are only represented by local interactions and other semantic information about them is lost in the process. We argue that a considerable improvement to the model's performance might be achieved by altering the way query and document terms are represented, and by analyzing the advantages of using the query IDF vector. In this work two methods are experimented to extract more information from the text: 1D convolutions and a soft attention mechanism.

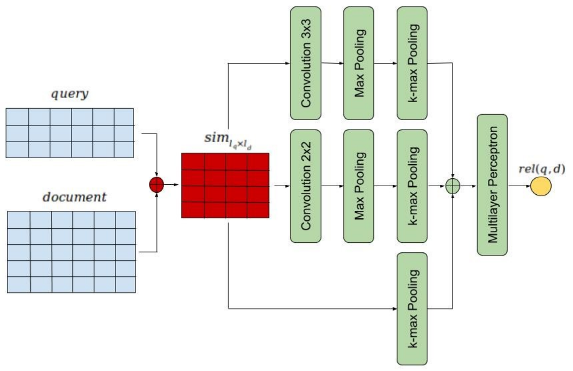


Fig. 4. Simplified baseline model: similarity matrix construction marked in red.

Not to confuse with 1×1 convolutions, one-dimensional convolutions act on 2D inputs. Commonly used in Natural Language Processing, it is normal to employ them after an embedding layer, in order to model word interaction within a context defined by the convolution window size. We argue that, even though 2D convolutions are already applied to detect bi-grams and tri-grams matching patterns between query and document, the similarity matrix, which is the basis of the model, consists of only unigram matches between words. Imagining we have two expressions: 'feeling blue' and 'being sad'. These two expressions have exactly the same meaning. Yet, if we calculated the cosine similarity of every word pair combination, we would probably get a high value for 'feeling' and 'being' but a low one for 'sad' and 'blue'. In this case, the cosine similarity matrix would not be able to properly detect that the two expressions are identical.

Therefore, we propose to implement an extra similarity matrix that is constructed the same way as the original one. However, beforehand, both the document and query embeddings will be passed by a 1D convolution layer that will theoretically learn how to represent more than one word in one single vector. The number of words that it represents depends on the kernel size used. The final extra matrix is then propagated through the network like the unigram matrix.

With the intention of increasing the local interaction information between every query term t_q and t_d, in comparison with the previously used cosine similarity, the matrix will now be three-dimensional, with size $l_q \times l_d \times n_c$. To attain this objective, another attention-like mechanism is employed. The word embeddings of both queries E_q and documents E_d are passed by a hidden layer, with n_c perceptrons, that first reduces their dimension by applying a linear projection, without using a non-linear activation function. The new projected embeddings, E_{qp} and E_{dp}, are in this way obtained and combined to produce the final 3D matrix $sim_{l_q \times l_d \times n_c}$, so that, given arbitrary positions i, j and k,

$$sim_{l_q \times l_d \times n_c}(i,j,k) = E_{q_p}(i,j) + E_{d_p}(k,j), \tag{3}$$

where E_{qp} and E_{dp} have sizes (l_q,n_c) and (l_d,n_c), correspondingly.

This new input matrix does not change the rest of the architecture and allows the CNNs to develop three dimensional filters and take into account more information for every t_q and t_d combination. Additionally, the use of cosine similarity is eradicated and the network will theoretically learn a more suitable query-document relationship function for the ad-hoc retrieval task.

D. *Loss Function*

We have previously explained the way a binary cross entropy is used in the baseline model as a cost function. In order to minimize this loss, the neural network updates its parameters according to the gradient of this function. Therefore, the way the loss function is constructed is ultimately a decisive factor for a good performance of the model.

In this work, a different loss function will be explored, a cross entropy with custom gains, that was designed to have a similar behavior as the evaluation metrics used. Backed up by its previous success, reported by Zamani et al. [16], the new cost function is designed as

$$L = - g(y^+, [y^+, Y^-])h(d^+) - \sum_{d^- \in D^-} g(Y^-_{d^-}, [y^+, Y^-])h(d^-) , \qquad (4a)$$

$$g(a,B) = \frac{2^a - 1}{\sum_{b \in B} 2^b - 1} , \qquad (4b)$$

$$h(d) = \log \frac{e^{rel(q,d)}}{e^{rel(q,d^+)} + \sum_{d^- \in D^-} e^{rel(q,d^-)}} , \qquad (4c)$$

where Y^- are the labels of the negative documents D^-, y^+ is the label of the positive document d^+, q represents the query, g is a gain function that is also applied in the nDCG [17] metric and h is the function used to normalize all the query-document relevance scores.

In addition to making the new loss function be more proportional to the evaluation metric, this new formulation also allows the use of a custom cross entropy with all the true labels involved, which were not directly employed in the preceding version (1).

V. Results

The experiments were made on two years of an IR competition, the TREC Web Track from 2013 and 2014. ClueWeb09-B and ClueWeb12 are the datasets used for our experiments, both of them used in this competition from 2009 to 2012 and 2013 to 2014, respectively. The former will only be used for training and validation, since access to the full ClueWeb09 dataset was not obtained, only to its category B, that contains less data than the original. The whole dataset is needed during test time because comparisons are made with other methods that reported results on it as well. Both datasets consist of millions of HTML Webpages, although only the small labeled part present in the TREC Web Track is used.

As it was previously done in the area [10], experiments of this work are mainly evaluated based on the nDCG@20 [17] and ERR@20 [18] score of reranks done over the TREC query likelihood baseline. The baseline consists of no more than a file containing, per query, up to ten thousand documents ordered by their relevance, according to the query likelihood model. This type of evaluation fits the way the whole retrieval system is constructed (Figure 1), considering that the model can be used to rerank documents provided by a search engine. Moreover, by using metrics @20, only the top twenty documents for each query are evaluated, which means that the assessment hardly penalizes systems who are not able to distinguish relevance grades.

TABLE I. ERR@20 AND NDCG@20 VALUES COMPARISON BETWEEN PACRR [10], QUERY LIKELIHOOD AND RPACRR

			ERR@20			nDCG@20		
Model	TrW	TrD	2013	2014	mean	2013	2014	mean
Query Likelihood	-	-	0.101	0.131	0.116	0.190	0.231	0.211
PACRR [10]	6061	OM	0.166	0.221	0.194	0.295	0.339	0.317
RPACRR	5552	OM	0.163	0.214	0.189	0.451	0.436	0.444
		SM	0.161	0.199	0.180	0.437	0.415	0.426

Stochastic gradient descent is enforced with Adam [19] as an optimizer, using a mini batch of 32 samples. To construct each sample of the batch, for each query q, a document d^+ is randomly sampled with rank x, as long as it has any D^- documents with rank $x - 1$. This sampling procedure allows the model to better differentiate documents with levels of relevance that are close to each other. The relevance group x is chosen with probability proportional to the number of documents in the group within the training set, so that the final training corpus keeps a distribution of labels similar to the initial.

Most of the hyperparameters are set according to the original used values by Hui et al. [10]. The most relevant ones are:

- l_d and l_q lengths in the similarity matrix are set to 800 and 16, respectively, and the first words of them are kept when their dimension exceeds these values;
- The size of the two hidden layers used in the multilayer perceptron at the end of the architecture is 32 and 16;
- Both the 2×2 and 3×3 convolution layers are applied with 32 filters;
- k-max pooling is used with $k = 3$.

The Diffbot API was used to convert the HTML Webpages to text strings. To deal with OOV words, the 300-dimensional Word2Vec CBOW vectors [12] are re-trained using Noise Contrastive Estimation with 5 negative samples and a context window size of 10.

Throughout our experiments, we resort to the use of dropout and do not add the normalized query IDF vector to the input of the MLP, as depicted in Figure 2, since we did not discovered gains in performance by using it.

A. *Reimplementation*

Reimplementing our baseline model, PACRR [10], is the first step to confirm our experiments' validity. It is important to remember that all the results were obtained by reranking the query-document pairs existent in the Query Likelihood submission. Therefore, Table I presents the results of both this model, the PACRR baseline model [10] and its reimplementation, RPACRR. TrW represents the number of trainable weights and TrD the type of data used to train the model. Since, the authors of PACRR [10] kindly open-sourced their own similarity matrices [20], whenever it is possible, experiments will be made using both this data (OM) and the selfmade matrices (SM) produced in this work. A key factor to retain is that the total amount of original matrices is around 120000 while the self-constructed ones are more or less 75000, which is notably lower. Moreover, the matrices are just the result of cosine similarity between query and document terms, which means that we do not have access to the raw text that Hui et. al [10] used.

By comparing these results from Table I, it is visible that PACRR was successfully reimplemented, if looking solely at the ERR@20 values, that are really similar. However, nDCG@20 scores were higher in our reimplementation, which leads us to conclude that RPACRR is actually better at getting the top 20 relevant documents but that the order of which it organizes them is mostly the same. nDCG only discounts based on the rank of the document, but does not consider any of the documents previously seen at higher ranks. On the other hand, ERR implicitly discounts documents based on the relevance of previously seen documents. In fact, as Chapelle at al. [18] claim, ERR differs from other metrics because it heavily discounts the contributions of documents that appear after highly relevant documents, i.e., the ones that appear at lower ranks.T hat explains the fact that nDCG@20 is higher, because there are more relevant documents, and ERR@20 is similar, since the first documents are the ones that contribute the most to this metric.

We believe this increase in the nDCG@20 score is due to some careful implementation details, and mostly to the added regularization.

Additionally, a slight decrease in performance when using the self-made matrices can also be noted. This phenomenon can be easily explained because of different data preprocessing and specially due to the fact that the original similarity matrices consist of way more training data than the self-made ones.

Table II presents the results from trying different proposed changes to the baseline model. During the remaining of this section, we will analyze these experiments, always comparing with the scores obtained in Table I.

TABLE II. ERR@20 AND NDCG@20 VALUES COMPARISON BETWEEN DIFFERENT RPACRR IMPLEMENTATIONS

			ERR@20			nDCG@20		
Model	TrW	TrD	2013	2014	mean	2013	2014	mean
RPACRR + CLF	5552	OM	0.173	0.226	0.200	0.452	0.437	0.445
		SM	0.170	0.210	0.190	0.444	0.425	0.435
RPACRR + 1x1conv	5616	OM	0.167	0.223	0.195	0.450	0.433	0.442
		SM	0.165	0.207	0.186	0.441	0.420	0.431
RPACRR + rSAtt	21680	SM	0.110	0.139	0.125	0.205	0.240	0.223
RPACRR + 1Dconv	16688	SM	0.142	0.175	0.159	0.252	0.293	0.273
RPACRR + 3DSAtt	19456	SM	0.127	0.159	0.143	0.220	0.256	0.238

B. Custom Loss Function

A characteristic that we wanted to investigate, using the original architecture of PACRR [10], was how the loss function was constructed. From Table II, it is possible to observe the results of using the custom loss function, CLF.

As expected theoretically, making the loss function closer to the metric that is being employed (nDCG in this case) increases the model's performance. The ERR@20 score also increases, manifesting that applying a custom cross entropy loss with a gain function similar to nDCG is definitely an improvement over the baseline.

C. Matrices Dimensionality Reduction

In this section we analyze two different ways of reducing dimensionality, giving some freedom to the model to learn what features to keep. Those methods are the previously described 1 × 1 convolution, replacing the need for max pooling, and the row-wise soft attention mechanism, that takes the place of the k-max pooling layers. Results of model runs with these methods are presented in Table II as 1x1conv and rSAtt, respectively. The latter was only run with the selfmade matrices as it requires access to the query and document embedding vectors.

Without using a lot of extra trainable parameters, it is possible to observe that 1 × 1 convolutions enhance the model's performance, as expected, since the model learns how to reduce third dimensions (channels) without applying an aggressive maximum operation.

In order to get similar results to the Query Likelihood baseline (Table I), a lot of hyperparameter tuning was needed for the row-wise attention mechanism. Replacing the k-max pooling layer was probably the most challenging task of this work, since k-max uses no trainable parameters, with the assumption that only the top *k* values of each row of the matrices are required for the subsequent layers.

From our experiments, the row-wise attention mechanism was the non-parameterless method that best worked for replacing k-max pooling. Nevertheless, it is possible to see that its results are similar to the Query Likelihood model. It can also be seen that the number of trainable parameters increases immensely when the k-max pooling layer is replaced, which might be indicative that a lot more train data is required to learn a proper dimensionality reduction with this row-wise attention mechanism.

D. Changing the Input

As previously discussed, having a single static cosine similarity matrix is a clear information bottleneck of the whole architecture. Table II shows results of methods that try to provide more additional information to the model, 1D convolutions (1Dconv), or improve the way that similarity matrix is constructed, soft attention mechanism (3DSAtt). Both architectures were only trained using the self-made matrices (SM) since we did not have access to the original embeddings used in baseline model [10].

As seen previously, the curse of the amount of learnable weights (TrW) is also present in here. In both methods, it is possible to note a decrease of the model's performance as this number of trainable parameters increases. Even though convolutions are known for not having a lot of learnable units, this 1D convolution has filters that act over embedding vectors with a dimension of 300, that increases the number of parameters drastically, even when using a kernel size of 2, when a reasonable number of these filters are applied.

E. *Discussion*

During the initial stage of these experiments we have successfully reproduced the baseline model and proposed modifications that proved to be advantageous and improved the model's performance in terms of ERR@20 and nDCG@20 scores. Nevertheless, it is important to point out that despite these deep learning systems showing good efficiency in reranking search results, they are limited by the retrieval system that selected the top documents for each query in the first place, which was the Query Likelihood model in our experiments and on the baseline model (Table I).

We have introduced regularization as a way to achieve better and more stable results and showed the importance of adapting the training scheme, approximating it more to the way these systems are evaluated.

Letting the model learn how to perform pooling operations can also be a beneficial factor, to some extent. I.e., we have shown how a slight change of the max pooling layer, without adding too many additional trainable parameters, results in a performance increase. On the other hand, alternative methods for replacing the k-max pooling did not work so well due to the amount of newly introduced variables.

Influenced by their recent success in Natural Language Processing [21], soft attention mechanisms have been employed extensively in this work. However, these mechanisms require a lot of data to train properly, which was not the case in here. Lack of training data was definitely a hard obstacle to overcome in this project, which conditioned how PACRR [10] was designed in the first place. Having only available 200 queries associated with the training set was definitely not enough to learn more complicated associations between query and documents that could then be generalized in the test set. Nonetheless, this was not clear at the beginning of this project, and it was only during our experiments that it became more evident that this was a transfer learning approach that constructs a soft matching function over fixed pre-trained word representations, which means that the pre-trained model plays a fundamental role.

In addition, there is no report of use of these type of mechanisms in the area of ad-hoc retrieval, which increased the motivation for our investigation. Although not performing as good as other architectures used in the first experiments, these models are still able to get a score increase over the Query Likelihood submission that they are reranking. This leads us to think that more work about the use of these methods still has to be explored and that they can eventually produce notable performance gains.

F. *Best Performing Model*

Having the previous experiments concluded, we also want to test a final model, that basically consists of an aggregation of the changes that yielded better results than the baseline model, i.e., superior ERR@20 and nDCG@20 scores than the ones reported for PACRR [10] in Table I. Therefore, this final model (Figure 5) is constructed with small modifications over the one depicted in Figure 2, those mostly being the replacement of the max pooling layers by convolutions with 1×1 kernels, the introduction of dropout layers after non-linear activations, the removal of the query IDF vector as an input to the network and the replacement of the binary cross entropy loss by a custom function (equation 4a).

TABLE III. ERR@20 AND NDCG@20 VALUES COMPARISON BETWEEN RPACRRF AND PACRR

			ERR@20			nDCG@20		
Model	TrW	TrD	2013	2014	mean	2013	2014	mean
PACRR [10]	6061	OM	0.166	0.221	0.194	0.295	0.339	0.317
RPACRRF	5616	OM	0.178	0.228	0.203	0.489	0.466	0.478
		SM	0.173	0.215	0.194	0.460	0.435	0.448

Table III shows the results for the final model, RPACRRF, comparing it with the original reported results by Hui et. al [10]. As expected, RPACRRF achieved better results than any antecedent model, according to the improvements previously obtained. It is also worth noting that this best performing model, depicted in Figure 5, does not have a notable computational overhead at test time when compared to the baseline model, since it ends up switching a maximum operation by a convolution but balances by having less input weights to the Multilayer Perceptron.

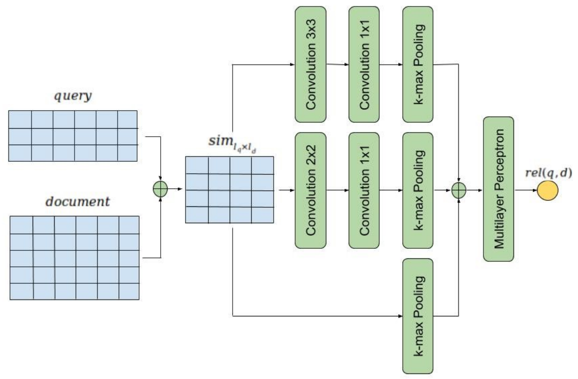


Fig. 5. Simplified final model.

The final model can be described as an extraction/ distillation-combination sequence, with CNN kernels extracting relevance matches, pooling layers and 1 × 1 convolutions distilling the matches into a series of small vectors for each query term, and a final block combining the query term signals into an ultimate relevance score for each query-document pair. Following this framework, we attempt to better understand the functionality of the convolution layers by visualizing their outputs.

As an example, we use the query "What are the symptoms of heart attack in both men and women", from the 2014 TREC Web Track. Figure 7 displays markups for different kernels sizes on the same document snippet, showing the strongest signal among all query terms, i.e., choosing the highest value for each column from the 2D matrix input to the different k-max pooling layers. The higher the opacity is (i.e., darkness of the text), the higher the output value is, in comparison with the other values from the same output matrix. The use of real valued cosine similarity in the input matrices allows the model to match related terms beyond exact matches, thereby expanding the query. For example, in Figure 7(a), terms such as ”coronary”, ”cardiac” and ”health” have relatively high weights though they do not appear in the query. Looking at the inputs that were passed through CNNs, Figure 7(b) and 7(c), we can see that most unigram signals still keep their relatively high weights and that other bigram and trigram matches increase the weights of other terms, with word combinations such as 'muscle strain' or 'coronary artery blockage'. For the higher dimension kernel signals, it is possible to notice that almost all terms have at least some weight, reducing the difference between the salient text and the remaining text. This is due to the way CNN kernels work when combined with real valued similarity. Taking the dot product of all terms in a window generally produces non-zero values and acts as a smoothing effect.

VI. Conclusions

The objective of this work is to contribute to the development of a reranking system for ad-hoc retrieval with deep learning techniques that eliminate the requirement for handcrafted query and document features.

We started by exploring previous work, selecting the state of the art model in the dataset available, and implemented it. New building blocks were then designed and optimized to mitigate different information bottlenecks of the neural network.

Despite the lack of a large volume of training data, this work managed to achieve good results in the reimplementation of a contemporary relevance matching model, on a known dataset, in the area of Information Retrieval. Moreover, it also shows some improvements by applying some techniques as regularization, 1×1 convolutions or tweaks in the loss function, as well as some proposed methods on how the model could potentially be adapted to retain or learn more information to ultimately produce a querydocument relevance score.

We expect that the work presented in this document will help future researchers in the areas of both Information Retrieval and Natural Language Processing. To aid scientific reproducibility, the full code base associated to this work is available at [22] for future use. This source code is the sole authorship of author 1.

Acknowledgment

This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020 and project LISBOA-01-0145-FEDER-031474

References

[1] K. Onal, Y. Zhang, I. Altingovde, M. Rahman, P. Karagoz, A. Braylan, B. Dang, H. Chang, H. Kim, Q. McNamara, A. Angert, E. Banner, V. Khetan, T. McDonnell, A. Nguyen, D. Xu, B. Wallace, M. de Rijke and M. Lease, *Neural information retrieval: at the end of the early years*, 2017.

[2] B. Mitra and N. Craswell, *Neural Models for Information Retrieval*, 2017.

[3] J. Guo, Y. Fan, Q. Ai and W. Croft, *A Deep Relevance Matching Model for Ad-hoc Retrieval*, 2016.

[4] R. Salakhutdinov and G. Hinton, *Semantic hashing*, 2009.

[5] P. Huang, N. Urbana, X. He, J. Gao, L. Deng, A. Acero and L. Heck, *Learning Deep Structured Semantic Models for Web Search using Clickthrough Data*, 2013.

[6] Y. Shen, X. He, J. Gao, L. Deng and G. Mesnil, *Learning Semantic Representations Using Convolutional Neural Networks for Web Search*, 2014.

[7] L. Pang, Y. Lan, J. Guo, J. Xu and X. Cheng, *A Study of MatchPyramid Models on Ad-hoc Retrieval*, 2016.

[8] B. Mitra, F. Diaz and N. Craswell, *Learning to Match Using Local and Distributed Representations of Text for Web Search*, 2016.

[9] C. Xiong, Z. Dai, J. Callan, Z. Liu and R. Power, *End-to-End Neural Ad-hoc Ranking with Kernel Pooling*, 2017.

[10] K. Hui, A. Yates, Berberich and G. de Melo, *PACRR: A Position-Aware Neural IR Model for Relevance Matching*, 2017.

[11] T. Mikolov, K. Chen, G. Corrado and J. Dean, *Efficient estimation of word representations in vector space*, 2013.

[12] Google, *Tool for computing continuous distributed representations of words*, https://code.google.com/archive/p/word2vec/ 2013.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, *GoogleNet*, 2014.

[14] M. Lin, Q. Chen and S. Yan, *Network In Network*, 2013.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 2014.

[16] H. Zamani, B. Mitra, X. Song, N. Craswell and S. Tiwary, *Neural Ranking Models with Multiple Document Fields*, 2017.

[17] K. Jarvelin and J. Kekalainen, *Cumulated gain-based evaluation of IR techniques*, 2002.

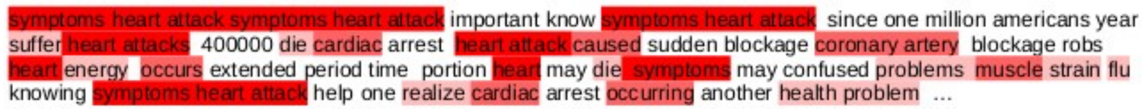
[18] O. Chapelle, D. Metlzer, Y. Zhang and P. Grinspan, *Expected reciprocal rank for graded relevance*, 2009.

[19] D. Kingma and J. Lei Ba, *Adam: a Method for Stochastic Optimization*, 2015.

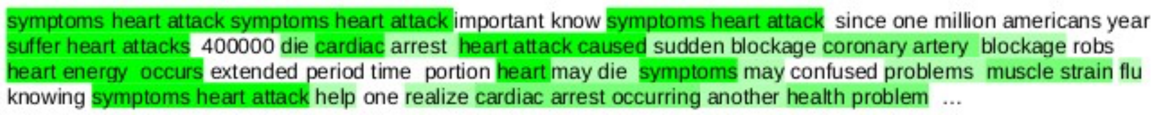
[20] K. Hui and A. Yates, *Cosine similarity matrices used in the original baseline model*, https://drive.google.com/file/d/0B3FrsWe6Y5YqdEtfSjI4N0h1LXM/view, 2017.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez and Kaiser, *Attention Is All You Need*, 2017.

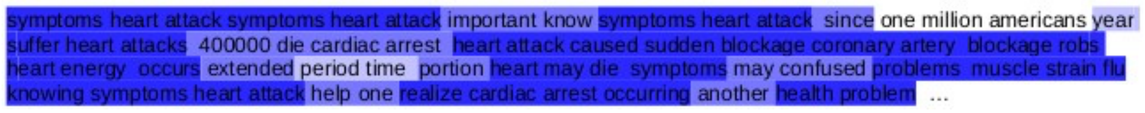
[22] João Lages, *Thesis source code repository*, https://github.com/JoaoLages/TREC _WebTrack 2018.


symptoms heart attack symptoms heart attack important know symptoms heart attack since one million americans year suffer heart attacks 400000 die cardiac arrest heart attack caused sudden blockage coronary artery blockage robs heart energy occurs extended period time portion heart may die symptoms may confused problems muscle strain flu knowing symptoms heart attack help one realize cardiac arrest occurring another health problem ...

(a) Text markup illustrating unigram term signals.


symptoms heart attack symptoms heart attack important know symptoms heart attack since one million americans year suffer heart attacks 400000 die cardiac arrest heart attack caused sudden blockage coronary artery blockage robs heart energy occurs extended period time portion heart may die symptoms may confused problems muscle strain flu knowing symptoms heart attack help one realize cardiac arrest occurring another health problem ...

(b) Text markup illustrating 2×2 kernel signals.


symptoms heart attack symptoms heart attack important know symptoms heart attack since one million americans year suffer heart attacks 400000 die cardiac arrest heart attack caused sudden blockage coronary artery blockage robs heart energy occurs extended period time portion heart may die symptoms may confused problems muscle strain flu knowing symptoms heart attack help one realize cardiac arrest occurring another health problem ...

(c) Text markup illustrating 3×3 kernel signals.

Fig. 6. Text markup illustrating the input to different k-max pooling layer