DeepTileBars: Visualizing Term Distribution for Neural Information Retrieval

Zhiwen Tang, Grace Hui Yang
InfoSense, Department of Computer Science
Georgetown University
zt79@georgetown.edu,huiyang@cs.georgetown.edu

Abstract

Most neural Information Retrieval (Neu-IR) models derive query-to-document ranking scores based on term-level matching. Inspired by TileBars, a classical term distribution visualization method, in this paper, we propose a novel Neu-IR model that handles query-to-document matching at the subtopic and higher levels. Our system first splits the documents into topical segments, "visualizes" the matchings between the query and the segments, and then feeds an interaction matrix into a Neu-IR model, DeepTileBars, to obtain the final ranking scores. DeepTileBars models the relevance signals occurring at different granularities in a document's topic hierarchy. It better captures the discourse structure of a document and thus the matching patterns. Although its design and implementation are light-weight, DeepTileBars outperforms other state-of-the-art Neu-IR models on benchmark datasets including the Text REtrieval Conference (TREC) 2010-2012 Web Tracks and LETOR 4.0.

Introduction

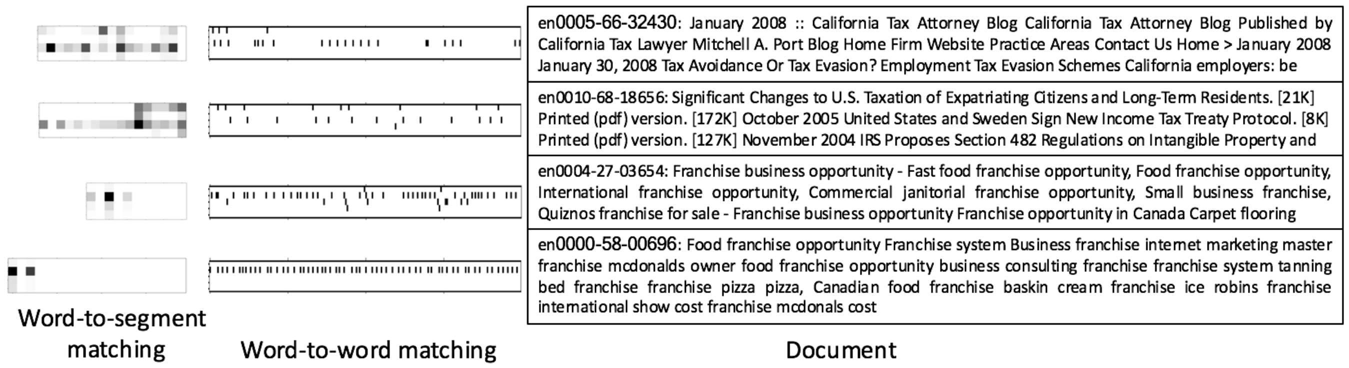
Numerous efforts have been devoted to advance Information Retrieval (IR) with deep neural networks (Guo et al. 2016; Pang et al. 2017; Hui et al. 2017; Fan et al. 2017; Pang et al. 2016b; Mitra, Diaz, and Craswell 2017; Xiong et al. 2017; Huang et al. 2013; Shen et al. 2014). These neural Information Retrieval (Neu-IR) models are often combined with a learning-to-rank framework (Liu and others 2009) to derive document relevance scores. Early experiments (Nguyen et al. 2017; Pang et al. 2016a) reported only marginal gains or even inferior performance to traditional IR methods such as BM25 (Robertson, Zaragoza, and others 2009) and language modeling (Zhai and Lafferty 2017). The more recent Neu-IR models attempted to incorporate well-known information retrieval principles and have started to show improvements over traditional IR methods.

The state-of-the-art Neu-IR models can be grouped into two categories (Guo et al. 2016). The first category is representation-focused. These models map the texts into a low-dimensional space and then compute document relevance scores in that space. Models in this family include DSSM (Huang et al. 2013) and CDSSM (Shen et al. 2014). The second category is interaction-focused. They first produce an interaction matrix, a.k.a. a matching matrix, between a query and a document. Each entry in the matrix is usually a simple relevance measurement, such as the cosine similarity, between the query term vector and the document term vector. The matrix is then fed into a deep neural network to learn the document relevance score. Models in this family include PACRR (Hui et al. 2017), DeepRank (Pang et al. 2017), DRMM (Guo et al. 2016), K-NRM (Xiong et al. 2017), MatchPyramid (Pang et al. 2016b) and HiNT (Fan et al. 2018). There are also hybrid models, such as DUET (Mitra, Diaz, and Craswell 2017), that combine the scores generated by the two categories.

The interaction-focused models are more popular. Partly, it is because of their close connection to the widely used learning-to-rank method. Also, they benefit from the interaction matrix's ability to reveal relevance signals visually, just like what a query highlighting function can do. For human readers, visualization could help accelerate the processing of information (Byrd 1999; Hornbæk and Frøkjær 2001). In real-world search engine development, there is also a high demand for visualization functions, such as query term highlighting and thumbnail images (Hearst 2009). These visualizations offer efficient, direct, and informative feedback to a search engine user. We think what makes visualization practically valuable to humans might also be valuable to a deep neural network for its resemblance to human neurons.

In the history of IR, visualizing relevance signals is not a new idea. In the 1990s, Hearst proposed TileBars (Hearst 1995) to help users better understand ad-hoc retrieval and the role played by each query term. TileBars visualizes how query terms distribute within a document and produce a compact image to explicitly show the relevance (the matches) to the user. In TileBars, a document is segmented with an algorithm called TextTiling (Hearst 1994). TextTiling splits the document at positions where a change in topic is detected. After that, query term hits are computed per term per segment and the distribution is plotted on a two-dimensional grid.

Given a query, TileBars attempts to capture the relevance patterns in the discourse structure of a document. Common patterns include chained, ringed, monolith, piecewise and hierarchical (Skorochod'ko 1971; Hearst 1994). TileBars focuses on the coarse patterns, especially the piece-



en0005-66-32430: January 2008 :: California Tax Attorney Blog California Tax Attorney Blog Published by California Tax Lawyer Mitchell A. Port Blog Home Firm Website Practice Areas Contact Us Home > January 2008 January 30, 2008 Tax Avoidance Or Tax Evasion? Employment Tax Evasion Schemes California employers: be

en0010-68-18656: Significant Changes to U.S. Taxation of Expatriating Citizens and Long-Term Residents. [21K] Printed (pdf) version. [172K] October 2005 United States and Sweden Sign New Income Tax Treaty Protocol. [8K] Printed (pdf) version. [127K] November 2004 IRS Proposes Section 482 Regulations on Intangible Property and

en0004-27-03654: Franchise business opportunity - Fast food franchise opportunity, Food franchise opportunity, International franchise opportunity, Commercial janitorial franchise opportunity, Small business franchise, Quiznos franchise for sale - Franchise business opportunity Franchise opportunity in Canada Carpet flooring

en0000-58-00696: Food franchise opportunity Franchise system Business franchise internet marketing master franchise mcdonalds owner food franchise opportunity business consulting franchise franchise system tanning bed franchise franchise pizza pizza, Canadian food franchise baskin cream franchise ice robins franchise international show cost franchise mcdonals cost

Figure 1: Word-to-segment vs. Word-to-word matching. Query terms are aligned vertically and document words/segments are aligned horizontally. The top two documents are relevant and the bottom two are irrelevant for TREC 2011 Web Track query 116 "California franchise tax board".

wise pattern. Other work in empirical discourse processing, for instance the Rhetorical Structure Theory, uses hierarchical models (Mann and Thompson 1988). Nonetheless, most work employs segments or subtopics as the analysis unit in their discourse models.

Unlike the discourse models, most Neu-IR models use words as the analysis units. In this research, we decide to learn from the discourse models. Inspired by TileBars, this paper studies the query-to-document matching at the segment or higher levels. We propose DeepTileBars, a new deep neural network architecture that leverages TileBars visualizations for ad-hoc text retrieval. By employing semantically more meaningful matching units, i.e. the segments, DeepTileBars is a step forward towards explainable artificial intelligence (XAI).

Figures 1 illustrate four TileBars visualizations for the query 'California franchise tax board'. The top two documents are relevant and the bottom two are irrelevant. The darker the cell, the more matches in the cell. With word-level query-to-document matching, we can see that the matched words are scattered around. In this example, an irrelevant document also has many dark cells because 'franchise' appears frequently in a long spam passage. It is quite difficult to tell the relevant matching patterns from the irrelevant ones. On the contrary, with word-to-segment matching, the words that belong to the same segment collapse into a single cell. Since the spam passage only contributes to one cell in the visualization, it is less likely for an irrelevant document to demonstrate patterns that are expected in a relevant one – for instance, continuous matched segments.

The piecewise discourse pattern used in the original TileBars paper already seems to be quite effective. We adopt it in this paper. In addition, we go beyond TileBars and include the hierarchical patterns in this work. To do so, we utilize a bag of Convocational Neural Networks (CNNs) (Krizhevsky, Sutskever, and Hinton 2012), each of which makes use of a kernel with a different size, to model the topical structure at various granularities. Practically, our model provides a hierarchical matching for a query-document pair. A relevance pattern could appear at any granularity in the hierarchy and our model aims to capture it at any level.

Our work focuses on text retrieval. Even though it is a visualization-based approach, image retrieval is not within our scope. Experiments on the Text REtrieval Conference (TREC) 2010-2012 Web Tracks (Clarke, Craswell, and Voorhees 2012) and LETOR 4.0 (Qin and Liu 2013) show that our model outperforms the state-of-the-art text-based Neu-IR models.

In the remainder of this paper, we first present our version of document segmentation and term distribution visualization. Note that this process belongs to the indexing phase of a search engine and will only be performed once for the entire corpus. Next, we describe an end-to-end Neu-IR model that makes use of these visualizations to retrieve documents. We then report the experiments, followed by the related work and the conclusion.

Visualizing Matched Text at the Segment Level

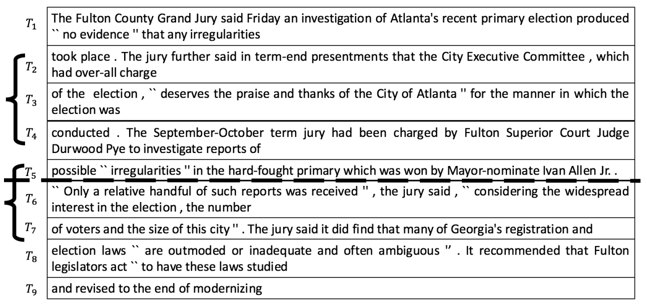
The construction of TileBars include three parts: segmentation, dimension standardization and coloring. The segmentation is done by TextTiling. We then prepare the interaction matrix into fixed dimensions and 'color' each cell.

Segmentation by TextTiling

TextTiling is a query-independent segmentation algorithm. It assumes that each document is composed of a linear sequence of segments, each of which represents a coherent topic. TextTiling inputs a document and outputs a list of topical segments. Figure 2 shows an excerpt from a long document that is used as a walking example to illustrate the algorithm. The algorithm takes three steps: Token Sequence Generation, Similarity Computation, and Boundary Determination.

1. **Token Sequence Generation:** We start with splitting a document into token sequences. A *token sequence* is like a pseudo-sentence, which contains a fixed number of words. After removing the stopwords, every α (set to 20 as recommended by Hearst) words are grouped into a token sequence.[1] These non-overlapping token sequences

[1]In our implementation, to preserve the natural paragraph



T_1	The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced `` no evidence '' that any irregularities
T_2	took place . The jury further said in term-end presentments that the City Executive Committee , which had over-all charge
T_3	of the election , `` deserves the praise and thanks of the City of Atlanta '' for the manner in which the election was
T_4	conducted . The September-October term jury had been charged by Fulton Superior Court Judge Durwood Pye to investigate reports of
T_5	possible `` irregularities '' in the hard-fought primary which was won by Mayor-nominate Ivan Allen Jr. .
T_6	`` Only a relative handful of such reports was received '' , the jury said , `` considering the widespread interest in the election , the number
T_7	of voters and the size of this city '' . The jury said it did find that many of Georgia's registration and
T_8	election laws `` are outmoded or inadequate and often ambiguous '' . It recommended that Fulton legislators act `` to have these laws studied
T_9	and revised to the end of modernizing

Figure 2: TextTiling ($\alpha = 20, \beta = 3$, stopwords are kept to preserve readability).

are the basic units to form a segment. In Figure 2, each line is a token sequence, denoted as T_i, with i indexes the sequences starting from 1.

2. **Similarity Computation:** Once obtaining the token sequences, the topical boundaries would be determined based on the similarity between the sequences as well as its contexts. The context for a token sequence is a sliding window of size β. β is set to 6 as recommended by Hearst. The similarity for two neighboring sequences is calculated over the two windows to which they each belong. Formally, given n token sequences, $T_1, T_2 ..., T_i, ..., T_n$, the similarity between T_i and T_{i+1} is calculated as the cosine similarity of term counts (tf) in the two windows $[T_{i-\beta+1}, ..., T_i]$ and $[T_{i+1}, ..., T_{i+\beta}]$:

$$sim(i, i+1) = \frac{\sum_{w_j} tf(w_j, i-\beta+1, i) \cdot tf(w_j, i+1, i+\beta)}{||tf(w_*, i-\beta+1, i)|| \cdot ||tf(w_*, i+1, i+\beta)||} \tag{1}$$

where w_j is the j^{th} word in the vocabulary, $tf\big(w_j, i-\beta+1, i\big)$ is the term frequency of w_j in a window holding sequences $T_{i-\beta+1}, ..., T_i$, and $tf\big(w_j, i+1, i+\beta\big)$ is the term frequency of w_j in a window holding $T_{i+1}, ..., T_{i+\beta}$, $||tf(w_*, i-\beta+1, i)|| = sqrt(\sum_{w_j} tf(w_j, i-\beta+1, i)^2)$, $||tf(w_*, i+1, i+\beta)|| = sqrt(\sum_{w_j} tf(w_j, i+1, i+\beta)^2)$.

Figure 2 illustrates an example to compute $sim(4, 5)$ with $\beta = 3$. The similarity between token sequences T_4 and T_5 is computed based on the window upto and including T_4, i.e. $[T_2, T_3, T_4]$, and the window after T_4, i.e. $[T_5, T_6, T_7]$. We obtain a similarity score for every neighboring pairs of token sequences.

3. **Boundary Determination:** The previous steps result in a series of similarity scores, each of which is created for a pair of token sequences. When plotting the similarity scores, we can observe "peaks" and "valleys" in this curve. When there is a dramatic drop in the curve, we say there is a possible topic change. A *depth* score is computed for each neighboring pairs of token sequences. It is defined as the sum of the absolute differences to its closest neighboring "peaks". The *depth* score between T_i and T_{i+1} is

$$\begin{aligned} depth(i, i+1) = &| \; sim(LeftPeak(i, i+1)) \\ &- sim(i, i+1) \; | \\ &+ | \; sim(RightPeak(i, i+1)) \\ &- sim(i, i+1) \; | \end{aligned} \tag{2}$$

where $LeftPeak$ computes the position of the closest "peak" on the left side and $RightPeak$ computes that on the right side. Boundaries are discovered at positions where the depth score is greater than $\mu - \delta/2$, where μ is the mean depth in the document and δ is the standard deviation. The threshold adjusts to different documents.

In Figure 2, a boundary is found between T_5 and T_6. Two topical segments B_1 (containing $T_1, T_2, ...$) and B_2 (containing T_6, T_7, T_8 and the rest) are generated for the document. When examining the actual content of these segments, we find that the two segments represent distinct topics: one is the overview of an investigation and another is about the jury's opinion.

TextTiling was a pioneering work in the 1990s for text segmentation. Unlike segmentation by a fixed passage length or by natural paragraphs, TextTiling's segments are based on term distributions and are expected to be topic-coherent. More sophisticated text segmentation methods have been proposed since. Examples include probabilistic models that use language modeling and language cues (Beeferman, Berger, and Lafferty 1999), clustering-based algorithms (Kazantseva and Szpakowicz 2011), topic modeling (Misra et al. 2009), and the recently proposed deep learning methods (Badjatiya et al. 2018). Compared with TextTiling, most successor approaches still hold the assumption that topics are laid sequentially in a document. Without loss of generality, we use TextTiling for its simplicity and effectiveness and focus on how a segment-based visualization can aid neural information retrieval.

Standardizing Dimensions

Each segment B_i would contain a different number of token sequences; and the total number of segments in a document would also vary from document to document. Note that this number is not only decided by the choices of token-sequence length (α) and window size (β) but also determined by the content of a document. In our experiments, there are documents consisting only of a title, which yield only one segment; whereas other longer documents split into hundreds of segments. As a result, for different query and document pairs, the dimensions of their TileBars visualizations vary. However, a neural network requires all its input vectors to be of equal size. We therefore need a mechanism to standardize the TileBars visualizations.

Suppose the original dimension of an image is $x \times y$, where x is the query length and y is the number of segments. Our task is to resize all visualizations to a chosen dimension $n_q \times n_b$.

Generally speaking, for visualizations with fewer than n_q query terms or n_b segments, we pad the grid with empty cells. This is equivalent to the zero padding technique widely

boundaries, rarely some token sequences are permitted to have sizes slightly different from 20.

used in computer vision (Szeliski 2010). For query q, its i^{th} word w_i is transformed into

$$w'_i = \begin{cases} w_i & i \leq x \\ \langle\text{empty word}\rangle & x < i \leq n_q \end{cases} \quad (3)$$

where x is the length of original query and n_q is the length of the transformed query and set to the maximum query length in the query collection.

Similarly to the query dimension, if $y \leq n_b$, i.e. the original document is relatively short, then, zero padding is done and each segment is transformed by:

$$B'_i = \begin{cases} B_i & i \leq y \\ \langle\text{empty segment}\rangle & y < i \leq n_b \end{cases} \quad (4)$$

where B'_i is the segment after transformation and B_i is the original segment. y is the original number of segments.

For documents with more than n_b segments, we 'squeeze' the content after (and including) the ${n_b}^{th}$ segment into the ${n_b}^{th}$ segment, which makes the ${n_b}^{th}$ segment the last in a document. This would preserve all the original content while only affect "resolution" of the last segment. Thus, if $y > n_b$, i.e., the original document is relatively long, then

$$B'_i = \begin{cases} B_i & i < n_b \\ concatenate(B_{n_b}, B_{n_b+1}, ...B_y) & i = n_b \end{cases} \quad (5)$$

After resizing, the visualization dimensions are fixed as $n_q \times n_b$ for all query-document pairs. From now on, we denote query words as w and segments as B after the standardization for the sake of notation simplicity.

Coloring

The original TileBars dyes the grids based only term frequency. The darker the color, the larger the intensity. In our work, we propose to incorporate multiple relevant features, similar to canonical colors in a color space, to "paint" the cells. Following the convention in multimedia research, we call each relevance feature a channel.

In theory, features indicating query-document relevance that are effective in existing learning-to-rank models can all be used here. However, to avoid interfering with the neural network's ability to select the best feature combinations, we propose to only employ features that are independent of each other. It is similar to only use a few scalar valued colors as the canonical colors in a color space, and leave the feature aggregation to the network itself.

Three features are used in this work. They are all proven to be essential in ad-hoc retrieval. They are term frequency, inverse document frequency, and word similarity based on distributional hypothesis. The $(i, j)^{th}$ cell in the matching visualization I for query w and segment B_j is 'painted' by:

$$\begin{pmatrix} tf(w_i, B_j) \\ idf(w_i) \times \mathbb{I}_{B_j}(w_i) \\ \max_{t \in B_j} e^{-(v_{w_i} - v_t)^2} \end{pmatrix} \quad (6)$$

where w_i is the i^{th} query term, B_j is the j^{th} text segment in a document, $tf(w_i, B_j)$ is the term frequency of w_i in B_j, and $idf(w_i)$ is the inverse document frequency of w_i. $\mathbb{I}_{B_j}(w_i)$ is an indicator function to show whether w_i is present in B_j. v_t is the embedded word vector of word t, which comes from a pre-trained word2vec model (Mikolov et al. 2013). We use Gaussian kernels as suggested by (Xiong et al. 2017; Pang et al. 2016a). The max operator is used to select the most similar word.

DeepTileBar: Deep Learning with TileBars

We propose a novel deep neural network, DeepTileBars, to evaluate document relevance. It consists of three layers of networks. It starts with detecting relevance signals with a layer of CNNs, followed by a layer of Long Short Term Memories (LSTMs) (Hochreiter and Schmidhuber 1997) to aggregate the signals. And then it decides the final relevance with a Multiple Layer Perceptron (MLP). Figure 4 illustrates the architecture of DeepTileBars. The input to the network is an $n_q \times n_b$ interaction matrix.

Word-to-Segment Matching

Following TileBars, DeepTileBars builds an interaction matrix by comparing a word and a topical segment. Each segment presumably corresponds to a topic. Per word information within the same segment is absorbed into a single cell and the word-level matching is no longer supported.

Using the proposed word-to-segment matching, we are able to produce relevance signals at the topic level and focus on the presence of query terms in consecutive topics. Matching queries to documents at the topic level, rather than at the word level, is perhaps easier to detect the stronger relevance signals with a bigger chunk of coherent text.

Our design is especially beneficial for proximity queries within a large window size. Most existing Neu-IR models would be sufficient for proximity queries within a tight window, e.g. 2 or 3 words apart (Hui et al. 2017; Pang et al. 2016a). However, when the required window size is large, scanning a word-to-word matching with CNNs might miss the true relevant. A word-to-segment matching, instead, could resolve this issue by merging topically coherent words into a single segment and obtain a stronger relevance signal. In addition, as demonstrated in Figure 1, segment-based matching would eliminate high hits in spam passages thus avoid false positives.

Bagging with Different Kernel Sizes

TextTiling partitions a document into segments and the segments are laid out sequentially without any further organization. However, topics in documents are often organized hierarchically. We can imagine that the segments form sections, and section form chapters, etc. If we could put several topical segments together, they might actually form a super topic – a topic that is at a higher level. It would thus be desirable if the topics and super topics can be handled at different granularities so that their hierarchical nature can be preserved.

Figure 3 shows the word-to-segment matching at different granularity levels. We combine k adjacent non-overlapping topical segments to show the term distribution at at different levels. k is the CNN kernel size. The bigger k is, the larger the CNN kernel, and the higher the topic level. We can see

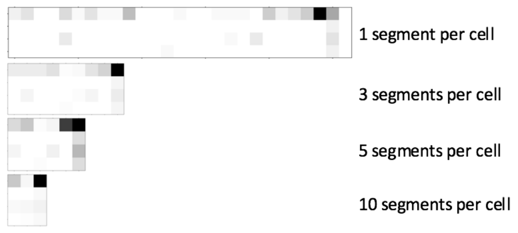


Figure 3: Word-to-segment matching at different granularity levels. Document enwp00-69-13554 for TREC 2011 Web Track query 116: California franchise tax board.

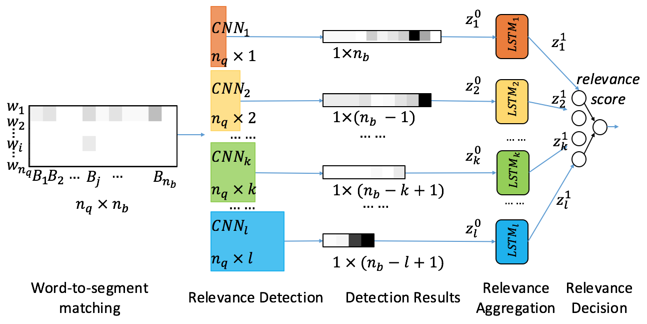


Figure 4: DeepTileBars Architecture.

that each granularity level provides different relevance patterns for the same document. A relevant pattern can appear at any level. Looking at the relevance patterns at all levels would thus increase our chances of finding the true relevant.

We thus propose to employ multiple CNNs (Krizhevsky, Sutskever, and Hinton 2012) with various kernel sizes to detect the relevance signals. Here we assume the kernel sizes vary from 1 to l, where l is the maximum number of segments the CNNs can handle. Practically, our model provides a hierarchical organization: token sequences are built from terms, segments (topics) are built from token sequences, and super topics are built by grouping adjacent topics with larger kernel sizes.

The Network

Figure 4 illustrates the network architecture. In the first CNN layer, there are l CNNs, $[CNN_1, CNN_2, ..., CNN_k, ..., CNN_l]$. Each CNN's kernel size differs. For the k^{th} CNN, its kernel size is $n_q \times k$. When each CNN scans through the input visualization, it produces a "tape"-like output. As k increases, the CNNs are able to capture the relevance signals at k adjacent text segments. We call these k adjacent text segments 'k-segment'. The l number of CNNs produce l outputs. The output from the k^{th} CNN denotes the relevance detected from all k-segments in the document. The shape of the k^{th} output is not a square, but a thin tape with dimension $1 \times (n_b - k + 1)$.

Note that Figure 3 is not the outputs of the CNNs. Instead, it shows the relevance signals per query term when combining adjacent topical segments. The CNN outputs are the third column of thin-tapes in Figure 4.

The CNN layer is denoted as z^0. The computation of z_k^0 with kernel size k is defined as

$$z_k^0 = CNN_k(I), \quad k = 1, 2, 3, ..., l \tag{7}$$

More specifically,

$$z_k^0[i] \;=\; act\left(\sum_{ch=1}^{|channels|}\sum_{u=0}^{n_q-1}\sum_{v=0}^{k-1}\theta_{k,ch}^0[u,v]\right.$$
$$\left. *I[u, i+v, ch] + b_k^0\right) \tag{8}$$

where act is the ReLU activation function, ch is the channel index, θ is the kernel coefficient, and b is the bias.

At the second LTSM layer, in order to minimize the loss of information, we use the same l number of LSTMs to accumulate the relevance signals. The output of the k^{th} CNN is fed into the k^{th} LSTM. An LSTM scans through its input step by step from the beginning to the end. The k^{th} LSTM then outputs its evaluation of the entire document at the granularity of k. For instances, the first LSTM, $LSTM_1$, accumulates relevance signals from all the 1-segments; the second LSTM, $LSTM_2$, accumulates those from all 2-segments. Thus, the LSTMs layer is able to output multiple relevance estimations at different granularities.

When scanning at the t^{th} timestep, where $t = 1, 2, ..., n_b - k + 1$, the k^{th} LSTM works as the following:

$$\begin{aligned} f_t^k &= \sigma(W_f^k[z_k^0[t], h_{t-1}] + b_f^k) \\ i_t^k &= \sigma(W_i^k[z_k^0[t], h_{t-1}] + b_i^k) \\ o_t^k &= \sigma(W_o^k[z_k^0[t], h_{t-1}] + b_o^k) \\ c_t^k &= f_t^k * c_{t-1}^k + i_t^k * tanh(W_c^k[z_k^0[t], h_{t-1}^k] + b_c^k) \\ h_t^k &= o_t^k * tanh(c_t^k) \end{aligned} \tag{9}$$

where $\sigma()$ is the hard sigmoid activation function, f_t^k, i_t^k, and o_t^k are the values of the forget gate, input gate, and output gate. c_t^k is the state. $W_f^k, W_i^k, W_o^k, W_c^k$ and $b_f^k, b_i^k, b_o^k, b_c^k$ are the weights and biases for the corresponding gates, h_t^k is the output, and [] is the concatenation operator. The computation of the second layer z^1 is thus

$$z_k^1 = LSTM_k(z_k^0) = h_{n_b-k+1}^k, \quad k = 1, 2, 3, ..., l \tag{10}$$

At the third layer of MLP, all outputs of LTSMs are concatenated together and fed into the layer to generate the final relevance decision. The computation of the third layer s is $s = MLP([z_1^1, ...z_l^1])$.

To summarize, the overall architecture (see Figure 4) of DeepTileBars is:

$$z_k^0 = CNN_k(I), \quad k = 1, 2, 3, ..., l \tag{11}$$

$$z_k^1 = LSTM_k(z_k^0), \quad k = 1, 2, 3, ..., l \tag{12}$$

$$s = MLP([z_1^1, z_2^1, ..., z_k^1, ..., z_l^1]) \tag{13}$$

We use a state-of-the-art Stochastic Gradient Descent (SGD) optimizer, Adam (Kingma and Ba 2015), to optimize the network. We also adopt the $L2$ regularization (at the first layer) and early stopping to avoid overfitting. The document ranking scores are obtained by a pairwise ranking loss function as in RankNet (Burges et al. 2005). It maximizes the difference between the relevant documents and the irrelevant documents: $J(\Theta) = \sum_{(q,d^+,d^-)} - \log \frac{1}{1+e^{-(s(d^+,\Theta)-s(d^-,\Theta))}}$, where s is neural network output (Eq. 13), Θ are the network parameters, and q, d^+, and d^- are query, relevant document and irrelevant document, respectively.

Experiment

We evaluate DeepTileBars' effectiveness on standard testbeds, including TREC Web Tracks and LETOR 4.0. We compare the performance of DeepTileBars with both traditional IR approaches and the state-of-the-art Neu-IR models.

Dataset and Metrics

We conduct experiments on the TREC 2010-2012 Web Track Ad-hoc tasks (Clarke, Craswell, and Voorhees 2012).[2] There are 50 queries each year. We combine the three years' data as a single dataset because the annotated data from any single year is not enough to train a deep model. In total, there are 150 queries and 38,948 judged documents. Our experiment is conducted on ClueWeb09 Category B, which contains more than 50 million English webpages (231 Gigabytes in size).[3] We implement a 10-fold cross-validation for fair evaluation. The official metrics used in TREC 2010-2012 Web Track ad-hoc tasks include Expected Reciprocal Rank (ERR)@20 (Chapelle et al. 2009), normalized Discounted Cumulative Gain (nDCG)@20 (Järvelin and Kekäläinen 2002) and Precision (P)@20. ERR and nDCG handle graded relevance judgments and Precision handles binary relevance judgements.

We also test our full model on the most recent MQ2008 dataset for LETOR 4.0. LETOR 4.0 is a common benchmark used by Neu-IR models. LETOR MQ2008 contains 784 queries and 15,211 annotated documents. The official metrics used in LETOR includes nDCG and Precision at different cutoff positions. We report results on queries that have at least one relevant documents in the judgment set.

Systems to Compare

- Traditional IR models: **BM25** (Robertson, Zaragoza, and others 2009) and **LM** (Zhai and Lafferty 2017). Both are highly effective IR models.
- TREC Best Runs: TREC Best is not a single system. Instead, it is a combination of best systems per year per metric, including the best systems in 2010 (Elsayed et al. 2010; Dinçer, Kocabas, and Karaoglan 2010), 2011 (Boytsov and Belova 2011) and 2012 (Al-akashi and Inkpen 2012). We believe they represent the best performance of TREC submissions from 2010 to 2012.
- Neu-IR models: **DRMM**, **MatchPyramid**, **DeepRank**, **HiNT**, **DUET**.[4]
- Variations of the proposed model: **DeepTileBars ($n_q \times x$)** that uses a CNN with a single kernel size $n_q \times x$; **DeepTileBars (w2w, all kernels)**, which uses all kernels with word-to-word matching; and **DeepTileBars(w2s, all kernels)**, our full model, using word-to-segment matching and all kernels.

Run	err@20	ndcg@20	p@20
TREC-Best	**0.188**	**0.236**	0.382
BM25	0.102	0.137	0.253
LM	0.118	0.166	0.297
DRMM	0.127	0.184	0.346
MatchPyramid	0.113	0.125	0.228
DeepRank	0.127	0.134	0.224
HiNT	0.157	0.205	0.322
DeepTileBars ($n_q \times 1$)	0.140	0.207	0.368
DeepTileBars ($n_q \times 3$)	0.150	0.212	0.369
DeepTileBars ($n_q \times 5$)	0.146	0.211	0.371
DeepTileBars ($n_q \times 7$)	0.142	0.207	0.366
DeepTileBars ($n_q \times 9$)	0.147	0.213	0.372
DeepTileBars (w2w, all kernels)	0.110	0.123	0.248
DeepTileBars (w2s, all kernels)	0.168	0.229	**0.384**

Table 1: TREC 2010-2012 Web Track Ad-hoc Tasks.

Parameter Settings

For the TextTiling algorithm, we set α to 20 and β to 6, as recommended by Hearst. For the query-document interaction matrix, the parameters are slight differences between the two datasets. In TREC Web, $n_q = 5$ and In LETOR $n_q = 9$. n_q is decided by the longest query after removing stopwords. For both datasets, $n_b = 30$. This is because more than 90% documents in TREC Web and more than 80% documents in LETOR contain no more than 30 segments.

For the DeepTileBars algorithm, we set $l = 10$ for both datasets. In TREC Web Track dataset, the number of filters of CNN with same kernel size and the number of units in each LSTM are both set to 3; while in LETOR, this number is set to 9. The MLP contains two hidden layers, with 32 and 16 units for TREC Web, and 128 and 16 units for LETOR.

On the TREC Web dataset, we re-implement DRMM, MatchPyramid, DeepRank, HiNT by following the configurations in their original papers. On LETOR, we include the reported results in their publications in Table 2 without repeating the experiments.

Results

Tables 1 and 2 report our experimental results for the TREC Web Tracks and the LETOR MQ2008 datasets. Official metrics are reported here.

[2]TREC 2009 also had the Web Track. However, it used relevance scales and metrics quite different from the other years. For fairness and consistency, our experiments exclude year 2009.

[3]http://lemurproject.org/clueweb09/.

[4]We did not run DUET on TREC Web Tracks because the dataset is too small to train a very deep model like DUET that requires a large amount of training data.

Run	p@5	p@10	ndcg@5	ndcg@10	Reported by
BM25	0.337	0.245	0.461	0.220	(Qin and Liu 2013)
LM	0.323	0.236	0.441	0.206	(Qin and Liu 2013)
DRMM	0.337	0.242	0.466	0.219	(Pang et al. 2017)
MatchPyramid	0.329	0.239	0.442	0.211	(Pang et al. 2017)
DeepRank	0.359	0.252	0.496	0.240	(Pang et al. 2017)
Duet	0.341	0.240	0.471	0.216	(Fan et al. 2018)
HiNT	0.367	0.255	0.501	0.244	(Fan et al. 2018)
DeepTileBars	**0.427**	**0.320**	**0.553**	**0.256**	

Table 2: LETOR-MQ2008.

It can be found that in the TREC Web Track dataset, DRMM, HiNT and our model DeepTileBars outperform traditional IR approaches. While other neural IR approaches, MatchPyramid, DeepRank do not perform as well on certain metrics. On the LETOR dataset, all the Neu-IR approaches achieve better performance than traditional methods. It indicates that properly designed deep neural networks could improve upon traditional approaches. We also notice that with different network architectures and different input formulations, the Neu-IR models achieve varied gains. It would be worthwhile exploring which architecture is a better fit for ad-hoc retrieval.

It is exciting to see that our model, DeepTileBars, outperforms other state-of-the-art neural IR systems in TREC Web Tracks and MQ2008. We think the gains come from the topical segmentation of the texts and the bagging of multiple CNNs with different kernel sizes. Word-to-segment matching provides relevance signals at the topic level. CNNs with different kernel sizes allow to evaluate document relevance at all granularities.

To investigate how DeepTileBars works internally, we experiment with a few variants of DeepTileBars with only one kernel. It can be found that adjusting the kernel size can only bring marginal improvement while the combination of all kernels boosts the retrieval performance.

When changing word-to-segment matching back to word-to-word matching, we observe a huge performance decrease. This confirms our claim that word-to-word matching is less desirable than word-to-segment matching in terms of finding relevance patterns.

However, we are still left behind by the TREC Best runs in some metrics. Some TREC Best runs used sophisticated term weighting methods without any deep learning (Dinçer, Kocabas, and Karaoglan 2010; Al-akashi and Inkpen 2012). Others used shallow neural networks or linear regression but with abundant feature engineering (Elsayed et al. 2010; Boytsov and Belova 2011). We look forward to the day when Neu-IR could catch up with the TREC best systems.

Related Work

There are only a few pieces of research that share similar intention with ours, i.e. visualizing the relevance signals in a document for deep learning. Works that are the closest to ours are MatchPyramid, HiNT, and ViP (Fan et al. 2017).

MatchPyramid was proposed for text matching tasks, including paraphrase identification and paper citation matching. By plotting the similarity between two sentences in an $n \times m$ matrix, where n and m are the lengths of two sentences respectively, deep neural networks were used to find the matching patterns between sentences with this image-like visualization. In MatchPyramid, the matching is done at the word level and experiments show it is less effective.

The more recent Neu-IR model, HiNT, adopted a similar idea to ours to perform passage-level retrieval. One major difference between HiNT and DeepTileBars is how they split the documents. HiNT splits documents by fixed sized passages whereas DeepTileBars splits them based on topic changes. The passages in HiNT are in fact more similar to our token sequences, which do not represent topical structure. As a result, the highest semantic level that HiNT is able to examine is similar to our segments. Levels higher than segments are not modeled in HiNT. In this sense, HiNT is not a true hierarchical Neu-IR model; instead, it is a segment-level only model. Meanwhile, although HiNT used a much more complex neural method, with a light-weighted architecture, DeepTileBars achieves better retrieval effectiveness in our experiments.

ViP also proposed to take advantage of visual features in a document. Instead of using the interaction matrix as the input image, ViP directly used a webpage's snapshot as so. ViP's experiments showed that even query-independent visual features would be able to improve the retrieval effectiveness. ViP's semantic units, such as the webpage sections and multimedia components, are similar to our hierarchical topics higher than the segments. In this sense, they are the most similar work to ours. However, we did not compare to their work in this paper due to our focus on texts.

Research from the field of Natural Language Processing has adopted similar designs of using multiple CNN kernel sizes. For example, (Kim 2014) used that for sentence classification. Their input was a $d \times m$ interaction matrix, where d was the dimension of the term vectors and m was the number of words in a sentence. They encoded a sentence using multiple CNNs with kernel sizes $d \times n$, where n was the size of an n-gram and obviously could vary. While our use of multiple kernel sizes is driven by the attempt to fuse relevance signals at multiple topical granularities, their modern way of representing n-grams (with different n) yielded a similar design to ours.

Conclusion

In this paper, we propose DeepTileBars, a Neu-IR model inspired by classical work in search engine visualization. The main contribution includes (1) word-to-segment matching and (2) bagging of different sized CNNs. Experiments show that our approach outperforms the state-of-the-art Neu-IR models. One exciting property about our work is that it segments a document roughly by topics. We think these topical segments are more meaningful units than segments of fixed lengths and natural paragraphs (which do not necessarily respect topical boundaries). Moreover, with multiple different sized kernels, a hierarchical modeling of document structure is practically enabled and probably contributes to the effectiveness of our approach.

Acknowledgement

This research was supported by NSF CAREER Award IIS-145374. Any opinions, findings, conclusions, or recommendations expressed in this paper are of the authors, and do not necessarily reflect those of the sponsor.

References

Al-akashi, F. H., and Inkpen, D. 2012. Query-structure based web page indexing. In *TREC'12*.

Badjatiya, P.; Kurisinkel, L. J.; Gupta, M.; and Varma, V. 2018. Attention-based neural text segmentation. In *ECIR'18*.

Beeferman, D.; Berger, A.; and Lafferty, J. 1999. Statistical models for text segmentation. *Machine learning* 34(1-3):177–210.

Boytsov, L., and Belova, A. 2011. Evaluating learning-to-rank methods in the web track adhoc task. In *TREC'11*.

Burges, C.; Shaked, T.; Renshaw, E.; Lazier, A.; Deeds, M.; Hamilton, N.; and Hullender, G. 2005. Learning to rank using gradient descent. In *ICML'05*.

Byrd, D. 1999. A scrollbar-based visualization for document navigation. In *DL'99*.

Chapelle, O.; Metlzer, D.; Zhang, Y.; and Grinspan, P. 2009. Expected reciprocal rank for graded relevance. In *CIKM'09*.

Clarke, C. L. A.; Craswell, N.; and Voorhees, E. M. 2012. Overview of the trec 2012 web track. In *TREC'12*.

Dinçer, B. T.; Kocabas, I.; and Karaoglan, B. 2010. Irra at trec 2010: Index term weighting by divergence from independence model. In *TREC'10*.

Elsayed, T.; Asadi, N.; Wang, L.; Lin, J. J.; and Metzler, D. 2010. Umd and usc/isi: Trec 2010 web track experiments with ivory. In *TREC'10*.

Fan, Y.; Guo, J.; Lan, Y.; Xu, J.; Pang, L.; and Cheng, X. 2017. Learning visual features from snapshots for web search. In *CIKM'17*.

Fan, Y.; Guo, J.; Lan, Y.; Xu, J.; Zhai, C.; and Cheng, X. 2018. Modeling diverse relevance patterns in ad-hoc retrieval. In *SIGIR'18*.

Guo, J.; Fan, Y.; Ai, Q.; and Croft, W. B. 2016. A deep relevance matching model for ad-hoc retrieval. In *CIKM'16*.

Hearst, M. A. 1994. Multi-paragraph segmentation expository text. In *ACL'94*.

Hearst, M. A. 1995. Tilebars: visualization of term distribution information in full text information access. In *CHI'95*.

Hearst, M. 2009. *Search user interfaces*. Cambridge University Press.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Hornbæk, K., and Frøkjær, E. 2001. Reading of electronic documents: the usability of linear, fisheye, and overview+ detail interfaces. In *CHI'01*.

Huang, P.-S.; He, X.; Gao, J.; Deng, L.; Acero, A.; and Heck, L. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM'13*.

Hui, K.; Yates, A.; Berberich, K.; and de Melo, G. 2017. Pacrr: A position-aware neural ir model for relevance matching. In *EMNLP'17*.

Järvelin, K., and Kekäläinen, J. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)* 20(4):422–446.

Kazantseva, A., and Szpakowicz, S. 2011. Linear text segmentation using affinity propagation. In *EMNLP'11*.

Kim, Y. 2014. Convolutional neural networks for sentence classification. In *EMNLP'14*.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. *ICLR'15*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS'12*.

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

Mann, W. C., and Thompson, S. A. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text* 8(3):243–281.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS'13*.

Misra, H.; Yvon, F.; Jose, J. M.; and Cappe, O. 2009. Text segmentation via topic modeling: an analytical study. In *CIKM'09*.

Mitra, B.; Diaz, F.; and Craswell, N. 2017. Learning to match using local and distributed representations of text for web search. In *WWW'17*.

Nguyen, G.-H.; Soulier, L.; Tamine, L.; and Bricon-Souf, N. 2017. Dsrim: A deep neural information retrieval model enhanced by a knowledge resource driven representation of documents. In *ICTIR'17*.

Pang, L.; Lan, Y.; Guo, J.; Xu, J.; and Cheng, X. 2016a. A study of matchpyramid models on ad-hoc retrieval. *Neu-IR'16 SIGIR Workshop on Neural Information Retrieval*.

Pang, L.; Lan, Y.; Guo, J.; Xu, J.; Wan, S.; and Cheng, X. 2016b. Text matching as image recognition. In *AAAI'16*.

Pang, L.; Lan, Y.; Guo, J.; Xu, J.; Xu, J.; and Cheng, X. 2017. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *CIKM'17*.

Qin, T., and Liu, T. 2013. Introducing LETOR 4.0 datasets. *CoRR* abs/1306.2597.

Robertson, S.; Zaragoza, H.; et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval* 3(4):333–389.

Shen, Y.; He, X.; Gao, J.; Deng, L.; and Mesnil, G. 2014. Learning semantic representations using convolutional neural networks for web search. In *WWW'14*.

Skorochod'ko, E. F. 1971. Adaptive method of automatic abstracting and indexing. In Freiman, C. V., ed., *Proceedings of the IFIP Congress 71*, volume 2, 1179–1182.

Szeliski, R. 2010. *Computer vision: algorithms and applications*. Springer Science & Business Media.

Xiong, C.; Dai, Z.; Callan, J.; Liu, Z.; and Power, R. 2017. End-to-end neural ad-hoc ranking with kernel pooling. In *SIGIR'17*.

Zhai, C., and Lafferty, J. 2017. A study of smoothing methods for language models applied to ad hoc information retrieval. In *ACM SIGIR Forum*, volume 51, 268–276.