Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval

Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, Rabab Ward

***Abstract*—This paper develops a model that addresses sentence embedding, a hot topic in current natural language processing research, using recurrent neural networks (RNN) with Long Short-Term Memory (LSTM) cells. The proposed LSTM-RNN model sequentially takes each word in a sentence, extracts its information, and embeds it into a semantic vector. Due to its ability to capture long term memory, the LSTM-RNN accumulates increasingly richer information as it goes through the sentence, and when it reaches the last word, the hidden layer of the network provides a semantic representation of the whole sentence. In this paper, the LSTM-RNN is trained in a *weakly supervised* manner on user click-through data logged by a commercial web search engine. Visualization and analysis are performed to understand how the embedding process works. The model is found to automatically attenuate the unimportant words and detects the salient keywords in the sentence. Furthermore, these detected keywords are found to automatically activate different cells of the LSTM-RNN, where words belonging to a similar topic activate the same cell. As a semantic representation of the sentence, the embedding vector can be used in many different applications. These automatic keyword detection and topic allocation abilities enabled by the LSTM-RNN allow the network to perform document retrieval, a difficult language processing task, where the similarity between the query and documents can be measured by the distance between their corresponding sentence embedding vectors computed by the LSTM-RNN. On a web search task, the LSTM-RNN embedding is shown to significantly outperform several existing state of the art methods. We emphasize that the proposed model generates sentence embedding vectors that are specially useful for web document retrieval tasks. A comparison with a well known general sentence embedding method, the Paragraph Vector, is performed. The results show that the proposed method in this paper significantly outperforms it for web document retrieval task.**

***Index Terms*—Deep Learning, Long Short-Term Memory, Sentence Embedding.**

I. Introduction

H. Palangi and R. Ward are with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, V6T 1Z4 Canada (e-mail: {hamidp,rababw}@ece.ubc.ca)

L. Deng, Y. Shen, J.Gao, X. He, J. Chen and X. Song are with Microsoft Research, Redmond, WA 98052 USA (e-mail: {deng,jfgao,xiaohe,yeshen,jianshuc,xinson}@microsoft.com)

LEARNING a good representation (or features) of input data is an important task in machine learning. In text and language processing, one such problem is learning of an embedding vector for a sentence; that is, to train a model that can automatically transform a sentence to a vector that encodes the semantic meaning of the sentence. While word embedding is learned using a loss function defined on word pairs, sentence embedding is learned using a loss function defined on sentence pairs. In the sentence embedding usually the relationship among words in the sentence, i.e., the context information, is taken into consideration. Therefore, sentence embedding is more suitable for tasks that require computing semantic similarities between text strings. By mapping texts into a unified semantic representation, the embedding vector can be further used for different language processing applications, such as machine translation [1], sentiment analysis [2], and information retrieval [3]. In machine translation, the recurrent neural networks (RNN) with Long Short-Term Memory (LSTM) cells, or the LSTM-RNN, is used to encode an English sentence into a vector, which contains the semantic meaning of the input sentence, and then another LSTM-RNN is used to generate a French (or another target language) sentence from the vector. The model is trained to best predict the output sentence. In [2], a paragraph vector is learned in an unsupervised manner as a distributed representation of sentences and documents, which are then used for sentiment analysis. Sentence embedding can also be applied to information retrieval, where the contextual information are properly represented by the vectors in the same space for fuzzy text matching [3].

In this paper, we propose to use an RNN to sequentially accept each word in a sentence and recurrently map it into a latent space together with the historical information. As the RNN reaches the last word in the sentence, the hidden activations form a natural embedding vector for the contextual information of the sentence. We further incorporate the LSTM cells into the RNN model (i.e. the LSTM-RNN) to address the difficulty of learning long term memory in RNN. The learning of such a model

is performed in a *weakly supervised* manner on the click-through data logged by a commercial web search engine. Although manually labelled data are insufficient in machine learning, logged data with limited feedback signals are massively available due to the widely used commercial web search engines. Limited feedback information such as click-through data provides a weak supervision signal that indicates the semantic similarity between the text on the query side and the clicked text on the document side. To exploit such a signal, the objective of our training is to maximize the similarity between the two vectors mapped by the LSTM-RNN from the query and the clicked document, respectively. Consequently, the learned embedding vectors of the query and clicked document are specifically useful for web document retrieval task.

An important contribution of this paper is to analyse the embedding process of the LSTM-RNN by visualizing the internal activation behaviours in response to different text inputs. We show that the embedding process of the learned LSTM-RNN effectively detects the keywords, while attenuating less important words, in the sentence automatically by switching on and off the gates within the LSTM-RNN cells. We further show that different cells in the learned model indeed correspond to different topics, and the keywords associated with a similar topic activate the same cell unit in the model. As the LSTM-RNN reads to the end of the sentence, the topic activation accumulates and the hidden vector at the last word encodes the rich contextual information of the entire sentence. For this reason, a natural application of sentence embedding is web search ranking, in which the embedding vector from the query can be used to match the embedding vectors of the candidate documents according to the maximum cosine similarity rule. Evaluated on a real web document ranking task, our proposed method significantly outperforms many of the existing state of the art methods in NDCG scores. Please note that when we refer to document in the paper we mean the title (headline) of the document.

II. Related Work

Inspired by the word embedding method [4], [5], the authors in [2] proposed an unsupervised learning method to learn a paragraph vector as a distributed representation of sentences and documents, which are then used for sentiment analysis with superior performance. However, the model is not designed to capture the fine-grained sentence structure. In [6], an unsupervised sentence embedding method is proposed with great performance on large corpus of contiguous text corpus, e.g., the BookCorpus [7]. The main idea is to encode the sentence $s(t)$ and then decode previous and next sentences, i.e., $s(t-1)$ and $s(t+1)$, using two separate decoders. The encoder and decoders are RNNs with Gated Recurrent Unit (GRU) [8]. However, this sentence embedding method is not designed for document retrieval task having a supervision among queries and clicked and unclicked documents. In [9], a Semi-Supervised Recursive Autoencoder (RAE) is proposed and used for sentiment prediction. Similar to our proposed method, it does not need any language specific sentiment parsers. A greedy approximation method is proposed to construct a tree structure for the input sentence. It assigns a vector per word. It can become practically problematic for large vocabularies. It also works both on unlabeled data and supervised sentiment data.

Similar to the recurrent models in this paper, The DSSM [3] and CLSM [10] models, developed for information retrieval, can also be interpreted as sentence embedding methods. However, DSSM treats the input sentence as a bag-of-words and does not model word dependencies explicitly. CLSM treats a sentence as a bag of n-grams, where n is defined by a window, and can capture *local* word dependencies. Then a Max-pooling layer is used to form a global feature vector. Methods in [11] are also convolutional based networks for Natural Language Processing (NLP). These models, by design, cannot capture long distance dependencies, i.e., dependencies among words belonging to non-overlapping n-grams. In [12] a Dynamic Convolutional Neural Network (DCNN) is proposed for sentence embedding. Similar to CLSM, DCNN does not rely on a parse tree and is easily applicable to any language. However, different from CLSM where a regular max-pooling is used, in DCNN a dynamic k-max-pooling is used. This means that instead of just keeping the largest entries among word vectors in one vector, k largest entries are kept in k different vectors. DCNN has shown good performance in sentiment prediction and question type classification tasks. In [13], a convolutional neural network architecture is proposed for sentence matching. It has shown great performance in several matching tasks. In [14], a Bilingually-constrained Recursive Auto-encoders (BRAE) is proposed to create semantic vector representation for phrases. Through experiments it is shown that the proposed method has great performance in two end-to-end SMT tasks.

Long short-term memory networks were developed in [15] to address the difficulty of capturing long term memory in RNN. It has been successfully applied to speech recognition, which achieves state-of-art performance [16], [17]. In text analysis, LSTM-RNN treats a sentence as a sequence of words with internal structures, i.e., word dependencies. It encodes a semantic vector of a sentence incrementally which differs from DSSM and CLSM. The encoding process is performed left-to-right, word-by-word. At each time step, a new word is encoded

into the semantic vector, and the word dependencies embedded in the vector are "updated". When the process reaches the end of the sentence, the semantic vector has embedded all the words and their dependencies, hence, can be viewed as a feature vector representation of the whole sentence. In the machine translation work [1], an input English sentence is converted into a vector representation using LSTM-RNN, and then another LSTM-RNN is used to generate an output French sentence. The model is trained to maximize the probability of predicting the correct output sentence. In [18], there are two main composition models, ADD model that is bag of words and BI model that is a summation over bi-gram pairs plus a non-linearity. In our proposed model, instead of simple summation, we have used LSTM model with letter tri-grams which keeps valuable information over long intervals (for long sentences) and throws away useless information. In [19], an encoder-decoder approach is proposed to jointly learn to align and translate sentences from English to French using RNNs. The concept of "attention" in the decoder, discussed in this paper, is closely related to how our proposed model extracts keywords in the document side. For further explanations please see section V-A2. In [20] a set of visualizations are presented for RNNs with and without LSTM cells and GRUs. Different from our work where the target task is sentence embedding for document retrieval, the target tasks in [20] were character level sequence modelling for text characters and source codes. Interesting observations about interpretability of some LSTM cells and statistics of gates activations are presented. In section V-A we show that some of the results of our visualization are consistent with the observations reported in [20]. We also present more detailed visualization specific to the document retrieval task using click-through data. We also present visualizations about how our proposed model can be used for keyword detection.

Different from the aforementioned studies, the method developed in this paper trains the model so that sentences that are paraphrase of each other are close in their semantic embedding vectors — see the description in Sec. IV further ahead. Another reason that LSTM-RNN is particularly effective for sentence embedding, is its robustness to noise. For example, in the web document ranking task, the noise comes from two sources: (i) Not every word in query / document is equally important, and we only want to "remember" salient words using the limited "memory". (ii) A word or phrase that is important to a document may not be relevant to a given query, and we only want to "remember" related words that are useful to compute the relevance of the document for a given query. We will illustrate robustness of LSTM-RNN in this paper. The structure of LSTM-RNN will also circumvent the serious limitation of using a fixed window size in CLSM. Our experiments show that this difference leads to significantly better results in web document retrieval task. Furthermore, it has other advantages. It allows us to capture keywords and key topics effectively. The models in this paper also do not need the extra max-pooling layer, as required by the CLSM, to capture global contextual information and they do so more effectively.

III. Sentence Embedding Using RNNs with and without LSTM Cells

In this section, we introduce the model of recurrent neural networks and its long short-term memory version for learning the sentence embedding vectors. We start with the basic RNN and then proceed to LSTM-RNN.

A. The basic version of RNN

The RNN is a type of deep neural networks that are "deep" in temporal dimension and it has been used extensively in time sequence modelling [21], [22], [23], [24], [25], [26], [27], [28], [29]. The main idea of using RNN for sentence embedding is to find a dense and low dimensional semantic representation by sequentially and recurrently processing each word in a sentence and mapping it into a low dimensional vector. In this model, the global contextual features of the whole text will be in the semantic representation of the last word in the text sequence — see Figure 1, where $\mathbf{x}(t)$ is the t-th word, coded as a 1-hot vector, $\mathbf{W}_h$ is a fixed hashing operator similar to the one used in [3] that converts the word vector to a letter tri-gram vector, $\mathbf{W}$ is the input weight matrix, $\mathbf{W}_{rec}$ is the recurrent weight matrix, $\mathbf{y}(t)$ is the hidden activation vector of the RNN, which can be used as a semantic representation of the t-th word, and $\mathbf{y}(t)$ associated to the last word $\mathbf{x}(m)$ is the semantic representation vector of the entire sentence. Note that this is very different from the approach in [3] where the bag-of-words representation is used for the whole text and no context information is used. This is also different from [10] where the sliding window of a fixed size (akin to an FIR filter) is used to capture local features and a max-pooling layer on the top to capture global features. In the RNN there is neither a fixed-sized window nor a max-pooling layer; rather the recurrence is used to capture the context information in the sequence (akin to an IIR filter).

The mathematical formulation of the above RNN model for sentence embedding can be expressed as

$$\begin{aligned} \mathbf{l}(t) &= \mathbf{W}_h \mathbf{x}(t) \\ \mathbf{y}(t) &= f(\mathbf{W}\mathbf{l}(t) + \mathbf{W}_{\mathrm{rec}}\mathbf{y}(t-1) + \mathbf{b}) \end{aligned} \tag{1}$$

where $\mathbf{W}$ and $\mathbf{W}_{\mathrm{rec}}$ are the input and recurrent matrices to be learned, $\mathbf{W}_h$ is a fixed word hashing operator, $\mathbf{b}$

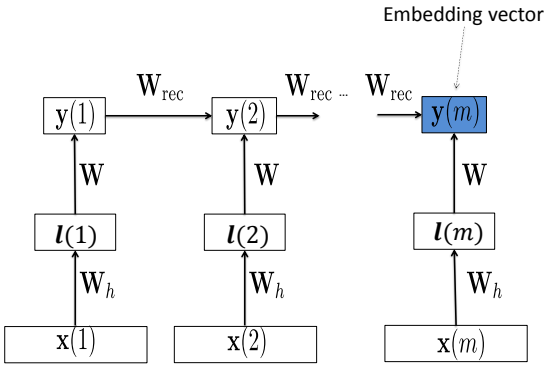


Fig. 1. The basic architecture of the RNN for sentence embedding, where temporal recurrence is used to model the contextual information across words in the text string. The hidden activation vector corresponding to the last word is the sentence embedding vector (blue).

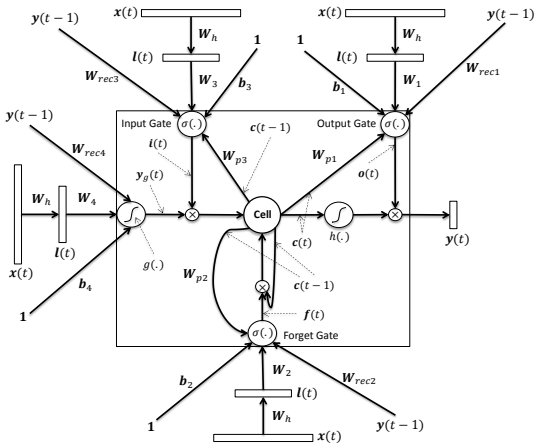


Fig. 2. The basic LSTM architecture used for sentence embedding

is the bias vector and $f(\cdot)$ is assumed to be $\tanh(\cdot)$. Note that the architecture proposed here for sentence embedding is slightly different from traditional RNN in that there is a word hashing layer that convert the high dimensional input into a relatively lower dimensional letter tri-gram representation. There is also no per word supervision during training, instead, the whole sentence has a label. This is explained in more detail in section IV.

B. The RNN with LSTM cells

Although RNN performs the transformation from the sentence to a vector in a principled manner, it is generally difficult to learn the long term dependency within the sequence due to vanishing gradients problem. One of the effective solutions for this problem in RNNs is using memory cells instead of neurons originally proposed in [15] as Long Short-Term Memory (LSTM) and completed in [30] and [31] by adding forget gate and peephole connections to the architecture.

We use the architecture of LSTM illustrated in Fig. 2 for the proposed sentence embedding method. In this figure, $\mathbf{i}(t)$, $\mathbf{f}(t)$, $\mathbf{o}(t)$, $\mathbf{c}(t)$ are input gate, forget gate, output gate and cell state vector respectively, $\mathbf{W}_{p1}$, $\mathbf{W}_{p2}$ and $\mathbf{W}_{p3}$ are peephole connections, $\mathbf{W}_i$, $\mathbf{W}_{reci}$ and $\mathbf{b}_i$, $i = 1, 2, 3, 4$ are input connections, recurrent connections and bias values, respectively, $g(\cdot)$ and $h(\cdot)$ are $\tanh(\cdot)$ function and $\sigma(\cdot)$ is the sigmoid function. We use this architecture to find $\mathbf{y}$ for each word, then use the $\mathbf{y}(m)$ corresponding to the last word in the sentence as the semantic vector for the entire sentence.

Considering Fig. 2, the forward pass for LSTM-RNN model is as follows:

$$
\begin{aligned}
\mathbf{y}_g(t) &= g(\mathbf{W}_4\mathbf{l}(t) + \mathbf{W}_{\text{rec4}}\mathbf{y}(t-1) + \mathbf{b}_4) \\
\mathbf{i}(t) &= \sigma(\mathbf{W}_3\mathbf{l}(t) + \mathbf{W}_{\text{rec3}}\mathbf{y}(t-1) + \mathbf{W}_{p3}\mathbf{c}(t-1) + \mathbf{b}_3) \\
\mathbf{f}(t) &= \sigma(\mathbf{W}_2\mathbf{l}(t) + \mathbf{W}_{\text{rec2}}\mathbf{y}(t-1) + \mathbf{W}_{p2}\mathbf{c}(t-1) + \mathbf{b}_2) \\
\mathbf{c}(t) &= \mathbf{f}(t) \circ \mathbf{c}(t-1) + \mathbf{i}(t) \circ \mathbf{y}_g(t) \\
\mathbf{o}(t) &= \sigma(\mathbf{W}_1\mathbf{l}(t) + \mathbf{W}_{\text{rec1}}\mathbf{y}(t-1) + \mathbf{W}_{p1}\mathbf{c}(t) + \mathbf{b}_1) \\
\mathbf{y}(t) &= \mathbf{o}(t) \circ h(\mathbf{c}(t))
\end{aligned}
\tag{2}
$$

where $\circ$ denotes Hadamard (element-wise) product. A diagram of the proposed model with more details is presented in section VI of Supplementary Materials.

IV. Learning Method

To learn a good semantic representation of the input sentence, our objective is *to make the embedding vectors for sentences of similar meaning as close as possible, and meanwhile, to make sentences of different meanings as far apart as possible*. This is challenging in practice since it is hard to collect a large amount of manually labelled data that give the semantic similarity signal between different sentences. Nevertheless, the widely used commercial web search engine is able to log massive amount of data with some limited user feedback signals. For example, given a particular query, the click-through information about the user-clicked document among many candidates is usually recorded and can be used as a weak (binary) supervision signal to indicate the semantic similarity between two sentences (on the query side and the document side). In this section, we explain how to leverage such a weak supervision signal to learn a sentence embedding vector that achieves the aforementioned training objective. Please also note that above objective to make sentences with similar meaning as close as possible is similar to machine translation

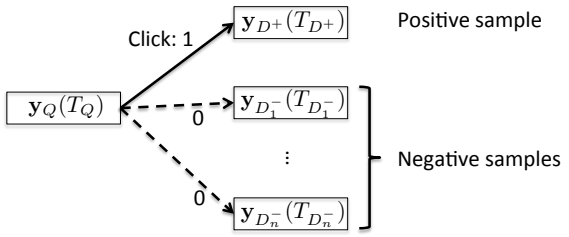


Fig. 3. The click-through signal can be used as a (binary) indication of the semantic similarity between the sentence on the query side and the sentence on the document side. The negative samples are randomly sampled from the training data.

tasks where two sentences belong to two different languages with similar meanings and we want to make their semantic representation as close as possible.

We now describe how to train the model to achieve the above objective using the click-through data logged by a commercial search engine. For a complete description of the click-through data please refer to section 2 in [32]. To begin with, we adopt the cosine similarity between the semantic vectors of two sentences as a measure for their similarity:

$$R(Q,D) = \frac{\mathbf{y}_Q(T_Q)^T \mathbf{y}_D(T_D)}{\|\mathbf{y}_Q(T_Q)\| \cdot \|\mathbf{y}_D(T_D)\|} \tag{3}$$

where T_Q and T_D are the lengths of the sentence Q and sentence D, respectively. In the context of training over click-through data, we will use Q and D to denote "query" and "document", respectively. In Figure 3, we show the sentence embedding vectors corresponding to the query, $\mathbf{y}_Q(T_Q)$, and all the documents, $\{\mathbf{y}_{D^+}(T_{D^+}), \mathbf{y}_{D_1^-}(T_{D_1^-}), \ldots, \mathbf{y}_{D_n^-}(T_{D_n^-})\}$, where the subscript D^+ denotes the (clicked) positive sample among the documents, and the subscript D_j^- denotes the j-th (un-clicked) negative sample. All these embedding vectors are generated by feeding the sentences into the RNN or LSTM-RNN model described in Sec. III and take the $\mathbf{y}$ corresponding to the last word — see the blue box in Figure 1.

We want to maximize the likelihood of the clicked document given query, which can be formulated as the following optimization problem:

$$L(\mathbf{\Lambda}) = \min_{\mathbf{\Lambda}} \left\{ -\log \prod_{r=1}^{N} P(D_r^+|Q_r) \right\} = \min_{\mathbf{\Lambda}} \sum_{r=1}^{N} l_r(\mathbf{\Lambda}) \tag{4}$$

where $\mathbf{\Lambda}$ denotes the collection of the model parameters; in regular RNN case, it includes $\mathbf{W}_{rec}$ and $\mathbf{W}$ in Figure 1, and in LSTM-RNN case, it includes $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$, $\mathbf{W}_4$, $\mathbf{W}_{rec1}$, $\mathbf{W}_{rec2}$, $\mathbf{W}_{rec3}$, $\mathbf{W}_{rec4}$, $\mathbf{W}_{p1}$, $\mathbf{W}_{p2}$, $\mathbf{W}_{p3}$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ and $\mathbf{b}_4$ in Figure 2. D_r^+ is the clicked document for r-th query, $P(D_r^+|Q_r)$ is the probability of clicked document given the r-th query, N is number of query / clicked-document pairs in the corpus and

$$\begin{aligned} l_r(\mathbf{\Lambda}) &= -\log \left(\frac{e^{\gamma R(Q_r, D_r^+)}}{e^{\gamma R(Q_r, D_r^+)} + \sum_{i=j}^{n} e^{\gamma R(Q_r, D_{r,j}^-)}} \right) \\ &= \log \left(1 + \sum_{j=1}^{n} e^{-\gamma \cdot \Delta_{r,j}} \right) \end{aligned} \tag{5}$$

where $\Delta_{r,j} = R(Q_r, D_r^+) - R(Q_r, D_{r,j}^-)$, $R(\cdot,\cdot)$ was defined earlier in (3), $D_{r,j}^-$ is the j-th negative candidate document for r-th query and n denotes the number of negative samples used during training.

The expression in (5) is a logistic loss over $\Delta_{r,j}$. It upper-bounds the pairwise accuracy, i.e., the 0 - 1 loss. Since the similarity measure is the cosine function, $\Delta_{r,j} \in [-2, 2]$. To have a larger range for $\Delta_{r,j}$, we use γ for scaling. It helps to penalize the prediction error more. Its value is set empirically by experiments on a held out dataset.

To train the RNN and LSTM-RNN, we use Back Propagation Through Time (BPTT). The update equations for parameter $\mathbf{\Lambda}$ at epoch k are as follows:

$$\begin{aligned} \triangle \mathbf{\Lambda}_k &= \mathbf{\Lambda}_k - \mathbf{\Lambda}_{k-1} \\ \triangle \mathbf{\Lambda}_k &= \mu_{k-1} \triangle \mathbf{\Lambda}_{k-1} - \epsilon_{k-1} \nabla L(\mathbf{\Lambda}_{k-1} + \mu_{k-1} \triangle \mathbf{\Lambda}_{k-1}) \end{aligned} \tag{6}$$

where $\nabla L(\cdot)$ is the gradient of the cost function in (4), ϵ is the learning rate and μ_k is a momentum parameter determined by the scheduling scheme used for training. Above equations are equivalent to Nesterov method in [33]. To see why, please refer to appendix A.1 of [34] where Nesterov method is derived as a momentum method. The gradient of the cost function, $\nabla L(\mathbf{\Lambda})$, is:

$$\nabla L(\mathbf{\Lambda}) = \underbrace{-\sum_{r=1}^{N} \sum_{j=1}^{n} \sum_{\tau=0}^{T} \alpha_{r,j} \frac{\partial \Delta_{r,j,\tau}}{\partial \mathbf{\Lambda}}}_{\text{one large update}} \tag{7}$$

where T is the number of time steps that we unfold the network over time and

$$\alpha_{r,j} = \frac{-\gamma e^{-\gamma \Delta_{r,j}}}{1 + \sum_{j=1}^{n} e^{-\gamma \Delta_{r,j}}}. \tag{8}$$

$\frac{\partial \Delta_{r,j,\tau}}{\partial \mathbf{\Lambda}}$ in (7) and error signals for different parameters of RNN and LSTM-RNN that are necessary for training are presented in Appendix A. Full derivation of gradients in both models is presented in section III of supplementary materials.

To accelerate training by parallelization, we use mini-batch training and one large update instead of incremental updates during back propagation through time. To resolve the gradient explosion problem we use gradient

Algorithm 1 Training LSTM-RNN for Sentence Embedding

Inputs: Fixed step size "ϵ", Scheduling for "μ", Gradient clip threshold "th_G", Maximum number of Epochs "$nEpoch$", Total number of query / clicked-document pairs "N", Total number of un-clicked (negative) documents for a given query "n", Maximum sequence length for truncated BPTT "T".
Outputs: Two trained models, one in query side "$\mathbf{\Lambda}_Q$", one in document side "$\mathbf{\Lambda}_D$".
Initialization: Set all parameters in $\mathbf{\Lambda}_Q$ and $\mathbf{\Lambda}_D$ to small random numbers, $i = 0$, $k = 1$.
procedure LSTM-RNN($\mathbf{\Lambda}_Q$,$\mathbf{\Lambda}_D$)
  while $i \leq nEpoch$ **do**
    for "first minibatch" $\rightarrow$ "last minibatch" **do**
      $r \leftarrow 1$
      while $r \leq N$ **do**
        for $j = 1 \rightarrow n$ **do**
          Compute $\alpha_{r,j}$ ▷ use (8)
          Compute $\sum_{\tau=0}^{T} \alpha_{r,j} \frac{\partial \Delta_{r,j,\tau}}{\partial \mathbf{\Lambda}_{k,Q}}$ ▷ use (14) to (44) in appendix A
          Compute $\sum_{\tau=0}^{T} \alpha_{r,j} \frac{\partial \Delta_{r,j,\tau}}{\partial \mathbf{\Lambda}_{k,D}}$ ▷ use (14) to (44) in appendix A
          sum above terms for Q and D over j
        end for
        sum above terms for Q and D over r
        $r \leftarrow r + 1$
      end while
      Compute $\nabla L(\mathbf{\Lambda}_{k,Q})$ ▷ use (7)
      Compute $\nabla L(\mathbf{\Lambda}_{k,D})$ ▷ use (7)
      if $\|\nabla L(\mathbf{\Lambda}_{k,Q})\| > th_G$ **then**
        $\nabla L(\mathbf{\Lambda}_{k,Q}) \leftarrow th_G \cdot \frac{\nabla L(\mathbf{\Lambda}_{k,Q})}{\|\nabla L(\mathbf{\Lambda}_{k,Q})\|}$
      end if
      if $\|\nabla L(\mathbf{\Lambda}_{k,D})\| > th_G$ **then**
        $\nabla L(\mathbf{\Lambda}_{k,D}) \leftarrow th_G \cdot \frac{\nabla L(\mathbf{\Lambda}_{k,D})}{\|\nabla L(\mathbf{\Lambda}_{k,D})\|}$
      end if
      Compute $\triangle \mathbf{\Lambda}_{k,Q}$ ▷ use (6)
      Compute $\triangle \mathbf{\Lambda}_{k,D}$ ▷ use (6)
      Update: $\mathbf{\Lambda}_{k,Q} \leftarrow \triangle \mathbf{\Lambda}_{k,Q} + \mathbf{\Lambda}_{k-1,Q}$
      Update: $\mathbf{\Lambda}_{k,D} \leftarrow \triangle \mathbf{\Lambda}_{k,D} + \mathbf{\Lambda}_{k-1,D}$
      $k \leftarrow k + 1$
    end for
    $i \leftarrow i + 1$
  end while
end procedure

re-normalization method described in [35], [24]. To accelerate the convergence, we use Nesterov method [33] and found it effective in training both RNN and LSTM-RNN for sentence embedding.

We have used a simple yet effective scheduling for μ_k for both RNN and LSTM-RNN models, in the first and last 2% of all parameter updates $\mu_k = 0.9$ and for the other 96% of all parameter updates $\mu_k = 0.995$. We have used a fixed step size for training RNN and a fixed step size for training LSTM-RNN.

A summary of training method for LSTM-RNN is presented in Algorithm 1.

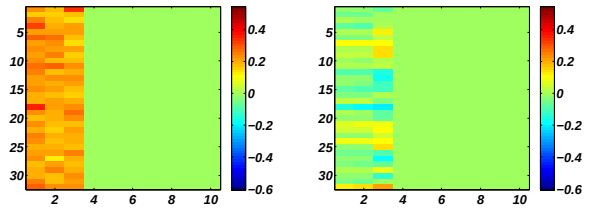
V. Analysis of the Sentence Embedding Process and Performance Evaluation

To understand how the LSTM-RNN performs sentence embedding, we use visualization tools to analyze the semantic vectors generated by our model. We would like to answer the following questions: (i) How are word dependencies and context information captured? (ii) How does LSTM-RNN attenuate unimportant information and detect critical information from the input sentence? Or, how are the keywords embedded into the semantic vector? (iii) How are the global topics identified by LSTM-RNN?

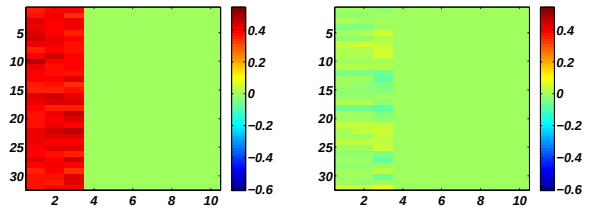
To answer these questions, we train the RNN with and without LSTM cells on the click-through dataset which are logged by a commercial web search engine. The training method has been described in Sec. IV. Description of the corpus is as follows. The training set includes 200,000 positive query / document pairs where only the clicked signal is used as a weak supervision for training LSTM. The relevance judgement set (test set) is constructed as follows. First, the queries are sampled from a year of search engine logs. Adult, spam, and bot queries are all removed. Queries are de-duped so that only unique queries remain. To reflex a natural query distribution, we do not try to control the quality of these queries. For example, in our query sets, there are around 20% misspelled queries, and around 20% navigational queries and 10% transactional queries, etc. Second, for each query, we collect Web documents to be judged by issuing the query to several popular search engines (e.g., Google, Bing) and fetching top-10 retrieval results from each. Finally, the query-document pairs are judged by a group of well-trained assessors. In this study all the queries are preprocessed as follows. The text is white-space tokenized and lower-cased, numbers are retained, and no stemming/inflection treatment is performed. Unless stated otherwise, in the experiments we used 4 negative samples, i.e., $n = 4$ in Fig. 3.

We now proceed to perform a comprehensive analysis by visualizing the trained RNN and LSTM-RNN models. In particular, we will visualize the on-and-off behaviors of the input gates, output gates, cell states, and the semantic vectors in LSTM-RNN model, which reveals how the model extracts useful information from the input sentence and embeds it properly into the semantic vector according to the topic information.

Although giving the full learning formula for all the model parameters in the previous section, we will remove the peephole connections and the forget gate from the LSTM-RNN model in the current task. This is because the length of each sequence, i.e., the number of words in a query or a document, is known in advance, and we set the state of each cell to zero in the beginning of a new sequence. Therefore, forget gates are not a great help here. Also, as long as the order of words is kept, the precise timing in the sequence is not of great concern. Therefore, peephole connections are not that important as well. Removing peephole connections and forget gate will also reduce the amount of training time, since a smaller number of parameters need to be learned.

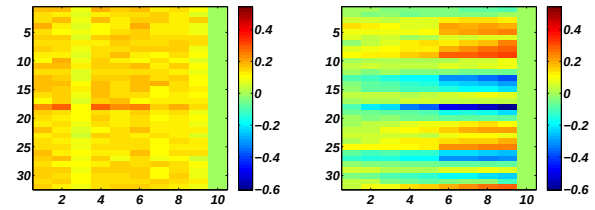


(a) $\mathbf{i}(t)$ (b) $\mathbf{c}(t)$

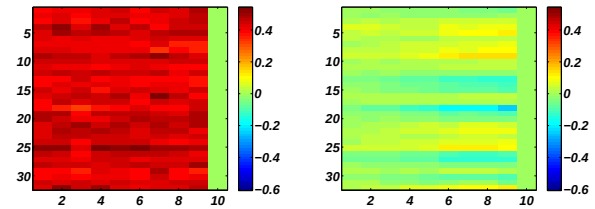


(c) $\mathbf{o}(t)$ (d) $\mathbf{y}(t)$

Fig. 4. Query: "*hotels in shanghai*". Since the sentence ends at the third word, all the values to the right of it are zero (green color).



(a) $\mathbf{i}(t)$ (b) $\mathbf{c}(t)$



(c) $\mathbf{o}(t)$ (d) $\mathbf{y}(t)$

Fig. 5. Document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*". Since the sentence ends at the ninth word, all the values to the right of it are zero (green color).

A. Analysis

In this section we would like to examine how the information in the input sentence is sequentially extracted and embedded into the semantic vector over time by the LSTM-RNN model.

1) Attenuating Unimportant Information: First, we examine the evolution of the semantic vector and how unimportant words are attenuated. Specifically, we feed the following input sentences from the test dataset into the trained LSTM-RNN model:

- Query: "*hotels in shanghai*"
- Document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

Activations of input gate, output gate, cell state and the embedding vector for each cell for query and document are shown in Fig. 4 and Fig. 5, respectively. The vertical axis is the cell index from 1 to 32, and the horizontal axis is the word index from 1 to 10 numbered from left to right in a sequence of words and color codes show activation values. From Figs.4–5, we make the following observations:

- Semantic representation $\mathbf{y}(t)$ and cell states $\mathbf{c}(t)$ are evolving over time. Valuable context information is gradually absorbed into $\mathbf{c}(t)$ and $\mathbf{y}(t)$, so that the information in these two vectors becomes richer over time, and the semantic information of the entire input sentence is embedded into vector $\mathbf{y}(t)$, which is obtained by applying output gates to the cell states $\mathbf{c}(t)$.
- The input gates evolve in such a way that it attenuates the unimportant information and detects the important information from the input sentence. For example, in Fig. 5(a), most of the input gate values corresponding to word 3, word 7 and word 9 have very small values (light green-yellow color)[1], which corresponds to the words "$accommodation$", "$discount$" and "$reservation$", respectively, in the document sentence. Interestingly, input gates reduce the effect of these three words in the final semantic representation, $\mathbf{y}(t)$, such that the semantic similarity between sentences from query and document sides are not affected by these words.

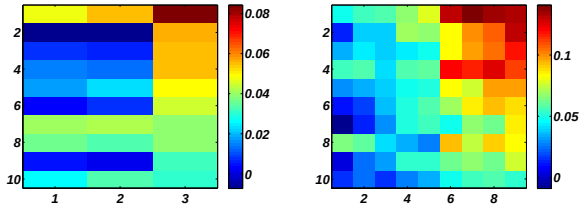
2) Keywords Extraction: In this section, we show how the trained LSTM-RNN extracts the important information, i.e., keywords, from the input sentences. To this end, we backtrack semantic representations, $\mathbf{y}(t)$, over time. We focus on the 10 most active cells in final semantic representation. Whenever there is a large enough change in cell activation value ($\mathbf{y}(t)$), we assume an important keyword has been detected by the model. We illustrate the result using the above example ("*hotels in shanghai*"). The evolution of the 10 most active cells activation, $\mathbf{y}(t)$, over time are shown in Fig. 6 for the query and the document sentences.[2]From Fig. 6, we also observe that different words activate different cells. In Tables I–II, we show the number of cells each word

[1]If this is not clearly visible, please refer to Fig. 1 in section I of supplementary materials. We have adjusted color bar for all figures to have the same range, for this reason the structure might not be clearly visible. More visualization examples could also be found in section IV of Supplementary Materials

[2]Likewise, the vertical axis is the cell index and horizontal axis is the word index in the sentence.

TABLE II
KEY WORDS FOR DOCUMENT: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

	shanghai	*hotels*	*accommodation*	*hotel*	*in*	*shanghai*	*discount*	*and*	*reservation*
Number of assigned cells out of 10 Left to Right	-	**4**	3	**8**	1	**8**	**5**	3	**4**
Number of assigned cells out of 10 Right to Left	**4**	**6**	**5**	**4**	**5**	1	**7**	**5**	-



(a) $\mathbf{y}(t)$ top 10 for query (b) $\mathbf{y}(t)$ top 10 for document

Fig. 6. Activation values, $\mathbf{y}(t)$, of 10 most active cells for Query: "*hotels in shanghai*" and Document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

TABLE I
KEY WORDS FOR QUERY: "*hotels in shanghai*"

Query	*hotels*	*in*	*shanghai*
Number of assigned cells out of 10 Left to Right	-	0	**7**
Number of assigned cells out of 10 Right to Left	**6**	0	-

activates.[3] We used Bidirectional LSTM-RNN to get the results of these tables where in the first row, LSTM-RNN reads sentences from left to right and in the second row it reads sentences from right to left. In these tables we labelled a word as a keyword if more than 40% of top 10 active cells in both directions declare it as keyword. The boldface numbers in the table show that the number of cells assigned to that word is more than 4, i.e., 40% of top 10 active cells. From the tables, we observe that the keywords activate more cells than the unimportant words, meaning that they are selectively embedded into the semantic vector.

3) Topic Allocation: Now, we further show that the trained LSTM-RNN model not only detects the keywords, but also allocates them properly to different cells according to the topics they belong to. To do this, we go through the test dataset using the trained LSTM-RNN model and search for the keywords that are detected by a specific cell. For simplicity, we use the following simple approach: for each given query we look into the keywords that are extracted by the 5 most active cells of LSTM-RNN and list them in Table III. Interestingly, each cell collects keywords of a specific topic. For example, cell 26 in Table III extracts keywords related to the topic "food" and cells 2 and 6 mainly focus on the keywords related to the topic "health".

[3]Note that before presenting the first word of the sequence, activation values are initially zero so that there is always a considerable change in the cell states after presenting the first word. For this reason, we have not indicated the number of cells detecting the first word as a keyword. Moreover, another keyword extraction example can be found in section IV of supplementary materials.

B. Performance Evaluation

1) Web Document Retrieval Task: In this section, we apply the proposed sentence embedding method to an important web document retrieval task for a commercial web search engine. Specifically, the RNN models (with and without LSTM cells) embed the sentences from the query and the document sides into their corresponding semantic vectors, and then compute the cosine similarity between these vectors to measure the semantic similarity between the query and candidate documents.

Experimental results for this task are shown in Table IV using the standard metric mean Normalized Discounted Cumulative Gain (NDCG) [36] (the higher the better) for evaluating the ranking performance of the RNN and LSTM-RNN on a standalone human-rated test dataset. We also trained several strong baselines, such as DSSM [3] and CLSM [10], on the same training dataset and evaluated their performance on the same task. For fair comparison, our proposed RNN and LSTM-RNN models are trained with the same number of parameters as the DSSM and CLSM models (14.4M parameters). Besides, we also include in Table IV two well-known information retrieval (IR) models, BM25 and PLSA, for the sake of benchmarking. The BM25 model uses the bag-of-words representation for queries and documents, which is a state-of-the-art document ranking model based on term matching, widely used as a baseline in IR society. PLSA (Probabilistic Latent Semantic Analysis) is a topic model proposed in [37], which is trained using the Maximum A Posterior estimation [38] on the documents side from the same training dataset. We experimented with a varying number of topics from 100 to 500 for PLSA, which gives similar performance, and we report in Table IV the results of using 500 topics. Results for a language model based method, uni-gram

TABLE III
KEYWORDS ASSIGNED TO EACH CELL OF LSTM-RNN FOR DIFFERENT QUERIES OF TWO TOPICS, "FOOD" AND "HEALTH"

Query	cell 1	cell 2	cell 3	cell 4	cell 5	cell 6	cell 7	cell 8	cell 9	cell 10	cell 11	cell 12	cell 13	cell 14	cell 15	cell 16
al yo yo sauce					yo			sauce			sauce					
atkins diet lasagna								diet								
blender recipes																
cake bakery edinburgh										bakery						
canning corn beef hash					beef, hash											
torre de pizza																
famous desserts							desserts									
fried chicken			chicken				chicken									
smoked turkey recipes																
italian sausage hoagies								sausage								
do you get allergy		allergy														
much pain will after total knee replacement	pain					pain, knee										
how to make whiter teeth													make, teeth		to	
illini community hospital		community, hospital						hospital		community						
implant infection		infection				infection										
introductory psychology		psychology				psychology										
narcotics during pregnancy side effects		pregnancy				pregnancy,effects, during							during			
fight sinus infections						infections										
health insurance high blood pressure		insurance				blood		high, blood								
all antidepressant medications		antidepressant, medications														

Query	cell 17	cell 18	cell 19	cell 20	cell 21	cell 22	cell 23	cell 24	cell 25	cell 26	cell 27	cell 28	cell 29	cell 30	cell 31	cell 32
al yo yo sauce																
atkins diet lasagna							diet							diet		
blender recipes										recipes						
cake bakery edinburgh				bakery						bakery						
canning corn beef hash										corn, beef						
torre de pizza										pizza			pizza			
famous desserts																
fried chicken										chicken						
smoked turkey recipes				turkey						recipes						
italian sausage hoagies	hoagies					sausage				sausage						
do you get allergy																
much pain will after total knee replacement	knee						replacement									
how to make whiter teeth										whiter						
illini community hospital						hospital								hospital		
implant infection									infection							
introductory psychology												psychology				
narcotics during pregnancy side effects																
fight sinus infections	sinus, infections								infections							
health insurance high blood pressure							high, pressure							insurance,high		
all antidepressant medications									antidepressant					medications		

language model (ULM) with Dirichlet smoothing, are also presented in the table.

To compare the performance of the proposed method with general sentence embedding methods in document retrieval task, we also performed experiments using two general sentence embedding methods.

1) In the first experiment, we used the method proposed in [2] that generates embedding vectors known as Paragraph Vectors. It is also known as doc2vec. It maps each word to a vector and then uses the vectors representing all words inside a context window to predict the vector representation of the next word. The main idea in this method is to use an additional paragraph token from previous sentences in the document inside the context window. This paragraph token is mapped to vector space using a different matrix from the one used to map the words. A primary version of this method is known as word2vec proposed in [39]. The only difference is that word2vec does not include the paragraph token.
   To use doc2vec on our dataset, we first trained doc2vec model on both train set (about 200,000 query-document pairs) and test set (about 900,000 query-document pairs). This gives us an embedding vector for every query and document in the dataset. We used the following parameters for training:
   - min-count=1 : minimum number of of words per sentence, sentences with words less than this will be ignored. We set it to 1 to make sure we do not throw away anything.
   - window=5 : fixed window size explained in [2]. We used different window sizes, it resulted in about just 0.4% difference in final NDCG values.
   - size=100 : feature vector dimension. We used 400 as well but did not get significantly different NDCG values.
   - sample=1e-4 : this is the down sampling ratio for the words that are repeated a lot in corpus.
   - negative=5 : the number of noise words, i.e., words used for negative sampling as explained in [2].
   - We used 30 epochs of training. We ran an experiment with 100 epochs but did not observe much difference in the results.
   - We used *gensim* [40] to perform experiments.

   To make sure that a meaningful model is trained, we used the trained doc2vec model to find the most similar words to two sample words in our dataset, e.g., the words "pizza" and "infection". The resulting words and corresponding scores are presented in section V of Supplementary Materials. As it is observed from the resulting words, the trained model is a meaningful model and can recognise semantic similarity.
   Doc2vec also assigns an embedding vector for

each query and document in our test set. We used these embedding vectors to calculate the cosine similarity score between each query-document pair in the test set. We used these scores to calculate NDCG values reported in Table IV for the Doc2Vec model.

Comparing the results of doc2vec model with our proposed method for document retrieval task shows that the proposed method in this paper significantly outperforms doc2vec. One reason for this is that we have used a very general sentence embedding method, doc2vec, for document retrieval task. This experiment shows that it is not a good idea to use a general sentence embedding method and using a better task oriented cost function, like the one proposed in this paper, is necessary.

2) In the second experiment, we used the Skip-Thought vectors proposed in [6]. During training, skip-thought method gets a tuple $(s(t-1), s(t), s(t+1))$ where it encodes the sentence $s(t)$ using one encoder, and tries to reconstruct the previous and next sentences, i.e., $s(t-1)$ and $s(t+1)$, using two separate decoders. The model uses RNNs with Gated Recurrent Unit (GRU) which is shown to perform as good as LSTM. In the paper, authors have emphasized that: "*Our model depends on having a training corpus of contiguous text*". Therefore, training it on our training set where we barely have more than one sentence in query or document title is not fair. However, since their model is trained on 11,038 books from BookCorpus dataset [7] which includes about 74 million sentences, we can use the trained model as an off-the-shelf sentence embedding method as authors have concluded in the conclusion of the paper.

   To do this we downloaded their trained models and word embeddings (its size was more than 2GB) available from "https://github.com/ryankiros/skip-thoughts". Then we encoded each query and its corresponding document title in our test set as vector.

   We used the combine-skip sentence embedding method, a vector of size 4800×1, where it is concatenation of a uni-skip, i.e., a unidirectional encoder resulting in a 2400×1 vector, and a bi-skip, i.e., a bidirectional encoder resulting in a 1200×1 vector by forward encoder and another 1200×1 vector by backward encoder. The authors have reported their best results with the combine-skip encoder.

   Using the 4800×1 embedding vectors for each query and document we calculated the scores and NDCG for the whole test set which are reported in Table IV.

   The proposed method in this paper is performing significantly better than the off-the-shelf skip-thought method for document retrieval task. Nevertheless, since we used skip-thought as an off-the-shelf sentence embedding method, its result is good. This result also confirms that learning embedding vectors using a model and cost function specifically designed for document retrieval task is necessary.

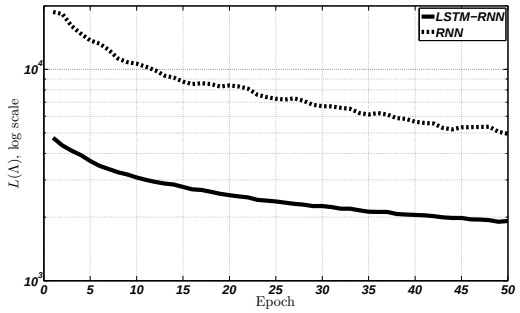


Fig. 7. LSTM-RNN compared to RNN during training: The vertical axis is logarithmic scale of the training cost, $L(\mathbf{\Lambda})$, in (4). Horizontal axis is the number of epochs during training.

As shown in Table IV, the LSTM-RNN significantly outperforms all these models, and exceeds the best baseline model (CLSM) by 1.3% in NDCG@1 score, which is a statistically significant improvement. As we pointed out in Sec. V-A, such an improvement comes from the LSTM-RNN's ability to embed the contextual and semantic information of the sentences into a finite dimension vector. In Table IV, we have also presented the results when different number of negative samples, n, is used. Generally, by increasing n we expect the performance to improve. This is because more negative samples results in a more accurate approximation of the partition function in (5). The results of using Bidirectional LSTM-RNN are also presented in Table IV. In this model, one LSTM-RNN reads queries and documents from left to right, and the other LSTM-RNN reads queries and documents from right to left. Then the embedding vectors from left to right and right to left LSTM-RNNs are concatenated to compute the cosine similarity score and NDCG values.

A comparison between the value of the cost function during training for LSTM-RNN and RNN on the click-through data is shown in Fig. 7. From this figure, we conclude that LSTM-RNN is optimizing the cost function in (4) more effectively. Please note that all parameters of both models are initialized randomly.

TABLE IV
COMPARISONS OF NDCG PERFORMANCE MEASURES (THE HIGHER THE BETTER) OF PROPOSED MODELS AND A SERIES OF BASELINE MODELS, WHERE *nhid* REFERS TO THE NUMBER OF HIDDEN UNITS, *ncell* REFERS TO NUMBER OF CELLS, *win* REFERS TO WINDOW SIZE, AND *n* IS THE NUMBER OF NEGATIVE SAMPLES WHICH IS SET TO 4 UNLESS OTHERWISE STATED. UNLESS STATED OTHERWISE, THE RNN AND LSTM-RNN MODELS ARE CHOSEN TO HAVE THE SAME NUMBER OF MODEL PARAMETERS AS THE DSSM AND CLSM MODELS: 14.4M, WHERE $1\text{M} = 10^6$. THE BOLDFACE NUMBERS ARE THE BEST RESULTS.

Model	NDCG @1	NDCG @3	NDCG @10
Skip-Thought off-the-shelf	26.9%	29.7%	36.2%
Doc2Vec	29.1%	31.8%	38.4%
ULM	30.4%	32.7%	38.5%
BM25	30.5%	32.8%	38.8%
PLSA (T=500)	30.8%	33.7%	40.2%
DSSM (nhid = 288/96) 2 Layers	31.0%	34.4%	41.7%
CLSM (nhid = 288/96, win=1) 2 Layers, 14.4 M parameters	31.8%	35.1%	42.6%
CLSM (nhid = 288/96, win=3) 2 Layers, 43.2 M parameters	32.1%	35.2%	42.7%
CLSM (nhid = 288/96, win=5) 2 Layers, 72 M parameters	32.0%	35.2%	42.6%
RNN (nhid = 288) 1 Layer	31.7%	35.0%	42.3%
LSTM-RNN (ncell = 32) 1 Layer, 4.8 M parameters	31.9%	35.5%	42.7%
LSTM-RNN (ncell = 64) 1 Layer, 9.6 M parameters	32.9%	36.3%	43.4%
LSTM-RNN (ncell = 96) 1 Layer, n = 2	32.6%	36.0%	43.4%
LSTM-RNN (ncell = 96) 1 Layer, n = 4	33.1%	36.5%	43.6%
LSTM-RNN (ncell = 96) 1 Layer, n = 6	33.1%	36.6%	43.6%
LSTM-RNN (ncell = 96) 1 Layer, n = 8	**33.1%**	**36.4%**	**43.7%**
Bidirectional LSTM-RNN (ncell = 96), 1 Layer	**33.2%**	**36.6%**	**43.6%**

VI. CONCLUSIONS AND FUTURE WORK

This paper addresses deep sentence embedding. We propose a model based on long short-term memory to model the long range context information and embed the key information of a sentence in one semantic vector. We show that the semantic vector evolves over time and only takes useful information from any new input. This has been made possible by input gates that detect useless information and attenuate it. Due to general limitation of available human labelled data, we proposed and implemented training the model with a *weak supervision* signal using user click-through data of a commercial web search engine.

By performing a detailed analysis on the model, we showed that: 1) The proposed model is robust to noise, i.e., it mainly embeds keywords in the final semantic vector representing the whole sentence and 2) In the proposed model, each cell is usually allocated to keywords from a specific topic. These findings have been supported using extensive examples. As a concrete sample application of the proposed sentence embedding method, we evaluated it on the important language processing task of web document retrieval. We showed that, for this task, the proposed method outperforms all existing state of the art methods significantly.

This work has been motivated by the earlier successes of deep learning methods in speech [41], [42], [43], [44], [45] and in semantic modelling [3], [10], [46], [47], and it adds further evidence for the effectiveness of these methods. Our future work will further extend the methods to include 1) Using the proposed sentence embedding method for other important language processing tasks for which we believe sentence embedding plays a key role, e.g., the question / answering task. 2) Exploit the prior information about the structure of the different matrices in Fig. 2 to develop a more effective cost function and learning method. 3) Exploiting attention mechanism in the proposed model to improve the performance and find out which words in the query are aligned to which words of the document.

APPENDIX A
EXPRESSIONS FOR THE GRADIENTS

In this appendix we present the final gradient expressions that are necessary to use for training the proposed models. Full derivations of these gradients are presented in section III of supplementary materials.

A. RNN

For the recurrent parameters, $\mathbf{\Lambda} = \mathbf{W}_{rec}$ (we have ommitted r subscript for simplicity):

$$\frac{\partial \Delta_{j,\tau}}{\partial \mathbf{W}_{rec}} = [\delta_{y_Q}^{D^+}(t-\tau)\mathbf{y}_Q^T(t-\tau-1)+ \delta_{y_D}^{D^+}(t-\tau)\mathbf{y}_{D^+}^T(t-\tau-1)] - [\delta_{y_Q}^{D_j^-}(t-\tau)\mathbf{y}_Q^T(t-\tau-1) + \delta_{y_D}^{D_j^-}(t-\tau)\mathbf{y}_{D_j^-}^T(t-\tau-1)] \quad (9)$$

where D_j^- means j-th candidate document that is not clicked and

$$\delta_{\mathbf{y}_Q}(t-\tau-1) = (1-\mathbf{y}_Q(t-\tau-1))\circ (1+\mathbf{y}_Q(t-\tau-1)) \circ \mathbf{W}_{rec}^T \delta_{\mathbf{y}_Q}(t-\tau) \quad (10)$$

and the same as (10) for $\delta_{\mathbf{y}_D}(t-\tau-1)$ with D subscript for document side model. Please also note that:

$$\delta_{\mathbf{y}_Q}(T_Q) = (1-\mathbf{y}_Q(T_Q)) \circ (1+\mathbf{y}_Q(T_Q))\circ (b.c.\mathbf{y}_D(T_D) - a.b^3.c.\mathbf{y}_Q(T_Q)),$$
$$\delta_{\mathbf{y}_D}(T_D) = (1-\mathbf{y}_D(T_D)) \circ (1+\mathbf{y}_D(T_D))\circ (b.c.\mathbf{y}_Q(T_Q) - a.b.c^3.\mathbf{y}_D(T_D)) \quad (11)$$

where

$$a = \mathbf{y}_Q(t=T_Q)^T \mathbf{y}_D(t=T_D)$$
$$b = \frac{1}{\|\mathbf{y}_Q(t=T_Q)\|},\ c = \frac{1}{\|\mathbf{y}_D(t=T_D)\|} \quad (12)$$

For the input parameters, $\mathbf{\Lambda} = \mathbf{W}$:

$$\frac{\partial \Delta_{j,\tau}}{\partial \mathbf{W}} = [\delta_{y_Q}^{D^+}(t-\tau)\mathbf{l}_Q^T(t-\tau) + \delta_{y_D}^{D^+}(t-\tau)\mathbf{l}_{D^+}^T(t-\tau)] - [\delta_{y_Q}^{D_j^-}(t-\tau)\mathbf{l}_Q^T(t-\tau) + \delta_{y_D}^{D_j^-}(t-\tau)\mathbf{l}_{D_j^-}^T(t-\tau)] \quad (13)$$

A full derivation of BPTT for RNN is presented in section III of supplementary materials.

B. LSTM-RNN

Starting with the cost function in (4), we use the Nesterov method described in (6) to update LSTM-RNN model parameters. Here, $\mathbf{\Lambda}$ is one of the weight matrices or bias vectors $\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_{rec1}, \mathbf{W}_{rec2}, \mathbf{W}_{rec3}, \mathbf{W}_{rec4}, \mathbf{W}_{p1}, \mathbf{W}_{p2}, \mathbf{W}_{p3}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4\}$ in the LSTM-RNN architecture. The general format of the gradient of the cost function, $\nabla L(\mathbf{\Lambda})$, is the same as (7). By definition of $\Delta_{r,j}$, we have:

$$\frac{\partial \Delta_{r,j}}{\partial \mathbf{\Lambda}} = \frac{\partial R(Q_r, D_r^+)}{\partial \mathbf{\Lambda}} - \frac{\partial R(Q_r, D_{r,j})}{\partial \mathbf{\Lambda}} \quad (14)$$

We omit r and j subscripts for simplicity and present $\frac{\partial R(Q,D)}{\partial \mathbf{\Lambda}}$ for different parameters of each cell of LSTM-RNN in the following subsections. This will complete the process of calculating $\nabla L(\mathbf{\Lambda})$ in (7) and then we can use (6) to update LSTM-RNN model parameters. In the subsequent subsections vectors $\mathbf{v}_Q$ and $\mathbf{v}_D$ are defined as:

$$\mathbf{v}_Q = (b.c.\mathbf{y}_D(t=T_D) - a.b^3.c.\mathbf{y}_Q(t=T_Q))$$
$$\mathbf{v}_D = (b.c.\mathbf{y}_Q(t=T_Q) - a.b.c^3.\mathbf{y}_D(t=T_D)) \quad (15)$$

where a, b and c are defined in (12). Full derivation of truncated BPTT for LSTM-RNN model is presented in section III of supplementary materials.

1) Output Gate: For recurrent connections we have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec1}} = \delta_{y_Q}^{rec1}(t).\mathbf{y}_Q(t-1)^T + \delta_{y_D}^{rec1}(t).\mathbf{y}_D(t-1)^T \quad (16)$$

where

$$\delta_{y_Q}^{rec1}(t) = \mathbf{o}_Q(t) \circ (1 - \mathbf{o}_Q(t)) \circ h(\mathbf{c}_Q(t)) \circ \mathbf{v}_Q(t) \quad (17)$$

and the same as (17) for $\delta_{y_D}^{rec1}(t)$ with subscript D for document side model. For input connections, $\mathbf{W}_1$, and peephole connections, $\mathbf{W}_{p1}$, we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_1} = \delta_{y_Q}^{rec1}(t).\mathbf{l}_Q(t)^T + \delta_{y_D}^{rec1}(t).\mathbf{l}_D(t)^T \quad (18)$$

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p1}} = \delta_{y_Q}^{rec1}(t).\mathbf{c}_Q(t)^T + \delta_{y_D}^{rec1}(t).\mathbf{c}_D(t)^T \quad (19)$$

The derivative for output gate bias values will be:

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_1} = \delta_{y_Q}^{rec1}(t) + \delta_{y_D}^{rec1}(t) \quad (20)$$

2) Input Gate: For the recurrent connections we have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec3}} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec3}} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec3}} \quad (21)$$

where

$$\delta_{y_Q}^{rec3}(t) = (1 - h(\mathbf{c}_Q(t))) \circ (1 + h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t)$$
$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec3}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec3}} + \mathbf{b}_{i,Q}(t).\mathbf{y}_Q(t-1)^T$$
$$\mathbf{b}_{i,Q}(t) = \mathbf{y}_{g,Q}(t) \circ \mathbf{i}_Q(t) \circ (1 - \mathbf{i}_Q(t)) \quad (22)$$

In equation (21), $\delta_{y_D}^{rec3}(t)$ and $\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec3}}$ are the same as (22) with D subscript. For the input connections we will have the following:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_3} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_3} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_3} \quad (23)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_3} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_3} + \mathbf{b}_{i,Q}(t).\mathbf{x}_Q(t)^T \quad (24)$$

For the peephole connections we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p3}} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p3}} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{p3}} \quad (25)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p3}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{p3}} + \mathbf{b}_{i,Q}(t).\mathbf{c}_Q(t-1)^T \quad (26)$$

For bias values, $\mathbf{b}_3$, we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_3} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_3} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_3} \quad (27)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_3} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_{i,Q}(t) \quad (28)$$

3) Forget Gate: For the recurrent connections we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec2}} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec2}} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec2}} \quad (29)$$

where

$$\delta_{y_Q}^{rec2}(t) = (1 - h(\mathbf{c}_Q(t))) \circ (1 + h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t)$$
$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec2}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec2}} + \mathbf{b}_{f,Q}(t).\mathbf{y}_Q(t-1)^T$$
$$\mathbf{b}_{f,Q}(t) = \mathbf{c}_Q(t-1) \circ \mathbf{f}_Q(t) \circ (1 - \mathbf{f}_Q(t)) \quad (30)$$

For input connections to forget gate we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_2} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_2} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_2} \quad (31)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_2} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_2} + \mathbf{b}_{f,Q}(t).\mathbf{x}_Q(t)^T \quad (32)$$

For peephole connections we have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p2}} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p2}} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{p2}} \quad (33)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p2}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{p2}} + \mathbf{b}_{f,Q}(t).\mathbf{c}_Q(t-1)^T \quad (34)$$

For forget gate's bias values we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_2} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_2} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_2} \quad (35)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_2} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_{f,Q}(t) \quad (36)$$

4) Input without Gating ($\mathbf{y}_g(t)$): For recurrent connections we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec4}} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec4}} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec4}} \quad (37)$$

where

$$\delta_{y_Q}^{rec4}(t) = (1 - h(\mathbf{c}_Q(t))) \circ (1 + h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t)$$
$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec4}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec4}} + \mathbf{b}_{g,Q}(t).\mathbf{y}_Q(t-1)^T$$
$$\mathbf{b}_{g,Q}(t) = \mathbf{i}_Q(t) \circ (1 - \mathbf{y}_{g,Q}(t)) \circ (1 + \mathbf{y}_{g,Q}(t)) \quad (38)$$

For input connection we have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_4} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_4} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_4} \quad (39)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_4} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_4} + \mathbf{b}_{g,Q}(t).\mathbf{x}_Q(t)^T \quad (40)$$

For bias values we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_4} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_4} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_4} \quad (41)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_4} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_4} + \mathbf{b}_{g,Q}(t) \quad (42)$$

5) Error signal backpropagation: Error signals are back propagated through time using following equations:

$$\delta_Q^{rec1}(t-1) = [\mathbf{o}_Q(t-1) \circ (1 - \mathbf{o}_Q(t-1)) \circ h(\mathbf{c}_Q(t-1))] \circ \mathbf{W}_{rec1}^T.\delta_Q^{rec1}(t) \quad (43)$$

$$\delta_Q^{rec_i}(t-1) = [(1 - h(\mathbf{c}_Q(t-1))) \circ (1 + h(\mathbf{c}_Q(t-1))) \circ \mathbf{o}_Q(t-1)] \circ \mathbf{W}_{rec_i}^T.\delta_Q^{rec_i}(t), \quad for \quad i \in \{2,3,4\} \quad (44)$$

REFERENCES

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[2] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *Proceedings of the 31st International Conference on Machine Learning*, pp. 1188–1196, 2014.

[3] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, ser. CIKM '13. ACM, 2013, pp. 2333–2338.

[4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[6] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, "Skip-thought vectors," *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[7] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," *arXiv preprint arXiv:1506.06724*, 2015.

[8] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *NIPS Deep Learning Workshop*, 2014.

[9] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '11, 2011, pp. 151–161.

[10] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval." CIKM, November 2014.

[11] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *International Conference on Machine Learning, ICML*, 2008.

[12] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, June 2014.

[13] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 2042–2050.

[14] J. Zhang, S. Liu, M. Li, M. Zhou, and C. Zong, "Bilingually-constrained phrase embeddings for machine translation," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL) (Volume 1: Long Papers)*, Baltimore, Maryland, 2014, pp. 111–121.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[16] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. ICASSP*, Vancouver, Canada, May 2013.

[17] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proceedings of the Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[18] K. M. Hermann and P. Blunsom, "Multilingual models for compositional distributed semantics," *arXiv preprint arXiv:1404.4641*, 2014.

[19] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ICLR2015*, 2015. [Online]. Available: http://arxiv.org/abs/1409.0473

[20] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," *arXiv preprint arXiv:1506.02078*, 2015.

[21] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[22] A. J. Robinson, "An application of recurrent nets to phone probability estimation," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 298–305, August 1994.

[23] L. Deng, K. Hassanein, and M. Elmasry, "Analysis of the correlation structure for a neural predictive model with application to speech recognition," *Neural Networks*, vol. 7, no. 2, pp. 331–339, 1994.

[24] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Proc. INTERSPEECH*, Makuhari, Japan, September 2010, pp. 1045–1048.

[25] A. Graves, "Sequence transduction with recurrent neural networks," in *Representation Learning Workshp, ICML*, 2012.

[26] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Proc. ICASSP*, Vancouver, Canada, May 2013.

[27] J. Chen and L. Deng, "A primal-dual method for training recurrent neural networks constrained by the echo-state property," in *Proceedings of the International Conf. on Learning Representations (ICLR)*, 2014.

[28] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding," in *Proc. INTERSPEECH*, Lyon, France, August 2013.

[29] L. Deng and J. Chen, "Sequence classification using high-level features extracted from deep neural networks," in *Proc. ICASSP*, 2014.

[30] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, pp. 2451–2471, 1999.

[31] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, Mar. 2003.

[32] J. Gao, W. Yuan, X. Li, K. Deng, and J.-Y. Nie, "Smoothing clickthrough data for web search ranking," in *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '09. New York, NY, USA: ACM, 2009, pp. 355–362.

[33] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983.

[34] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *ICML (3)'13*, 2013, pp. 1139–1147.

[35] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks." in *ICML 2013*, ser. JMLR Proceedings, vol. 28. JMLR.org, 2013, pp. 1310–1318.

[36] K. Järvelin and J. Kekäläinen, "Ir evaluation methods for retrieving highly relevant documents," in *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR. ACM, 2000, pp. 41–48.

[37] T. Hofmann, "Probabilistic latent semantic analysis," in *In Proc. of Uncertainty in Artificial Intelligence, UAI99*, 1999, pp. 289–296.

[38] J. Gao, K. Toutanova, and W.-t. Yih, "Clickthrough-based latent semantic models for web search," ser. SIGIR '11. ACM, 2011, pp. 675–684.

[39] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations (ICLR)*, 2013. [Online]. Available: arXiv:1301.3781

[40] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[41] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Large vocabulary continuous speech recognition with context-dependent DBN-HMMs," in *Proc. IEEE ICASSP*, Prague, Czech, May 2011, pp. 4688–4691.

[42] D. Yu and L. Deng, "Deep learning and its applications to signal and information processing [exploratory dsp]," *IEEE Signal Processing Magazine*, vol. 28, no. 1, pp. 145 –154, jan. 2011.

[43] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30 –42, jan. 2012.

[44] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, November 2012.

[45] L. Deng, D. Yu, and J. Platt, "Scalable stacking and learning for building deep architectures," in *Proc. ICASSP*, march 2012, pp. 2133 –2136.
[46] J. Gao, P. Pantel, M. Gamon, X. He, L. Deng, and Y. Shen, "Modeling interestingness with deep neural networks," in *Proc. EMNLP*, 2014.
[47] J. Gao, X. He, W. tau Yih, and L. Deng, "Learning continuous phrase representations for translation modeling," in *Proc. ACL*, 2014.

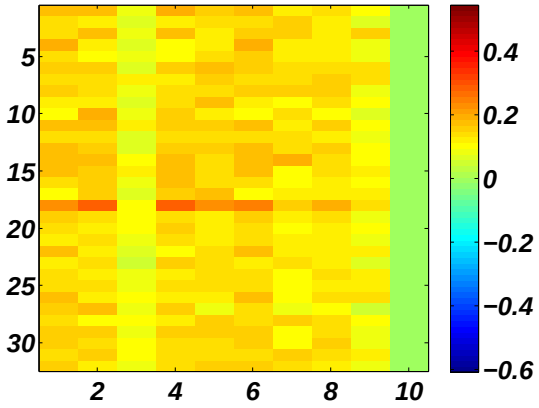


Fig. 8. Input gate, $\mathbf{i}(t)$, for Document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

SUPPLEMENTARY MATERIAL

APPENDIX B
A MORE CLEAR FIGURE FOR INPUT GATE FOR "*hotels in shanghai*" EXAMPLE

In this section we present a more clear figure for part (a) of Fig. 5 that shows the structure of the input gate for document side of "*hotels in shanghai*" example. As it is clearly visible from this figure, the input gate values for most of the cells corresponding to word 3, word 7 and word 9 in document side of LSTM-RNN have very small values (light green-yellow color). These are corresponding to words "*accommodation*", "*discount*" and "*reservation*" respectively in the document title. Interestingly, input gates are trying to reduce effect of these three words in the final representation ($\mathbf{y}(t)$) because the LSTM-RNN model is trained to maximize the similarity between query and document if they are a good match.

APPENDIX C
A CLOSER LOOK AT RNNS WITH AND WITHOUT LSTM CELLS IN WEB DOCUMENT RETRIEVAL TASK

In this section we further show examples to reveal the advantage of LSTM-RNN sentence embedding compared to the RNN sentence embedding.

First, we compare the scores assigned by trained RNN and LSTM-RNN to our "*hotels in shanghai*" example. On average, each query in our test dataset is associated with 15 web documents (URLs). Each query / document pair has a relevance label which is human generated. These relevance labels are "Bad", "Fair", "Good" and "Excellent". This example is rated as a "Good" match in the dataset. The score for this pair assigned by RNN is "0.8165" while the score assigned by LSTM-RNN is "0.9161". Please note that the score is between 0 and 1. This means that the score assigned by LSTM-RNN is more correspondent with the human generated label.

TABLE V
RNNS WITH & WITHOUT LSTM CELLS FOR THE SAME QUERY: "*hotels in shanghai*"

	hotels	*in*	*shanghai*
Number of assigned cells out of 10 (LSTM-RNN)	-	0	7
Number of assigned neurons out of 10 (RNN)	-	2	9

Second, we compare the number of assigned neurons and cells to each word by RNN and LSTM-RNN respectively. To do this, we rely on the 10 most active cells and neurons in the final semantic vectors in both models. Results are presented in Table V and Table VI for query and document respectively. An interesting observation is that RNN sometimes assigns neurons to unimportant words, e.g., 6 neurons are assigned to the word "*in*" in Table VI.

As another example we consider the query, "*how to fix bath tub wont turn off*". This example is rated as a "Bad" match in the dataset by human. It is good to know that the score for this pair assigned by RNN is "0.7016" while the score assigned by LSTM-RNN is "0.5944". This shows the score generated by LSTM-RNN is closer to human generated label.

Number of assigned neurons and cells to each word by RNN and LSTM-RNN are presented in Table VII and Table VIII for query and document. This is out of 10 most active neurons and cells in the semantic vector of RNN and LSTM-RNN. Examples of RNN assigning neurons to unimportant words are 3 neurons to the word "a" and 4 neurons to the word "you" in Table VIII.

APPENDIX D
DERIVATION OF BPTT FOR RNN AND LSTM-RNN

In this appendix we present the full derivation of the gradients for RNN and LSTM-RNN.

A. Derivation of BPTT for RNN

From (4) and (5) we have:

$$\frac{\partial L(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} = \sum_{r=1}^{N} \frac{\partial l_r(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} = -\sum_{r=1}^{N}\sum_{j=1}^{n} \alpha_{r,j} \frac{\partial \Delta_{r,j}}{\partial \mathbf{\Lambda}} \quad (45)$$

where

$$\alpha_{r,j} = \frac{-\gamma e^{-\gamma \Delta_{r,j}}}{1 + \sum_{j=1}^{n} e^{-\gamma \Delta_{r,j}}} \quad (46)$$

TABLE VI
RNNs WITH & WITHOUT LSTM CELLS FOR THE SAME DOCUMENT: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

	$shanghai$	$hotels$	$accommodation$	$hotel$	in	$shanghai$	$discount$	and	$reservation$
Number of assigned cells out of 10 (LSTM-RNN)	-	4	3	8	1	8	5	3	4
Number of assigned neurons out of 10 (RNN)	-	10	7	9	6	8	3	2	6

TABLE VII
RNN VERSUS LSTM-RNN FOR QUERY: "$how\ to\ fix\ bath\ tub\ wont\ turn\ off$"

	how	to	fix	$bath$	tub	$wont$	$turn$	off
Number of assigned cells out of 10 (LSTM-RNN)	-	0	4	7	6	3	5	0
Number of assigned neurons out of 10 (RNN)	-	1	10	4	6	2	7	1

TABLE VIII
RNN VERSUS LSTM-RNN FOR DOCUMENT: "$how\ do\ you\ paint\ a\ bathtub\ and\ what\ paint\ should\ldots$"

	how	do	you	$paint$	a	$bathtub$	and	$what$	$paint$	$should\ you\ldots$
Number of assigned cells out of 10(LSTM-RNN)	-	1	1	7	0	9	2	3	8	4
Number of assigned neurons out of 10(RNN)	-	1	4	4	3	7	2	5	4	7

and

$$\Delta_{r,j} = R(Q_r, D_r^+) - R(Q_r, D_{r,j}) \tag{47}$$

We need to find $\frac{\partial \Delta_{r,j}}{\partial \mathbf{\Lambda}}$ for input weights and recurrent weights. We omit r subscript for simplicity.

1) Recurrent Weights:

$$\frac{\partial \Delta_j}{\partial \mathbf{W}_{rec}} = \frac{\partial R(Q, D^+)}{\partial \mathbf{W}_{rec}} - \frac{\partial R(Q, D_j^-)}{\partial \mathbf{W}_{rec}} \tag{48}$$

We divide $R(D, Q)$ into three components:

$$R(Q, D) = \underbrace{\mathbf{y}_Q(t = T_Q)^T \mathbf{y}_D(t = T_D)}_{a}. \underbrace{\frac{1}{\|\mathbf{y}_Q(t = T_Q)\|}}_{b} . \underbrace{\frac{1}{\|\mathbf{y}_D(t = T_D)\|}}_{c} \tag{49}$$

then

$$\frac{\partial R(Q, D)}{\partial \mathbf{W}_{rec}} = \underbrace{\frac{\partial a}{\partial \mathbf{W}_{rec}}.b.c}_{\mathbf{D}} + \underbrace{a.\frac{\partial b}{\partial \mathbf{W}_{rec}}.c}_{\mathbf{E}} + \underbrace{a.b.\frac{\partial c}{\partial \mathbf{W}_{rec}}}_{\mathbf{F}} \tag{50}$$

We have

$$\begin{aligned}\mathbf{D} &= \frac{\partial \mathbf{y}_Q(t = T_Q)^T \mathbf{y}_D(t = T_D).b.c}{\partial \mathbf{W}_{rec}} \\ &= \frac{\partial \mathbf{y}_Q(t = T_Q)^T \mathbf{y}_D(t = T_D).b.c}{\partial \mathbf{y}_Q(t = T_Q)} . \frac{\partial \mathbf{y}_Q(t = T_Q)}{\partial \mathbf{W}_{rec}} + \\ &\frac{\partial \mathbf{y}_Q(t = T_Q)^T \mathbf{y}_D(t = T_D).b.c}{\partial \mathbf{y}_D(t = T_D)} . \frac{\partial \mathbf{y}_D(t = T_D)}{\partial \mathbf{W}_{rec}} \\ &= \mathbf{y}_D(t = T_D).b.c. \frac{\partial \mathbf{y}_Q(t = T_Q)}{\partial \mathbf{W}_{rec}} + \\ &\mathbf{y}_Q(t = T_Q). \underbrace{(b.c)^T}_{b.c} . \frac{\partial \mathbf{y}_D(t = T_D)}{\partial \mathbf{W}_{rec}}\end{aligned} \tag{51}$$

Since $f(.) = tanh(.)$, using chain rule we have

$$\frac{\partial \mathbf{y}_Q(t = T_Q)}{\mathbf{W}_{rec}} = [(1 - \mathbf{y}_Q(t = T_Q)) \circ (1 + \mathbf{y}_Q(t = T_Q))]\mathbf{y}_Q(t-1)^T \tag{52}$$

and therefore

$$\begin{aligned}\mathbf{D} = &[b.c.\mathbf{y}_D(t = T_D) \circ (1 - \mathbf{y}_Q(t = T_Q)) \circ \\ &(1 + \mathbf{y}_Q(t = T_Q))]\mathbf{y}_Q(t-1)^T + \\ &[b.c.\mathbf{y}_Q(t = T_Q) \circ (1 - \mathbf{y}_D(t = T_D)) \circ \\ &(1 + \mathbf{y}_D(t = T_D))]\mathbf{y}_D(t-1)^T\end{aligned} \tag{53}$$

To find $\mathbf{E}$ we use following basic rule:

$$\frac{\partial}{\partial \mathbf{x}} \|\mathbf{x} - \mathbf{a}\|_2 = \frac{\mathbf{x} - \mathbf{a}}{\|\mathbf{x} - \mathbf{a}\|_2} \tag{54}$$

Therefore

$$\mathbf{E} = a.c.\frac{\partial}{\partial \mathbf{W}_{rec}}(\|\mathbf{y}_Q(t=T_Q)\|)^{-1} = \\ - a.c.(\|\mathbf{y}_Q(t=T_Q)\|)^{-2}.\frac{\partial\|\mathbf{y}_Q(t=T_Q)\|}{\partial \mathbf{W}_{rec}} \\ = -a.c.(\|\mathbf{y}_Q(t=T_Q)\|)^{-2}.\frac{\mathbf{y}_Q(t=T_Q)}{\|\mathbf{y}_Q(t=T_Q)\|}\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \mathbf{W}_{rec}} \\ = -[a.c.b^3.\mathbf{y}_Q(t=T_Q)\circ(1-\mathbf{y}_Q(t=T_Q))\circ \\ (1+\mathbf{y}_Q(t=T_Q))]\mathbf{y}_Q(t-1) \quad (55)$$

$\mathbf{F}$ is calculated similar to (55):

$$\mathbf{F} = -[a.b.c^3.\mathbf{y}_D(t=T_D)\circ(1-\mathbf{y}_D(t=T_D))\circ \\ (1+\mathbf{y}_D(t=T_D))]\mathbf{y}_D(t-1) \quad (56)$$

Considering (50),(53),(55) and (56) we have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec}} = \delta_{\mathbf{y}_Q}(t)\mathbf{y}_Q(t-1)^T + \delta_{\mathbf{y}_D}(t)\mathbf{y}_D(t-1)^T \quad (57)$$

where

$$\delta_{\mathbf{y}_Q}(t=T_Q) = (1-\mathbf{y}_Q(t=T_Q))\circ(1+\mathbf{y}_Q(t=T_Q))\circ \\ (b.c.\mathbf{y}_D(t=T_D) - a.b^3.c.\mathbf{y}_Q(t=T_Q)), \\ \delta_{\mathbf{y}_D}(t=T_D) = (1-\mathbf{y}_D(t=T_D))\circ(1+\mathbf{y}_D(t=T_D))\circ \\ (b.c.\mathbf{y}_Q(t=T_Q) - a.b.c^3.\mathbf{y}_D(t=T_D)) \quad (58)$$

Equation (58) will just unfold the network one time step, to unfold it over rest of time steps using backpropagation we have:

$$\delta_{\mathbf{y}_Q}(t-\tau-1) = (1-\mathbf{y}_Q(t-\tau-1))\circ \\ (1+\mathbf{y}_Q(t-\tau-1))\circ \mathbf{W}_{rec}^T\delta_{\mathbf{y}_Q}(t-\tau), \\ \delta_{\mathbf{y}_D}(t-\tau-1) = (1-\mathbf{y}_D(t-\tau-1))\circ \\ (1+\mathbf{y}_D(t-\tau-1))\circ \mathbf{W}_{rec}^T\delta_{\mathbf{y}_D}(t-\tau) \quad (59)$$

where τ is the number of time steps that we unfold the network over time which is from 0 to T_Q and T_D for queries and documents respectively. Now using (48) we have:

$$\frac{\partial \Delta_{j,\tau}}{\partial \mathbf{W}_{rec}} = [\delta_{y_Q}^{D^+}(t-\tau)\mathbf{y}_Q^T(t-\tau-1)+ \\ \delta_{y_D}^{D^+}(t-\tau)\mathbf{y}_{D^+}^T(t-\tau-1)] - [\delta_{y_Q}^{D_j^-}(t-\tau)\mathbf{y}_Q^T(t-\tau-1) \\ + \delta_{y_D}^{D_j^-}(t-\tau)\mathbf{y}_{D_j^-}^T(t-\tau-1)] \quad (60)$$

To calculate final value of gradient we should fold back the network over time and use (45), we will have:

$$\frac{\partial L(\mathbf{\Lambda})}{\partial \mathbf{W}_{rec}} = \underbrace{-\sum_{r=1}^{N}\sum_{j=1}^{n}\sum_{\tau=0}^{T}\alpha_{r,j,T_{D,Q}}\frac{\partial \Delta_{r,j,\tau}}{\partial \mathbf{W}_{rec}}}_{\text{one large update}} \quad (61)$$

2) Input Weights: Using a similar procedure we will have the following for input weights:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}} = \delta_{\mathbf{y}_Q}(t-\tau)\mathbf{l}_Q(t-\tau)^T + \delta_{\mathbf{y}_D}(t-\tau)\mathbf{l}_D(t-\tau)^T \quad (62)$$

where

$$\delta_{\mathbf{y}_Q}(t-\tau) = (1-\mathbf{y}_Q(t-\tau))\circ(1+\mathbf{y}_Q(t-\tau))\circ \\ (b.c.\mathbf{y}_D(t-\tau) - a.b^3.c.\mathbf{y}_Q(t-\tau)), \\ \delta_{\mathbf{y}_D}(t-\tau) = (1-\mathbf{y}_D(t-\tau))\circ(1+\mathbf{y}_D(t-\tau))\circ \\ (b.c.\mathbf{y}_Q(t-\tau) - a.b.c^3.\mathbf{y}_D(t-\tau)) \quad (63)$$

Therefore:

$$\frac{\partial \Delta_{j,\tau}}{\partial \mathbf{W}} = \\ [\delta_{y_Q}^{D^+}(t-\tau)\mathbf{l}_Q^T(t-\tau) + \delta_{y_D}^{D^+}(t-\tau)\mathbf{l}_{D^+}^T(t-\tau)] - \\ [\delta_{y_Q}^{D_j^-}(t-\tau)\mathbf{l}_Q^T(t-\tau) + \delta_{y_D}^{D_j^-}(t-\tau)\mathbf{l}_{D_j^-}^T(t-\tau)] \quad (64)$$

and therefore:

$$\frac{\partial L(\mathbf{\Lambda})}{\partial \mathbf{W}} = \underbrace{-\sum_{r=1}^{N}\sum_{j=1}^{n}\sum_{\tau=0}^{T}\alpha_{r,j}\frac{\partial \Delta_{r,j,\tau}}{\partial \mathbf{W}}}_{\text{one large update}} \quad (65)$$

B. Derivation of BPTT for LSTM-RNN

Following from (50) for every parameter, $\mathbf{\Lambda}$, in LSTM-RNN architecture we have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{\Lambda}} = \underbrace{\frac{\partial a}{\partial \mathbf{\Lambda}}.b.c}_{\mathbf{D}} + \underbrace{a.\frac{\partial b}{\partial \mathbf{\Lambda}}.c}_{\mathbf{E}} + \underbrace{a.b.\frac{\partial c}{\partial \mathbf{\Lambda}}}_{\mathbf{F}} \quad (66)$$

and from (51):

$$\mathbf{D} = \mathbf{y}_D(t=T_D).b.c.\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \mathbf{\Lambda}} + \\ \mathbf{y}_Q(t=T_Q).b.c.\frac{\partial \mathbf{y}_D(t=T_D)}{\partial \mathbf{\Lambda}} \quad (67)$$

From (55) and (56) we have:

$$\mathbf{E} = -a.c.b^3.\mathbf{y}_Q(t=T_Q)\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \mathbf{\Lambda}} \quad (68)$$

$$\mathbf{F} = -a.b.c^3.\mathbf{y}_D(t=T_D)\frac{\partial \mathbf{y}_D(t=T_D)}{\partial \mathbf{\Lambda}} \quad (69)$$

Therefore

$$\frac{\partial R(Q,D)}{\partial \mathbf{\Lambda}} = \mathbf{D}+\mathbf{E}+\mathbf{F} = \\ \mathbf{v}_Q\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \mathbf{\Lambda}} + \mathbf{v}_D\frac{\partial \mathbf{y}_D(t=T_D)}{\partial \mathbf{\Lambda}} \quad (70)$$

where

$$\mathbf{v}_Q = (b.c.\mathbf{y}_D(t=T_D) - a.b^3.c.\mathbf{y}_Q(t=T_Q)) \\ \mathbf{v}_D = (b.c.\mathbf{y}_Q(t=T_Q) - a.b.c^3.\mathbf{y}_D(t=T_D)) \quad (71)$$

1) Output Gate: Since $\alpha \circ \beta = diag(\alpha)\beta = diag(\beta)\alpha$ where $diag(\alpha)$ is a diagonal matrix whose main diagonal entries are entries of vector α, we have:

$$\begin{aligned}
&\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec1}} = \frac{\partial}{\partial \mathbf{W}_{rec1}}(diag(h(\mathbf{c}(t))).\mathbf{o}(t)) \\
&= \underbrace{\frac{\partial diag(h(\mathbf{c}(t)))}{\partial \mathbf{W}_{rec1}}}_{zero}.\mathbf{o}(t) + diag(h(\mathbf{c}(t))).\frac{\partial \mathbf{o}(t)}{\partial \mathbf{W}_{rec1}} \\
&= \mathbf{o}(t) \circ (1 - \mathbf{o}(t)) \circ h(\mathbf{c}(t)).\mathbf{y}(t-1)^T
\end{aligned} \tag{72}$$

Substituting (72) in (70) we have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec1}} = \delta^{rec1}_{y_Q}(t).\mathbf{y}_Q(t-1)^T + \delta^{rec1}_{y_D}(t).\mathbf{y}_D(t-1)^T \tag{73}$$

where

$$\begin{aligned}
\delta^{rec1}_{y_Q}(t) &= \mathbf{o}_Q(t) \circ (1 - \mathbf{o}_Q(t)) \circ h(\mathbf{c}_Q(t)) \circ \mathbf{v}_Q(t) \\
\delta^{rec1}_{y_D}(t) &= \mathbf{o}_D(t) \circ (1 - \mathbf{o}_D(t)) \circ h(\mathbf{c}_D(t)) \circ \mathbf{v}_D(t)
\end{aligned} \tag{74}$$

with a similar derivation for $\mathbf{W}_1$ and $\mathbf{W}_{p1}$ we get:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_1} = \delta^{rec1}_{y_Q}(t).\mathbf{l}_Q(t)^T + \delta^{rec1}_{y_D}(t).\mathbf{l}_D(t)^T \tag{75}$$

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p1}} = \delta^{rec1}_{y_Q}(t).\mathbf{c}_Q(t)^T + \delta^{rec1}_{y_D}(t).\mathbf{c}_D(t)^T \tag{76}$$

For output gate bias values we have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_1} = \delta^{rec1}_{y_Q}(t) + \delta^{rec1}_{y_D}(t) \tag{77}$$

2) Input Gate: Similar to output gate we start with:

$$\begin{aligned}
&\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} = \frac{\partial}{\partial \mathbf{W}_{rec3}}(diag(\mathbf{o}(t)).h(\mathbf{c}(t))) \\
&= \underbrace{\frac{\partial diag(\mathbf{o}(t))}{\partial \mathbf{W}_{rec3}}}_{zero}.h(\mathbf{c}(t)) + diag(\mathbf{o}(t)).\frac{\partial h(\mathbf{c}(t))}{\partial \mathbf{W}_{rec3}} \\
&= diag(\mathbf{o}(t)).(1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t)))\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}}
\end{aligned} \tag{78}$$

To find $\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}}$ assuming $\mathbf{f}(t) = 1$ (we derive formulation for $\mathbf{f}(t) \neq 1$ from this simple solution) we have:

$$\begin{aligned}
&\mathbf{c}(0) = 0 \\
&\mathbf{c}(1) = \mathbf{c}(0) + \mathbf{i}(1) \circ \mathbf{y}_g(1) = \mathbf{i}(1) \circ \mathbf{y}_g(1) \\
&\mathbf{c}(2) = \mathbf{c}(1) + \mathbf{i}(2) \circ \mathbf{y}_g(2) \\
&\dots \\
&\mathbf{c}(t) = \sum_{k=1}^{t} \mathbf{i}(k) \circ \mathbf{y}_g(k) = \sum_{k=1}^{t} diag(\mathbf{y}_g(k)).\mathbf{i}(k)
\end{aligned} \tag{79}$$

Therefore

$$\begin{aligned}
\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}} &= \sum_{k=1}^{t}[\underbrace{\frac{\partial diag(\mathbf{y}_g(k))}{\mathbf{W}_{rec3}}}_{zero}.\mathbf{i}(k) + diag(\mathbf{y}_g(k)).\frac{\partial \mathbf{i}(k)}{\mathbf{W}_{rec3}}] \\
&= \sum_{k=1}^{t} diag(\mathbf{y}_g(k)).\mathbf{i}(k) \circ (1 - \mathbf{i}(k)).\mathbf{y}(k-1)^T
\end{aligned} \tag{80}$$

(81)

and

$$\begin{aligned}
\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} &= \sum_{k=1}^{t}[\underbrace{\mathbf{o}(t) \circ (1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t)))}_{\mathbf{a}(t)} \\
&\circ \underbrace{\mathbf{y}_g(k) \circ \mathbf{i}(k) \circ (1 - \mathbf{i}(k))}_{\mathbf{b}(k)}]\mathbf{y}(k-1)^T
\end{aligned} \tag{82}$$

But this is expensive to implement, to resolve it we have:

$$\begin{aligned}
\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} &= \underbrace{\sum_{k=1}^{t-1}[\mathbf{a}(t) \circ \mathbf{b}(k)]\mathbf{y}(k-1)^T}_{\text{expensive part}} \\
&+ [\mathbf{a}(t) \circ \mathbf{b}(t)]\mathbf{y}(t-1)^T \\
&= diag(\mathbf{a}(t)) \underbrace{\sum_{k=1}^{t-1} \mathbf{b}(k).\mathbf{y}(k-1)^T}_{\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec3}}} \\
&+ diag(\mathbf{a}(t)).\mathbf{b}(t).\mathbf{y}(t-1)^T
\end{aligned} \tag{83}$$

Therefore

$$\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} = [diag(\mathbf{a}(t))][\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec3}} + \mathbf{b}(t).\mathbf{y}(t-1)^T] \tag{84}$$

For $\mathbf{f}(t) \neq 1$ we have

$$\begin{aligned}
\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} &= [diag(\mathbf{a}(t))][diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec3}} \\
&+ \mathbf{b}_i(t).\mathbf{y}(t-1)^T]
\end{aligned} \tag{85}$$

where

$$\begin{aligned}
\mathbf{a}(t) &= \mathbf{o}(t) \circ (1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t))) \\
\mathbf{b}_i(t) &= \mathbf{y}_g(t) \circ \mathbf{i}(t) \circ (1 - \mathbf{i}(t))
\end{aligned} \tag{86}$$

substituting above equation in (70) we will have:

$$\begin{aligned}
\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec3}} &= diag(\delta^{rec3}_{y_Q}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec3}} \\
&+ diag(\delta^{rec3}_{y_D}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec3}}
\end{aligned} \tag{87}$$

where

$$\begin{aligned}
&\delta^{rec3}_{y_Q}(t) = (1 - h(\mathbf{c}_Q(t))) \circ (1 + h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t) \\
&\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec3}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec3}} + \mathbf{b}_{i,Q}(t).\mathbf{y}_Q(t-1)^T \\
&\mathbf{b}_{i,Q}(t) = \mathbf{y}_{g,Q}(t) \circ \mathbf{i}_Q(t) \circ (1 - \mathbf{i}_Q(t))
\end{aligned} \tag{88}$$

In equation (87), $\delta_{y_D}^{rec3}(t)$ and $\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec3}}$ are the same as (88) with D subscript. Therefore, update equations for $\mathbf{W}_{rec3}$ are (87), (88) for Q and D and (6).

With a similar procedure for $\mathbf{W}_3$ we will have the following:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_3} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_3} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_3} \tag{89}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_3} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_3} + \mathbf{b}_{i,Q}(t).\mathbf{x}_Q(t)^T \tag{90}$$

Therefore, update equations for $\mathbf{W}_3$ are (89), (90) for Q and D and (6).

For peephole connections we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p3}} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p3}} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{p3}} \tag{91}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p3}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{p3}} + \mathbf{b}_{i,Q}(t).\mathbf{c}_Q(t-1)^T \tag{92}$$

Hence, update equations for $\mathbf{W}_{p3}$ are (91), (92) for Q and D and (6).

Following similar derivation for bias values $\mathbf{b}_3$ we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_3} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_3} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_3} \tag{93}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_3} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_{i,Q}(t) \tag{94}$$

Update equations for $\mathbf{b}_3$ are (93), (94) for Q and D and (6).

3) Forget Gate: For forget gate, with a similar derivation to input gate we will have

$$\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec2}} = [diag(\mathbf{a}(t))][diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec2}} + \mathbf{b}_f(t).\mathbf{y}(t-1)^T] \tag{95}$$

where

$$\mathbf{a}(t) = \mathbf{o}(t) \circ (1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t)))$$
$$\mathbf{b}_f(t) = \mathbf{c}(t-1) \circ \mathbf{f}(t) \circ (1 - \mathbf{f}(t)) \tag{96}$$

substituting above equation in (70) we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec2}} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec2}} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec2}} \tag{97}$$

where

$$\delta_{y_Q}^{rec2}(t) = (1 - h(\mathbf{c}_Q(t))) \circ (1 + h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t)$$
$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec2}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec2}} + \mathbf{b}_{f,Q}(t).\mathbf{y}_Q(t-1)^T$$
$$\mathbf{b}_{f,Q}(t) = \mathbf{c}_Q(t-1) \circ \mathbf{f}_Q(t) \circ (1 - \mathbf{f}_Q(t)) \tag{98}$$

Therefore, update equations for $\mathbf{W}_{rec2}$ are (97), (98) for Q and D and (6).

For input weights to forget gate, $\mathbf{W}_2$, we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_2} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_2} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_2} \tag{99}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_2} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_2} + \mathbf{b}_{f,Q}(t).\mathbf{x}_Q(t)^T \tag{100}$$

Therefore, update equations for $\mathbf{W}_2$ are (99), (100) for Q and D and (6).

For peephole connections, $\mathbf{W}_{p2}$, we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p2}} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p2}} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{p2}} \tag{101}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p2}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{p2}} + \mathbf{b}_{f,Q}(t).\mathbf{c}_Q(t-1)^T \tag{102}$$

Therefore, update equations for $\mathbf{W}_{p2}$ are (101), (102) for Q and D and (6).

Update equations for forget gate bias values, $\mathbf{b}_2$, will be following equations and (6):

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_2} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_2} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_2} \tag{103}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_2} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_{f,Q}(t) \tag{104}$$

4) Input without Gating ($\mathbf{y}_g(t)$): Gradients for $\mathbf{y}_g(t)$ parameters are as follows:

$$\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec4}} = [diag(\mathbf{a}(t))][diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec4}} + \mathbf{b}_g(t).\mathbf{y}(t-1)^T] \quad (105)$$

where

$$\mathbf{a}(t) = \mathbf{o}(t) \circ (1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t)))$$
$$\mathbf{b}_g(t) = \mathbf{i}(t) \circ (1 - \mathbf{y}_g(t)) \circ (1 + \mathbf{y}_g(t)) \quad (106)$$

substituting above equation in (70) we will have:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec4}} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec4}} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec4}} \quad (107)$$

where

$$\delta_{y_Q}^{rec4}(t) = (1 - h(\mathbf{c}_Q(t))) \circ (1 + h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t)$$
$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec4}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec4}} + \mathbf{b}_{g,Q}(t).\mathbf{y}_Q(t-1)^T$$
$$\mathbf{b}_{g,Q}(t) = \mathbf{i}_Q(t) \circ (1 - \mathbf{y}_{g,Q}(t)) \circ (1 + \mathbf{y}_{g,Q}(t)) \quad (108)$$

Therefore, update equations for $\mathbf{W}_{rec4}$ are (107), (108) for Q and D and (6).

For input weight parameters, $\mathbf{W}_4$, we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_4} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_4} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_4} \quad (109)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_4} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_4} + \mathbf{b}_{g,Q}(t).\mathbf{x}_Q(t)^T \quad (110)$$

Therefore, update equations for $\mathbf{W}_4$ are (109), (110) for Q and D and (6).

Gradients with respect to bias values, $\mathbf{b}_4$, are

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_4} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_4} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_4} \quad (111)$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_4} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_4} + \mathbf{b}_{g,Q}(t) \quad (112)$$

Therefore, update equations for $\mathbf{b}_4$ are (111), (112) for Q and D and (6). There is no peephole connections for $\mathbf{y}_g(t)$.

APPENDIX E
LSTM-RNN VISUALIZATION

In this appendix we present more examples of LSTM-RNN visualization.

A. LSTM-RNN Semantic Vectors: Another Example

Consider the following example from test dataset:

- Query: "*how to fix bath tub wont turn off*"
- Document: "*how do you paint a bathtub and what paint* $\underbrace{\textit{should you use yahoo answers}}_{\text{treated as one word}}$"

Activations of input gate, output gate, cell state and cell output for each cell for query and document are presented in Fig.9 and Fig.10 respectively based on a trained LSTM-RNN model.

Three interesting observations from Fig.9 and Fig.10:

- Semantic representation $\mathbf{y}(t)$ and cell states $\mathbf{c}(t)$ are evolving over time.
- In part (a) of Fig.10, we observe that input gate values for most of the cells corresponding to word 3, word 4, word 7 and word 9 in document side of LSTM-RNN have very small values (light blue color). These are corresponding to words "*you*", "*paint*", "*and*" and "*paint*" respectively in the document title. Interestingly, input gates are trying to reduce effect of these words in the final representation ($\mathbf{y}(t)$) because the LSTM-RNN model is trained to maximize the similarity between query and document if they are a good match.
- $\mathbf{y}(t)$ is used as semantic representation after applying output gate on cell states. Note that valuable context information is stored in cell states $\mathbf{c}(t)$.

B. Key Word Extraction: Another Example

Evolution of 10 most active cells over time for the second example are presented in Fig. 11 for query and Fig. 12 for document. Number of assigned cells out of 10 most active cells to each word are presented in Table IX and Table X.

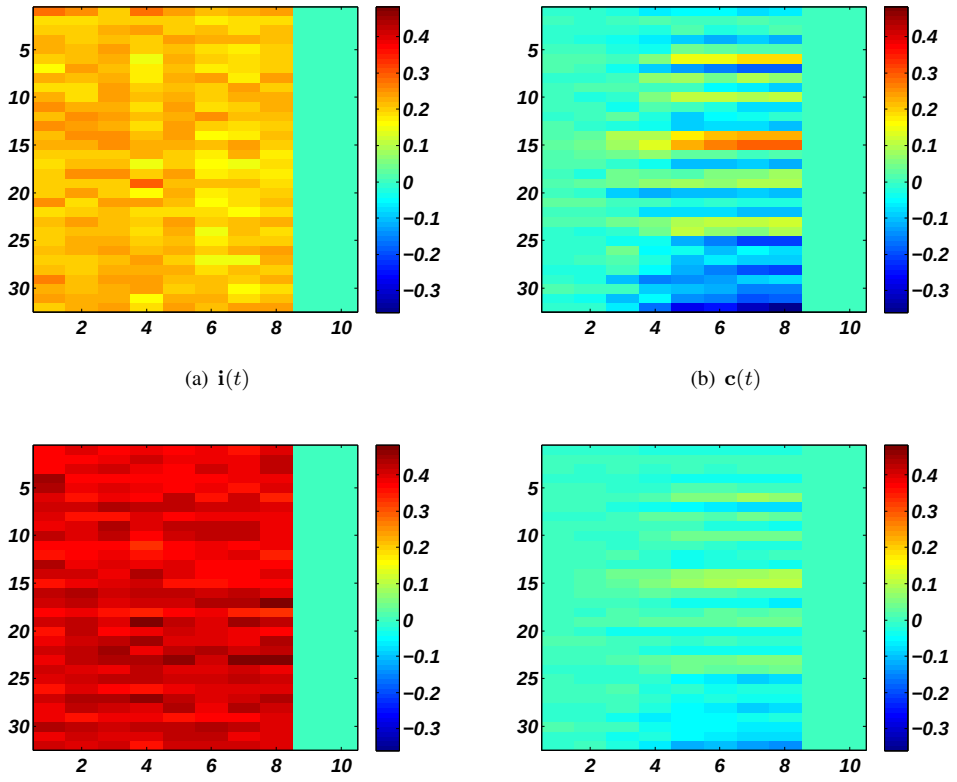
APPENDIX F
DOC2VEC SIMILARITY TEST

To make sure that a meaningful model is trained, we used the trained doc2vec model to find the most similar words to two sample words in our dataset, the words "pizza" and "infection". The resulting words and corresponding scores are as follows:

---

print(model.most-similar('pizza')) :

[(u'recipes', 0.9316294193267822),
(u'recipe', 0.9295548796653748),
(u'food', 0.9250608682632446),
(u'restaurants', 0.9223555326461792),
(u'bar', 0.9191627502441406),
(u'sabayon', 0.916868269443512),
(u'page', 0.9160783290863037),



(a) $\mathbf{i}(t)$ (b) $\mathbf{c}(t)$

(c) $\mathbf{o}(t)$ (d) $\mathbf{y}(t)$

Fig. 9. Query: "*how to fix bath tub wont turn off*"

TABLE IX
KEYWORD EXTRACTION FOR QUERY: "$how\ to\ fix\ bath\ tub\ wont\ turn\ off$"

	how	to	fix	$bath$	tub	$wont$	$turn$	off
Number of assigned cells out of 10 Left to Right	-	0	**4**	**7**	**6**	3	**5**	0
Number of assigned cells out of 10 Right to Left	**4**	1	**6**	**7**	**6**	**7**	**7**	-

(u'restaurant', 0.9112323522567749),
(u'house', 0.9104640483856201),
(u'the', 0.9103578925132751)]

---

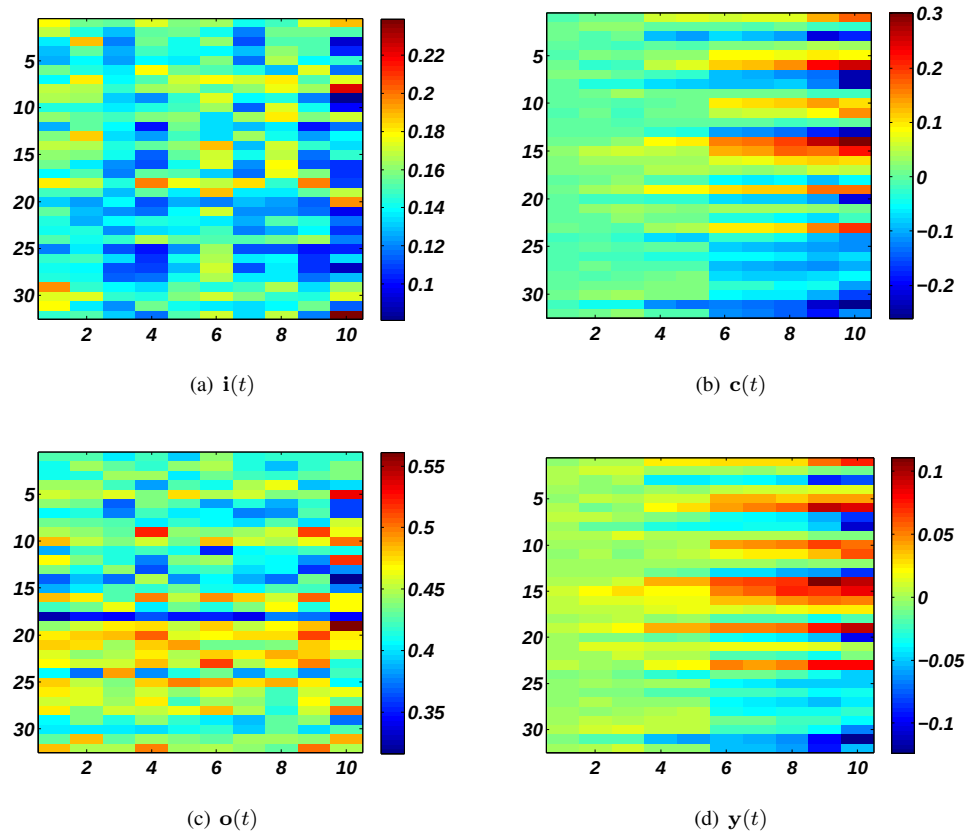
print(model.most-similar('infection')):

[(u'infections', 0.9698576927185059),
(u'treatment', 0.9143450856208801),
(u'symptoms', 0.9138627052307129),
(u'disease', 0.9100595712661743),
(u'palpitations', 0.9083651304244995),
(u'pneumonia', 0.9073051810264587),
(u'medical', 0.9043352603912354),
(u'abdomen', 0.9034136533737183),
(u'medlineplus', 0.9032401442527771),
(u'gout', 0.9027985334396362)]

As it is observed from the resulting words, the trained model is a meaningful model and can recognise semantic similarity.

APPENDIX G
DIAGRAM OF THE PROPOSED MODEL

To clarify the difference between the proposed method and the general sentence embedding methods, in this section we present a diagram illustrating the training procedure of the proposed model. It is presented in Fig. 13. In this figure n is the number of negative



(a) $\mathbf{i}(t)$ (b) $\mathbf{c}(t)$

(c) $\mathbf{o}(t)$ (d) $\mathbf{y}(t)$

Fig. 10. Document: "*how do you paint a bathtub and what paint should . . .*"

TABLE X
KEYWORD EXTRACTION FOR DOCUMENT: "*how do you paint a bathtub and what paint should . . .*"

	how	*do*	*you*	*paint*	*a*	*bathtub*	*and*	*what*	*paint*	*should you . . .*
Number of assigned cells out of 10 Left to Right	-	1	1	**7**	0	**9**	2	3	**8**	4
Number of assigned cells out of 10 Right to Left	**5**	**9**	**5**	**4**	**8**	**4**	**5**	**5**	**9**	-

(unclicked) documents. The other parameters in this figure are similar to those used in Fig. 2 and Fig. 3 of the paper.

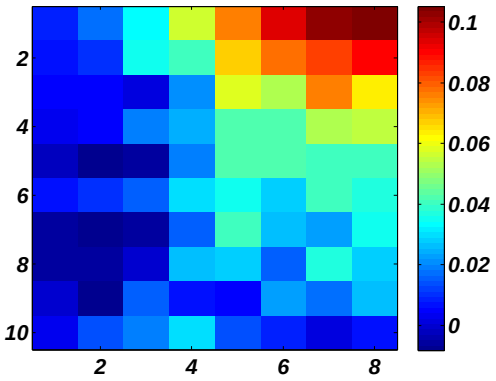


Fig. 11. Query: "*how to fix bath tub wont turn off*"

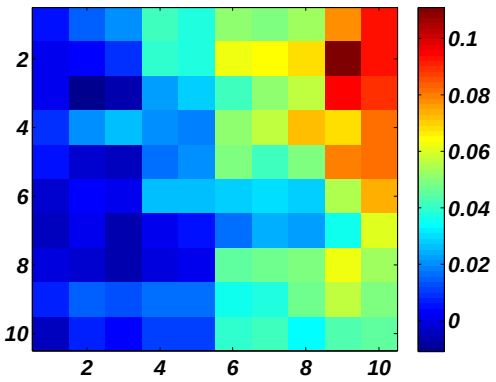


Fig. 12. Document: "*how do you paint a bathtub and what paint should . . .*"

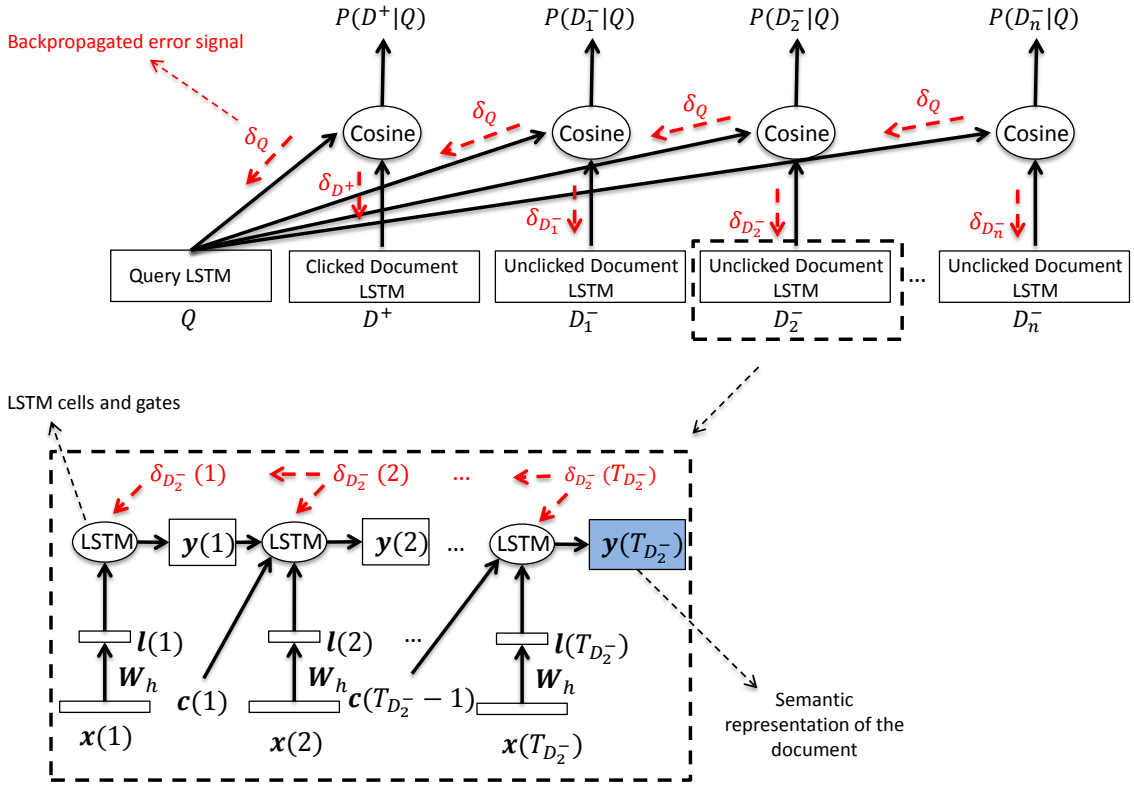


Fig. 13. Architecture of the proposed method.