Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks

Aliaksei Severyn*
Google Inc.
aseveryn@gmail.com

Alessandro Moschitti†
Qatar Computing Research Institute
amoschitti@qf.org.qa

ABSTRACT

Learning a similarity function between pairs of objects is at the core of learning to rank approaches. In information retrieval tasks we typically deal with query-document pairs, in question answering – question-answer pairs. However, before learning can take place, such pairs needs to be mapped from the original space of symbolic words into some feature space encoding various aspects of their relatedness, e.g. lexical, syntactic and semantic. Feature engineering is often a laborious task and may require external knowledge sources that are not always available or difficult to obtain. Recently, deep learning approaches have gained a lot of attention from the research community and industry for their ability to automatically learn optimal feature representation for a given task, while claiming state-of-the-art performance in many tasks in computer vision, speech recognition and natural language processing. In this paper, we present a convolutional neural network architecture for reranking pairs of short texts, where we learn the optimal representation of text pairs and a similarity function to relate them in a supervised way from the available training data. Our network takes only words in the input, thus requiring minimal preprocessing. In particular, we consider the task of reranking short text pairs where elements of the pair are sentences. We test our deep learning system on two popular retrieval tasks from TREC: Question Answering and Microblog Retrieval. Our model demonstrates strong performance on the first task beating previous state-of-the-art systems by about 3% absolute points in both MAP and MRR and shows comparable results on tweet reranking, while enjoying the benefits of no manual feature engineering and no additional syntactic parsers.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: H.3.3 Information Search and Retrieval; I.5.1 [**Pattern Recognition**]: Models—*Neural nets*

Keywords

Convolutional neural networks; learning to rank; Question Answering; Microblog search

*The work was carried out at University of Trento.
†Professor at University of Trento, DISI.

SIGIR'15, August 09 - 13, 2015, Santiago, Chile.

DOI: http://dx.doi.org/10.1145/2766462.2767738.

1. INTRODUCTION

Encoding query-document pairs into discriminative feature vectors that are input to a learning-to-rank algorithm is a critical step in building an accurate reranker. The core assumption is that relevant documents have high semantic similarity to the queries and, hence, the main effort lies in mapping a query and a document into a joint feature space where their similarity can be efficiently established.

The most widely used approach is to encode input text pairs using many complex lexical, syntactic and semantic features and then compute various similarity measures between the obtained representations. For example, in answer passage reranking [31] employ complex linguistic features, modelling syntactic and semantic information as bags of syntactic and semantic role dependencies and build similarity and translation models over these representations.

However, the choice of representations and features is a completely empirical process, driven by the intuition, experience and domain expertise. Moreover, although using syntactic and semantic information has been shown to improve performance, it can be computationally expensive and require a large number of external tools — syntactic parsers, lexicons, knowledge bases, etc. Furthermore, adapting to new domains requires additional effort to tune feature extraction pipelines and adding new resources that may not even exist.

Recently, it has been shown that the problem of semantic text matching can be efficiently tackled using distributional word matching, where a large number of lexical semantic resources are used for matching questions with a candidate answer [33].

Deep learning approaches generalize the distributional word matching problem to matching sentences and take it one step further by learning the optimal sentence representations for a given task. Deep neural networks are able to effectively capture the compositional process of mapping the meaning of individual words in a sentence to a continuous representation of the sentence. In particular, it has been recently shown that convolutional neural networks are able to efficiently learn to embed input sentences into low-dimensional vector space preserving important syntactic and semantic aspects of the input sentence, which leads to state-of-the-art results in many NLP tasks [18, 19, 38]. Perhaps one of the greatest advantages of deep neural networks is that they are trained in an end-to-end fashion, thus removing the need for manual feature engineering and greatly reducing the need for adapting to new tasks and domains.

In this paper, we describe a novel deep learning architecture for reranking short texts, where questions and documents are limited to a single sentence. The main building blocks of our architecture are two distributional sentence models based on convolutional neural networks. These underlying sentence models work in parallel, mapping queries and documents to their distributional vectors, which are then used to learn the semantic similarity between them.

The distinctive properties of our model are: (i) we use a state-of-the-art distributional sentence model for learning to map input sentences to vectors, which are then used to measure the similarity between them; (ii) our model encodes query-document pairs in a rich representation using not only their similarity score but also their intermediate representations; (iii) the architecture of our network makes it straightforward to include any additional similarity features to the model; and finally (iv) our model does not require manual feature engineering or external resources. We only require to initialize word embeddings from some large unsupervised corpora [1]

Our sentence model is based on a convolutional neural network architecture that has recently showed state-of-the-art results on many NLP sentence classification tasks [18, 19]. However, our model uses it only to generate intermediate representation of input sentences for computing their similarity. To compute the similarity score we use an approach used in the deep learning model of [38], which recently established new state-of-the-art results on answer sentence selection task. However, their model operates only on unigram or bigrams, while our architecture learns to extract and compose n-grams of higher degrees, thus allowing for capturing longer range dependencies. Additionally, our architecture uses not only the intermediate representations of questions and answers to compute their similarity but also includes them in the final representation, which constitutes a much richer representation of the question-answer pairs. Finally, our model is trained end-to-end, while in [38] the output of the deep learning model is used to learn a logistic regression classifier.

We test our model on two popular retrieval tasks from TREC: answer sentence selection and Microblog retrieval. Our model shows a considerable improvement on the first task beating recent state-of-the-art system. On the second task, our model demonstrates that previous state-of-the-art retrieval systems can benefit from using our deep learning model.

In the following, we give a problem formulation and provide a brief overview of learning to rank approaches. Next, we describe our deep learning model and describe our experiments.

2. LEARNING TO RANK

This section briefly describes the problem of reranking text pairs which encompasses a large set of tasks in IR, e.g., answer sentence selection in question answering, microblog retrieval, etc. We argue that deriving an efficient representation of query-document pairs required to train a learning to rank model plays an important role in training an accurate reranker.

2.1 Problem formulation

The most typical setup in supervised learning to rank tasks is as follows: we are given a set of retrieved lists, where each query $\mathbf{q}_i \in \mathbf{Q}$ comes together with its list of candidate documents $\mathbf{D}_i = \{\mathbf{d}_{i_1}, \mathbf{d}_{i_2}, ..., \mathbf{d}_{i_n}\}$. The candidate set comes with their relevancy judgements $\{y_{i_1}, y_{i_2}, \ldots, y_{i_n}\}$, where documents that are relevant have labels equal to 1 (or higher) and 0 otherwise. The goal is to build a model that for each query $\mathbf{q}_i$ and its candidate list $\mathbf{D}_i$ generates an optimal ranking $\mathbf{R}$, s.t. relevant documents appear at the top of the list.

More formally, the task is to learn a ranking function:

$$h(\mathbf{w}, \psi(\mathbf{q}_i, \mathbf{D}_i)) \rightarrow \mathbf{R},$$

where function $\psi(\cdot)$ maps query-document pairs to a feature vector representation where each component reflects a certain type of similarity, e.g., lexical, syntactic, and semantic. The weight vector $\mathbf{w}$ is a parameter of the model and is learned during the training.

2.2 Learning to Rank approaches

There are three most common approaches in IR to learn the ranking function h, namely, *pointwise*, *pairwise* and *listwise*.

Pointwise approach is perhaps the most simple way to build a reranker where the training instances are triples $(\mathbf{q}_i, \mathbf{d}_{ij}, y_{ij})$ and it is enough to train a binary classifier: $h(\mathbf{w}, \psi(\mathbf{q}_i, \mathbf{d}_{ij})) \rightarrow y_{ij}$, where ψ maps query-document pair to a feature vector and $\mathbf{w}$ is a vector of model weights.

The decision function $h(\cdot)$ typically takes a linear form simply computing a dot product between the model weights $\mathbf{w}$ and a feature representation of a pair generated by $\psi(\cdot)$. At test time, the learned model is used to classify unseen pairs $(\mathbf{q}_i, \mathbf{d}_{ij})$, where the raw scores are used to establish the global rank $\mathbf{R}$ of the documents in the retrieved set. This approch is widely used in practice because of its simplicity and effectiveness.

A more advanced approaches to reranking, is *pairwise*, where the model is explicitly trained to score correct pairs higher than incorrect pairs with a certain margin:

$$h(\mathbf{w}, \psi(\mathbf{q}_i, \mathbf{d}_{ij})) \geq h(\mathbf{w}, \psi(\mathbf{q}_i, \mathbf{d}_{ik})) + \epsilon,$$

where document $\mathbf{d}_{ij}$ is relevant and $\mathbf{d}_{ik}$ is not. Conceptually similar to the *pointwise* method described above, the *pairwise* approach exploits more information about the ground truth labelling of the input candidates. However, it requires to consider a larger number of training instances (potentially quadratic in the size of the candidate document set) than the *pointwise* method, which may lead to slower training times. Still both *pointwise* and *pairwise* approaches ignore the fact that ranking is a prediction task on a list of objects.

The third method, referred to as a *listwise* approach [6], treats a query with its list of candidates as a single instance in learning, thus able to capture considerably more information about the ground truth ordering of input candidates.

While *pairwise* and *listwise* approaches claim to yield better performance, they are more complicated to implement and less effective train. Most often, producing a better representation $\psi()$ that encodes various aspects of similarity between the input query-document pairs plays a far more important role in training an accurate reranker than choosing between different ranking approaches. Hence, in this paper we adopt a simple *pointwise* method to reranking and focus on modelling a rich representation of query-document pairs using deep learning approaches which is described next.

3. OUR DEEP LEARNING MODEL

This section explains our deep learning model for reranking short text pairs. Its main building blocks are two distributional *sentence models* based on convolutional neural networks (ConvNets). These underlying sentence models work in parallel mapping queries and documents to their distributional vectors, which are then used to learn the semantic similarity between them.

In the following, we first describe our *sentence model* for mapping queries and documents to their intermediate representations and then describe how they can be used for learning semantic matching between input query-document pairs.

3.1 Sentence model

The architecture of our ConvNet for mapping sentences to feature vectors is shown on Fig. 1. It is mainly inspired by the architectures used in [18, 19] for performing various sentence classifica-

[1]Given a large training corpora our network can also optimize the embeddings directly for the task, thus omitting the need to pre-train the embeddings.

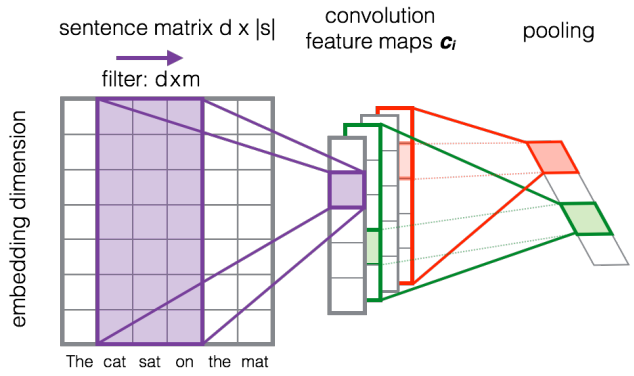

Figure 1: Our sentence model for mapping input sentences to their intermediate feature representations.

tion tasks. However, different from previous work the goal of our distributional sentence model is to learn good intermediate representations of the queries and documents, which are then used for computing their semantic matching.

Our network is composed of a single *wide* convolutional layer followed by a non-linearity and simple *max* pooling.

The input to the network are raw words that need to be translated into real-valued feature vectors to be processed by subsequent layers of the network. In the following we give a brief explanation of the main components of our convolutional neural network: sentence matrix, activations, convolutional and pooling layers.

3.1.1 Sentence matrix

The input to our sentence model is a sentence $\mathbf{s}$ treated as a sequence of words: $[w_1, .., w_{|s|}]$, where each word is drawn from a vocabulary V. Words are represented by distributional vectors $\mathbf{w} \in \mathbb{R}^d$ looked up in a word embeddings matrix $\mathbf{W} \in \mathbb{R}^{d \times |V|}$ which is formed by concatenating embeddings of all words in V. For convenience and ease of lookup operations in $\mathbf{W}$, words are mapped to integer indices $1, \ldots, |V|$.

For each input sentence $\mathbf{s}$ we build a sentence matrix $\mathbf{S} \in \mathbb{R}^{d \times |s|}$, where each column i represents a word embedding $\mathbf{w}_i$ at the corresponding position i in a sentence (see Fig. 1):

$$\mathbf{S} = \begin{bmatrix} | & | & | \\ \mathbf{w}_1 & \ldots & \mathbf{w}_{|s|} \\ | & | & | \end{bmatrix}$$

To learn to capture and compose features of individual words in a given sentence from low-level word embeddings into higher level semantic concepts, the neural network applies a series of transformations to the input sentence matrix $\mathbf{S}$ using convolution, non-linearity and pooling operations, which we describe next.

3.1.2 Convolution feature maps

The aim of the convolutional layer is to extract patterns, i.e., discriminative word sequences found within the input sentences that are common throughout the training instances.

More formally, the convolution operation $*$ between two vectors $\mathbf{s} \in \mathbb{R}^{|s|}$ and $\mathbf{f} \in \mathbb{R}^m$ (called a filter of size m) results in a vector $\mathbf{c} \in \mathbb{R}^{|s|+m-1}$ where each component is as follows:

$$\mathbf{c}_i = (\mathbf{s} * \mathbf{f})_i = \mathbf{s}_{[i-m+1:i]}^T \cdot \mathbf{f} = \sum_{k=i}^{i+m-1} s_k f_k \qquad (1)$$

The range of allowed values for i defines two types of convolution: *narrow* and *wide*. The *narrow* type restricts i to be in the range $[1, |s| - m + 1]$, which in turn restricts the filter width to be $\leq |s|$. To compute the *wide* type of convolution i ranges from 1 to $|s|$ and sets no restrictions on the size of m and s. The benefits of one type of convolution over the other when dealing with text are discussed in detail in [18]. In short, the *wide* convolution is able to better handle words at boundaries giving equal attention to all words in the sentence, unlike in *narrow* convolution, where words close to boundaries are seen fewer times. More importantly, *wide* convolution also guarantees to always yield valid values even when $\mathbf{s}$ is shorter than the filter size m. Hence, we use *wide* convolution in our sentence model. In practice, to compute the *wide* convolution it is enough to pad the input sequence with $m - 1$ zeros from left and right.

Given that the input to our ConvNet are sentence matrices $\mathbf{S} \in \mathbb{R}^{d \times |s|}$, the arguments of Eq. 1 are matrices and a convolution filter is also a matrix of weights: $\mathbf{F} \in \mathbb{R}^{d \times m}$. Note that the convolution filter is of the same dimensionality d as the input sentence matrix. As shown in Fig. 1, it slides along the column dimension of $\mathbf{S}$ producing a vector $\mathbf{c} \in \mathbb{R}^{|s|-m+1}$ in output. Each component c_i is the result of computing an element-wise product between a column slice of $\mathbf{S}$ and the filter matrix $\mathbf{F}$, which is then flattened and summed producing a single value.

It should be noted that an alternative way of computing a convolution was explored in [18], where a series of convolutions are computed between each row of a sentence matrix and a corresponding row of the filter matrix. Essentially, it is a vectorized form of 1d convolution applied between corresponding rows of $\mathbf{S}$ and $\mathbf{F}$. As a result, the output feature map is a matrix $\mathbf{C} \in \mathbb{R}^{d \times |s|-m+1}$ rather than a vector as above. While, intuitively, being a more general way to process the input matrix $\mathbf{S}$, where individual filters are applied to each respective dimension, it introduces more parameters to the model and requires a way to reduce the dimensionality of the resulting feature map. To address this issue, the authors apply a folding operation, which sums every two rows element-wise, thus effectively reducing the size of the representation by 2.

So far we have described a way to compute a convolution between the input sentence matrix and a single filter. In practice, deep learning models apply a set of filters that work in parallel generating muliple feature maps (also shown on Fig. 1). The resulting filter bank $\mathbf{F} \in \mathbb{R}^{n \times d \times m}$ produces a set of feature maps of dimension $n \times (|s| - m + 1)$.

In practice, we also add a bias vector[2] $\mathbf{b} \in \mathbb{R}^n$ to the result of a convolution – a single b_i value for each feature map $\mathbf{c}_i$.

3.1.3 Activation units

To allow the network learn non-linear decision boundaries, each convolutional layer is typically followed by a non-linear activation function $\alpha()$ applied element-wise to the output of the preceding layer. Among the most common choices of activation functions are the following: sigmoid (or logistic), hyperbolic tangent *tanh*, and a rectified linear (ReLU) function defined as simply $max(0, \mathbf{x})$ to ensure that feature maps are always positive. The choice of activation function has been shown to affect the convergence rate and the quality of obtained the solution. In particular, [22] shows that rectified linear unit has significant benefits over sigmoid and $tanh$ overcoming some of the their shortcomings.

3.1.4 Pooling

The output from the convolutional layer (passed through the activation function) are then passed to the pooling layer, whose goal

[2]bias is needed to allow the network learn an appropriate threshold

is to aggregate the information and reduce the representation. The result of the pooling operation is:

$$\mathbf{c}_{\text{pooled}} = \begin{bmatrix} \text{pool}(\alpha(\mathbf{c}_1 + b_1 * \mathbf{e})) \\ \dots \\ \text{pool}(\alpha(\mathbf{c}_n + b_n * \mathbf{e})) \end{bmatrix}$$

where $\mathbf{c}_i$ is the ith convolutional feature map with added bias (the bias is added to each element of $\mathbf{c}_i$ and $\mathbf{e}$ is a unit vector of the same size as $\mathbf{c}_i$) and passed through the activation function $\alpha()$.

There are two conventional choices for the pool$(\cdot)$ operation: average and max. Both operations apply to columns of the feature map matrix, by mapping them to a single value: pool$(\mathbf{c}_i) : \mathbb{R}^{1\times(|s|+m-1)} \rightarrow \mathbb{R}$. This is also demonstrated in Fig. 1.

Both *average* and *max* pooling methods exhibit certain disadvantages: in average pooling, all elements of the input are considered, which may weaken strong activation values. This is especially critical with *tanh* non-linearity, where strong positive and negative activations can cancel each other out.

The *max* pooling is used more widely and does not suffer from the drawbacks of average pooling. However, as shown in [40], it can lead to strong overfitting on the training set and, hence, poor generalization on the test data. To mitigate the overfitting issue of max pooling several variants of stochastic pooling have been proposed in [40].

Recently, *max* pooling has been generalized to k-max pooling [18], where instead of a single max value, k values are extracted in their original order. This allows for extracting several largest activation values from the input sentence. As a consequence deeper architectures with several convolutional layers can be used. In [18], the authors also propose dynamic k-max pooling, where the value of k depends on the sentence size and the level in the convolution hierarchy [18].

Convolutional layer passed through the activation function together with pooling layer acts as a non-linear feature extractor. Given that multiple feature maps are used in parallel to process the input, deep learning networks are able to build rich feature representations of the input.

This ends the description of our sentence model. In the following we present our deep learning network for learning to match short text pairs.

3.2 Our architecture for matching text pairs

The architecture of our model for matching query-document pairs is presented in Fig. 2. Our sentence models based on ConvNets (described in Sec. 3.1) learn to map input sentences to vectors, which can then be used to compute their similarity. These are then used to compute a query-document similarity score, which together with the query and document vectors are joined in a single representation.

In the following we describe how the intermediate representations produced by the sentence model can be used to compute query-document similarity scores and give a brief explanation of the remaining layers, e.g. hidden and softmax, used in our network.

3.2.1 Matching query and documents

Given the output of our sentence ConvNets for processing queries and documents, their resulting vector representations $\mathbf{x}_q$ and $\mathbf{x}_d$, can be used to compute a query-document similarity score. We follow the approach of [2] that defines the similarity between $\mathbf{x}_q$ and $\mathbf{x}_d$ vectors as follows:

$$sim(\mathbf{x}_q, \mathbf{x}_d) = \mathbf{x}_q^T \mathbf{M} \mathbf{x}_d, \tag{2}$$

where $\mathbf{M} \in \mathbb{R}^{d\times d}$ is a similarity matrix. The Eq. 2 can be viewed as a model of the noisy channel approach from machine translation, which has been widely used as a scoring model in information retrieval and question answering [13]. In this model, we seek a transformation of the candidate document $\mathbf{x}'_d = \mathbf{M}\mathbf{x}_d$ that is the closest to the input query $\mathbf{x}_q$. The similarity matrix $\mathbf{M}$ is a parameter of the network and is optimized during the training.

3.2.2 Hidden layers

The hidden layer computes the following transformation:

$$\alpha(\mathbf{w}_h \cdot \mathbf{x} + b),$$

where $\mathbf{w}_h$ is the weight vector of the hidden layer and $\alpha()$ is the non-linearity. Our model includes an additional hidden layer right before the softmax layer (described next) to allow for modelling interactions between the components of the intermediate representation.

3.2.3 Softmax

The output of the penultimate convolutional and pooling layers is flattened to a dense vector $\mathbf{x}$, which is passed to a fully connected softmax layer. It computes the probability distribution over the labels:

$$p(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^T \theta_\mathbf{j}}}{\sum_{k=1}^{K} e^{\mathbf{x}^T \theta_\mathbf{k}}},$$

where θ_k is a weight vector of the k-th class. $\mathbf{x}$ can be thought of as a final abstract representation of the input example obtained by a series of transformations from the input layer through a series of convolutional and pooling operations.

3.2.4 The information flow

Here we provide a full description of our deep learning network (shown on Fig. 2) that maps input sentences to class probabilities.

The output of our sentence models (Sec. 3.1) are distributional representations of a query $\mathbf{x}_q$ and a document $\mathbf{x}_d$. These are then matched using a similarity matrix $\mathbf{M}$ according to Eq. 2. This produces a single score x_{sim} capturing various aspects of similarity (syntactic and semantic) between the input queries and documents. Note that it is also straight-forward to add additional features $\mathbf{x}_{\text{feat}}$ to the model.

The join layer concatenates all intermediate vectors, the similarity score and any additional features into a single vector:

$$\mathbf{x}_{\text{join}} = [\mathbf{x}_q^T; x_{\text{sim}}; \mathbf{x}_d^T; \mathbf{x}_{\text{feat}}^T]$$

This vector is then passed through a fully connected hidden layer, which allows for modelling interactions between the components of the joined representation vector. Finally, the output of the hidden layer is further fed to the softmax classification layer, which generates a distribution over the class labels.

3.3 Training

The model is trained to minimise the cross-entropy cost function:

$$\begin{aligned} \mathcal{C} &= -\log \textstyle\prod_{i=1}^{N} p(y_i|\mathbf{q}_i, \mathbf{d}_i) + \lambda\|\theta\|_2^2 \\ &= -\textstyle\sum_{i=1}^{N} [y_i \log \mathbf{a}_i + (1-y_i)\log(1-\mathbf{a}_i)] + \lambda\|\theta\|_2^d, \end{aligned} \tag{3}$$

where $\mathbf{a}$ is the output from the softmax layer. θ contains all the parameters optimized by the network:

$$\theta = \{\mathbf{W}; \mathbf{F}_q; \mathbf{b}_q; \mathbf{F}_d; \mathbf{b}_d; \mathbf{M}; \mathbf{w}_h; b_h; \mathbf{w}_s; b_s\},$$

namely the word embeddings matrix $\mathbf{W}$, filter weights and biases of the convolutional layers, similarity matrix $\mathbf{M}$, weights and biases of the hidden and softmax layers.

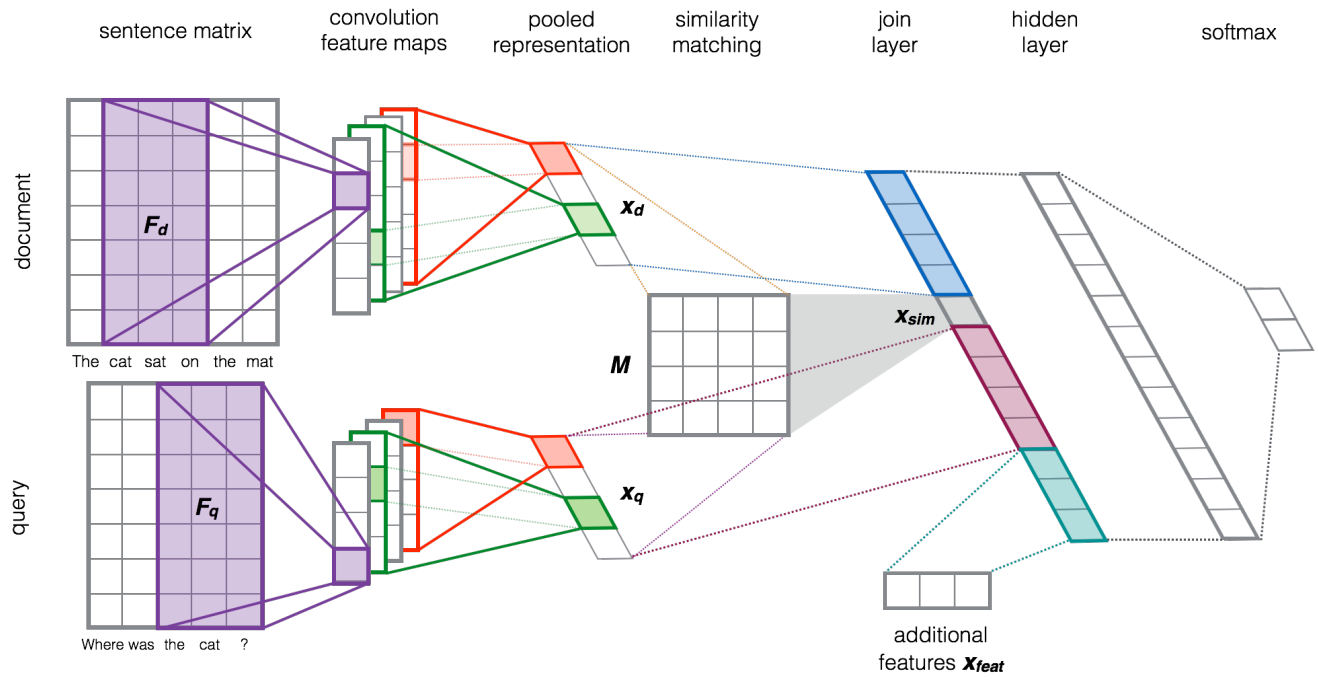

Figure 2: Our deep learning architecture for reranking short text pairs.

The parameters of the network are optimized with stochastic gradient descent (SGD) using backpropogation algorithm to compute the gradients. To speedup the convergence rate of SGD various modifications to the update rule have been proposed: *momentum*, *Adagrad* [12], *Adadelta* [39], etc. *Adagrad* scales the learning rate of SGD on each dimension based on the $l2$ norm of the history of the error gradient. *Adadelta* uses both the error gradient history like *Adagrad* and the weight update history. It has the advantage of not having to set a learning rate at all.

3.4 Regularization

While neural networks have a large capacity to learn complex decision functions they tend to easily overfit especially on small and medium sized datasets. To mitigate the overfitting issue we augment the cost function with l_2-norm regularization terms for the parameters of the network.

We also experiment with another popular and effective technique to improve regularization of the NNs — dropout [30]. Dropout prevents feature co-adaptation by setting to zero (dropping out) a portion of hidden units during the forward phase when computing the activations at the softmax output layer. As suggested in [14] dropout acts as an approximate model averaging.

4. EXPERIMENTS AND EVALUATION

We evaluate our deep learning model on two popular retrieval benchmarks from TREC: answer sentence selection and TREC microblog retrieval.

4.1 Training and hyperparameters

The parameters of our deep learning model were (chosen on a dev set of the answer sentence selection dataset) as follows: the width m of the convolution filters is set to 5 and the number of convolutional feature maps is 100. We use ReLU activation function and a simple max-pooling. The size of the hidden layer is equal to the size of the $\boldsymbol{x}_{\text{join}}$ vector obtained after concatenating query and document vectors from the distributional models, similarity score and additional features (if used).

To train the network we use stochastic gradient descent with shuffled mini-batches. We eliminate the need to tune the learning rate by using the Adadelta update rule [39]. The batch size is set to 50 examples. The network is trained for 25 epochs with early stopping, i.e., we stop the training if no update to the best accuracy on the dev set has been made for the last 5 epochs. The accuracy computed on the dev set is the MAP score. At test time we use the parameters of the network that were obtained with the best MAP score on the development (dev) set, i.e., we compute the MAP score after each 10 mini-batch updates and save the network parameters if a new best dev MAP score was obtained. In practice, the training converges after a few epochs. We set a value for L2 regularization term to $1e-5$ for the parameters of convolutional layers and $1e-4$ for all the others. The dropout rate is set to $p = 0.5$.

4.2 Word embeddings

While our model allows for learning the word embeddings directly for a given task, we keep the word matrix parameter $\boldsymbol{W}$ static. This is due to a common experience that a minimal size of the dataset required for tuning the word embeddings for a given task should be at least in the order of hundred thousands, while in our case the number of query-document pairs is one order of magnitude smaller. Hence, similar to [11, 19, 38] we keep the word embeddings fixed and initialize the word matrix $\boldsymbol{W}$ from an unsupervised neural language model. We choose the dimensionality of our word embeddings to be 50 to be on the line with the deep learning model of [38].

4.3 Size of the model

Given that the dimensionality of the word embeddings is 50, the number of parameters in the convolution layer of each sentence

model is $100 \times 5 \times 50$. Hence, the total number of parameters in each of the two convolutional networks that map sentences to vectors is 25k. The similarity matrix is $\boldsymbol{M} \in \mathbb{R}^{100\times100}$, which adds another 10k parameters to the model. The fully connected hidden layer is and a softmax add about 40k parameters. Hence the total number of parameters in the network is about 100k.

5. ANSWER SENTENCE SELECTION

Our first experiment is on answer sentence selection dataset, where answer candidates are limited to a single sentence. Given a question with its list of candidate answers the task is to rank the candidate answers based on their relatedness to the question.

5.1 Experimental setup

Data and setup. We test our model on the manually curated TREC QA dataset[3] from Wang et al. [36], which appears to be one of the most widely used benchmarks for answer reranking. The dataset contains a set of factoid questions, where candidate answers are limited to a single sentence. The set of questions are collected from TREC QA tracks 8-13. The manual judgement of candidate answer sentences is provided for the entire TREC 13 set and for the first 100 questions from TREC 8-12. The motivation behind this annotation effort is that TREC provides only the answer patterns to identify if a given passage contains a correct answer key or not. This results in many unrelated candidate answers marked as correct simply because regular expressions cannot always match the correct answer keys.

To enable direct comparison with the previous work, we use the same train, dev and test sets. Table 1 summarizes the datasets used in our experiments. An additional training set TRAIN-ALL provided by Wang et. al [36] contains 1,229 questions from the entire TREC 8-12 collection and comes with automatic judgements. This set represents a more noisy setting, nevertheless, it provides many more QA pairs for learning.

Word embeddings. We initialize the word embeddings by running `word2vec` tool [20] on the English Wikipedia dump and the AQUAINT corpus[4] containing roughly 375 million words. To train the embeddings we use the skipgram model with window size 5 and filtering words with frequency less than 5. The resulting model contains 50-dimensional vectors for about 3.5 million words. Embeddings for words not present in the `word2vec` model are randomly initialized with each component sampled from the uniform distribution $U[-0.25, 0.25]$.

We minimally preprocess the data only performing tokenization and lowercasing all words. To reduce the size of the resulting vocabulary V, we also replace all digits with 0. The size of the word vocabulary V for experiments using TRAIN set is 17,023 with approximately 95% of words initialized using `wor2vec` embeddings and the remaining 5% words are initialized at random as described in Sec. 4.2. For the TRAIN-ALL setting the $|V| = 56,953$ with 85% words found in the `word2vec` model.

Additional features. Given that a certain percentage of the words in our word embedding matrix are initialized at random (about 15% for the TRAIN-ALL) and a relatively small number of QA pairs prevents the network to directly learn them from the training data, similarity matching performed by the network will be suboptimal between many question-answer pairs.

Additionally, even for the words found in the word matrix, as noted in [38], one of the weaknesses of approaches relying on distributional word vectors is their inability to deal with numbers and proper nouns. This is especially important for factoid question answering, where most of the questions are of type *what, when, who* that are looking for answers containing numbers or proper nouns.

To mitigate the above two issues, we follow the approach in [38] and include additional features establishing relatedness between question-answer pairs. In particular, we compute word overlap measures between each question-answer pair and include it as an additional feature vector $\boldsymbol{x}_{\text{feat}}$ in our model. This feature vector contains only four features: word overlap and IDF-weighted word overlap computed between all words and only non-stop words. Computing these features is straightforward and does not require additional pre-processing or external resources.

Evaluation. The two metrics used to evaluate the quality of our model are Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR), which are common in Information Retrieval and Question Answering.

MRR is computed as follows: $MRR = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{rank(q)}$, where *rank(q)* is the position of the first correct answer in the candidate list. MRR is only looking at the rank of the first correct answer, hence it is more suitable in cases where for each question there is only a single correct answer. Differently, MAP examines the ranks of all the correct answers. It is computed as the mean over the average precision scores for each query $q \in Q$: $\frac{1}{Q}\sum_{q=1}^{Q} AveP(q)$. We use the official `trec_eval` scorer to compute the above metrics.

Table 1: Summary of TREC QA datasets for answer reranking.

Data	# Questions	# QA pairs	% Correct
TRAIN-ALL	1,229	53,417	12.0%
TRAIN	94	4,718	7.4%
DEV	82	1,148	19.3%
TEST	100	1,517	18.7%

5.2 Results and discussion

We report the results of our deep learning model on the TRAIN and TRAIN-ALL sets also when additional word overlap features are used. Additionally, we report the results from a recent deep learning system in [38] that has established the new state-of-the-art results in the same setting.

Table 2 summarises the results for the setting when the network is trained using only input question-answer pairs without using any additional features. As we can see our deep learning architecture demonstrates a much stronger performance compared to the system in [38]. The deep learning model from [38], similarly to ours, relies on a convolutional neural network to learn intermediate representations. However, their convolutional neural network operates only on unigram or bigrams, while in our architecture we use a larger width of the convolution filter, thus allowing for capturing longer range dependencies. Additionally, along with the question-answer similarity score, our architecture includes intermediate representations of the question and the answer, which together constitute a much richer representation. This results in a large improvement of about 8% absolute points in MAP for TRAIN and almost 10% when trained with more data from TRAIN-ALL. This emphasizes the importance of learning high quality sentence models.

Table 3 provides the results when additional word *overlap* features are added to the model. Simple word overlap features help to improve the question-answer matching. Our model shows an improvement of about a significant improvement over previous state-

[3] http://cs.stanford.edu/people/mengqiu/data/qg-emnlp07-data.tgz
[4] https://catalog.ldc.upenn.edu/LDC2002T31

Table 2: Results on TRAIN and TRAIN-ALL from Trec QA.

Model	MAP	MRR
TRAIN		
Yu et al. [38] (unigram)	.5387	.6284
Yu et al. [38] (bigram)	.5476	.6437
Our model	**.6258**	**.6591**
TRAIN-ALL		
Yu et al. [38] (unigram)	.5470	.6329
Yu et al. [38] (bigram)	.5693	.6613
Our model	**.6709**	**.7280**

Table 3: Results on TREC QA when augmenting the deep learning model with word overlap features.

Model	MAP	MRR
TRAIN		
Yu et al. [38] (unigram)	.6889	.7727
Yu et al. [38] (bigram)	.7058	.7800
Our model	**.7329**	**.7962**
TRAIN-ALL		
Yu et al. [38] (unigram)	.6934	.7677
Yu et al. [38] (bigram)	.7113	.7846
Our model	**.7459**	**.8078**

Table 4: Survey of the results on the QA answer selection task.

Model	MAP	MRR
Wang et al. (2007) [36]	.6029	.6852
Heilman and Smith (2010) [15]	.6091	.6917
Wang and Manning (2010) [35]	.5951	.6951
Yao et al. (2013) [37]	.6307	.7477
Severyn & Moschitti (2013) [26]	.6781	.7358
Yih et al. (2013) [33]	.7092	.7700
Yu et al. (2014) [38]	.7113	.7846
Our model (TRAIN)	**.7329**	**.7962**
Our model (TRAIN-ALL)	**.7459**	**.8078**

of-the-art in both MAP and MRR when training on TRAIN and TRAIN-ALL. Note that the results are significantly better than when no overlap features are used. This is possibly due to the fact that the distrubutional representations fail to establish the relatedness in some cases and simple word overlap matching can help to drive the model in the right direction.

Table 4 reports the results of the previously published systems on this task. Our model trained on a small TRAIN dataset beats all of the previous state-of-the-art systems. The improvement is further emphasized when the system is trained using more question-answer pairs from TRAIN-ALL showing an improvement of about 3% absolute points in both MAP and MRR. The results are very promising considering that our system requires no manual feature engineering (other than simple word overlap features), no expensive preprocessing using various NLP parsers, and no external semantic resources other than using pre-initialized word embeddings that can be easily trained provided a large amount of unsupervised text.

In the spirit, our system is most similar to a recent deep learning architecture from Yu et al. (2014) [38]. However, we employ a more expressive convolutional neural network for learning intermediate representations of the query and the answer. This allows for performing a more accurate matching between question-answer pairs. Additionally, our architecture includes intermediate question and answer representations in the model, which result in a richer representation of question-answer pairs. Finally, we train our system in an end-to-end fashion, while [38] use the output of their deep learning system as a feature in a logistic regression classifier.

Table 5: Summary of TREC Microblog datasets.

Data	# Topic	# Tweet pairs	% Correct	# Runs
TMB2011	49	60,129	5.1%	184
TMB2012	59	73,073	8.6%	120

6. TREC MICROBLOG RETRIEVAL

To assess the effectiveness and generality of our deep learning model for text matching, we apply it on tweet reranking task. We focus on the 2011 and 2012 editions of the ad-hoc retrieval task at TREC microblog tracks [23, 29]. We follow the setup in [27], where they represent query-tweet pairs with a shallow syntactic models to learn a tree kernel reranker. In contrast, our model does not rely on any syntactic parsers and requires virtually no preprocessing other than tokenizaiton and lower-casing. Our main research question is: Can our neural network that requires no manual feature engineering and expensive pre-processing steps improve on top of the state-of-the-art learning-to-rank and retrieval algorithms?

To answer this question, we test our model in the following settings: we treat the participant systems in the TREC microblog tasks as a black-box, and implement our model on top of them using only their raw scores (ranks) as a single feature in our model. This allows us to see whether our model is able to learn information complementary to the approaches used by such retrieval algorithms. Our setup replicates the experiments in [27] to allow for comparing to their model.

6.1 Experimental setup

Data and setup. Our dataset is the tweet corpus used in both TREC Microblog tracks in 2011 (TMB2011) and 2012 (TMB2012). It consists of 16M tweets spread over two weeks, and a set of 49 (TMB2011) and 59 (TMB2012) timestamped topics. We minimally preprocess the tweets—we normalize elongations (e.g., sooo → so), normalize URLs and author ids. Additionally, we use the system runs submitted at TMB2011 and TMB2012, which contain 184 and 120 models, respectively. This is summarized in Table 5.

Word embeddings. We used the `word2vec` tool to learn the word embeddings from the provided 16M tweet corpus, with the following setting: (i) we removed non-english tweets, which reduces the corpus to 8.4M tweets and (ii) we used the skipgram model with window size 5 and filtering words with frequency less than 5. The trained model contains 330k words. We use word embeddings of size 50 — same as for the previous task. To build the word embedding matrix $\boldsymbol{W}$, we extract the vocabulary from all tweets present in TMB2011 and TMB2012. The resulting vocabulary contains 150k words out of which only 60% are found in the word embeddings model. This is due to a very large number of misspellings and words occurring only once (hence they are filted by the `word2vec` tool). This has a negative impact on the performance of our deep learning model since around 40% of the word vectors are randomly initialized. At the same time it is not possible to tune the word embeddings on the training set, as it will overfit due to the small number of the query-tweet pairs available for training.

Training. We train our system on the runs submitted at TMB2011, and test it on the TMB2012 runs. We focus on one direction only to avoid training bias, since TMB2011 topics were already used for learning systems in TMB2012.

Submission run as a feature. We use the output of participant systems as follows: we use rank positions of each tweet rather than raw scores, since scores for each system are scaled differently, while ranks are uniform across systems. We apply the following transformation of the rank r: $1/\log(r+1)$. In the training phase, we take the top 30 systems from the TMB2011 track (in terms of P@30). For each query-tweet pair we compute the average transformed rank over the top systems. This score is then used as a single feature $\boldsymbol{x}_{\text{feat}}$ by our model. In the testing phase, we generate this feature as follows: for each participant system that we want to improve, we use the transformed rank of the query-tweet taken from their submission run.

Evaluation. We report on the official evaluation metric for the TREC 2012 Microblog track, i.e., precision at 30 (P@30), and also on mean average precision (MAP). Following [4, 23], we regard minimally and highly relevant documents as relevant and use the TMB2012 evaluation script. For significance testing, we use a pairwise t-test, where $\triangle$ and $\blacktriangle$ denote significance at $\alpha = 0.05$ and $\alpha = 0.01$, respectively. Triangles point up for improvement over the baseline, and down otherwise. We also report the improvement in the absolute rank (R) in the official TMB2012 ranking.

6.2 Results and discussion

Table 6 reports the results for re-ranking runs of the best 30 systems from TMB2012 (based on their P@30 score) when we train our system using the top 30 runs from TMB2011.

First, we note that our model improves P@30 for the majority of the systems with a relative improvement ranging from several points up to 10% with about 6% on average. This is remarkable, given that the pool of participants in TMB2012 was large, and the top systems are therefore likely to be very strong baselines.

Secondly, we note that the relative improvement of our model is on the par with the STRUCT model from [27], which relies on using syntactic parsers to train a tree kernel reranker. In contrast, our model requires no manual feature engineering and virtually no preprocessing and external resources. Similar to the observation made in [27], our model has a precision-enhancing effect. In cases where MAP drops a bit it can be seen that our model sometimes lowers relevant documents in the runs. It is possible that our model favours query-tweet pairs that exhibit semantic matching of higher quality, and that it down-ranks tweets that are of lower quality but are nonetheless relevant. Another important aspect is the fact that a large portion of the word embeddings (about 40%) used by the network are initialized at random, which has a negative impact on the accuracy of our model.

Looking at the improvement in absolute position in the official ranking (R), we see that, on average, our deep learning model boosts the absolute position in the official ranking for top 30 systems by about 7.8 positions.

All in all, the results suggest that our deep learning model with no changes in its architecture is able to capture additional information and can be useful when coupled with many state-of-the-art microblog search algorithms.

While improving the top systems from 2012 represents a challenging task, it is also interesting to assess the potential improvement for lower ranking systems. We follow [27] and report our results on the 30 systems from the middle and the bottom of the official ranking. Table 7 summarizes the average improvements for three groups of systems: top-30, middle-30, and bottom-30.

Table 7: Comparison of the averaged relative improvements for the top, middle (mid), and bottom (btm) 30 systems from TMB2012.

	STRUCT [27]		Our model	
band	MAP	P@30	MAP	P@30
top	3.3%	5.3%	2.0%	6.2%
mid	12.2%	12.9%	9.8%	13.7%
btm	22.1%	25.1%	18.7%	24.3%

We find that the improvement over underperforming systems is much larger than for stronger systems. In particular, for the bottom 30 systems, our approach achieves an average relative improvement of 20% in both MAP and P@30. The performance of our model is on the par with the STRUCT model [27].

We expect that learning word embeddings on a larger corpora such that the percentage of the words present in the word embedding matrix $\boldsymbol{W}$ should help to improve the accuracy of our system. Moreover, similar to the situation observed with answer selection experiments, we expect that using more training data would improve the generalization of our model. As one possible solution to getting more training data, it could be interesting to experiment with training our model on much larger pseudo test collections similar to the ones proposed in [4]. We leave it for the future work.

7. RELATED WORK

Our learning to rank method is based on a deep learning model for advanced text representations using distributional word embeddings. Distributional representations have a long tradition in IR, e.g., Latent Semantic Analysis [10], which more recently has also been characterized by studies on distributional models based on word similarities. Their main properties is to alleviate the problem of data sparseness. In particular, such representations can be derived with several methods, e.g., by counting the frequencies of co-occurring words around a given token in large corpora. Such distributed representations can be obtained by applying neural language models that learn word embeddings, e.g., [3] and more recently using recursive autoencoders [34], and convolutional neural networks [8].

Our application of learning to rank models concerns passage reranking. For example, [17, 24] designed classifiers of question and answer passage pairs. Several approaches were devoted to reranking passages containing definition/description, e.g., [21, 28, 31]. [1] used a cascading approach, where the ranking produced by one ranker is used as input to the next stage.

Language models for reranking were applied in [7], where answer ranks were computed based on the probabilities of bigram models generating candidate answers. Language models were also applied to definitional QA in [9, 25, 32].

Our work more directly targets the task of answer sentence selection, i.e., the task of selecting a sentence that contains the information required to answer a given question from a set of candidates (for example, provided by a search engine). In particular, the state of the art in answer sentence selection is given by Wang et al., 2007 [36], who use quasi-synchronous grammar to model relations between a question and a candidate answer with the syntactic transformations. Heilman & Smith, 2010 [15] develop an improved Tree Edit Distance (TED) model for learning tree transformations in a q/a pair. They search for a good sequence of tree edit operations using complex and computationally expensive Tree Kernel-based heuristic. Wang & Manning, 2010 [35] develop a probabilistic

Table 6: System performance on the top 30 runs from TMB2012, using the top 10, 20 or 30 runs from TMB2011 for training.

		TMB2012		STRUCT [27]			Our model		
#	runs	MAP	P@30	MAP	P@30	R%	MAP	P@30	R%
1	hitURLrun3	0.3469	0.4695	0.3328 (-4.1%)$^{\nabla}$	0.4774 (1.7%)	0	0.3326 (-4.1%)$^{\nabla}$	0.4836 (3.0%)	0
2	kobeMHC2	0.3070	0.4689	0.3037 (-1.1%)	0.4768 (1.7%)	1	0.3052 (-0.6%)	0.4899 (4.5%)$^{\triangle}$	1
3	kobeMHC	0.2986	0.4616	0.2965 (-0.7%)	0.4718 (2.2%)	2	0.2999 (0.4%)	0.4830 (4.6%)$^{\triangle}$	2
4	uwatgclrman	0.2836	0.4571	0.2995 (5.6%)$^{\blacktriangle}$	0.4712 (3.1%)$^{\triangle}$	3	0.2738 (-3.5%)$^{\nabla}$	0.4516 (-1.2%)	-1
5	kobeL2R	0.2767	0.4429	0.2744 (-0.8%)	0.4463 (0.8%)	0	0.2677 (-3.3%)$^{\nabla}$	0.4409 (-0.5%)	-2
6	hitQryFBrun4	0.3186	0.4424	0.3118 (-2.1%)	0.4554 (2.9%)	2	0.3220 (1.1%)$^{\blacktriangle}$	0.4849 (9.6%)$^{\blacktriangle}$	5
7	hitLRrun1	0.3355	0.4379	0.3226 (-3.9%)$^{\nabla}$	0.4525 (3.3%)	2	0.3188 (-5.0%)$^{\nabla}$	0.4610 (5.3%)$^{\blacktriangle}$	3
8	FASILKOM01	0.2682	0.4367	0.2820 (5.2%)$^{\blacktriangle}$	0.4531 (3.8%)$^{\blacktriangle}$	3	0.2622 (-2.2%)$^{\nabla}$	0.4346 (-0.5%)	-1
9	hitDELMrun2	0.3197	0.4345	0.3105 (-2.9%)	0.4424 (1.8%)	4	0.3246 (1.5%)	0.4723 (8.7%)$^{\blacktriangle}$	8
10	tsqe	0.2843	0.4339	0.2836 (-0.3%)	0.4441 (2.4%)	5	0.2917 (2.6%)	0.4660 (7.4%)$^{\blacktriangle}$	7
11	ICTWDSERUN1	0.2715	0.4299	0.2862 (5.4%)$^{\blacktriangle}$	0.4582 (6.6%)$^{\blacktriangle}$	7	0.2765 (1.8%)$^{\blacktriangle}$	0.4484 (4.3%)$^{\triangle}$	6
12	ICTWDSERUN2	0.2671	0.4266	0.2785 (4.3%)$^{\triangle}$	0.4475 (4.9%)$^{\blacktriangle}$	7	0.2786 (4.3%)$^{\triangle}$	0.4478 (5.0%)$^{\triangle}$	7
13	cmuPrfPhrE	0.3179	0.4254	0.3172 (-0.2%)	0.4469 (5.1%)$^{\blacktriangle}$	8	0.3321 (4.5%)$^{\triangle}$	0.4585 (7.8%)$^{\blacktriangle}$	9
14	cmuPrfPhrENo	0.3198	0.4249	0.3179 (-0.6%)	0.4486 (5.6%)$^{\blacktriangle}$	9	0.3359 (5.0%)$^{\triangle}$	0.4591 (8.1%)$^{\blacktriangle}$	10
15	cmuPrfPhr	0.3167	0.4198	0.3130 (-1.2%)	0.4379 (4.3%)$^{\triangle}$	8	0.3282 (3.6%)$^{\triangle}$	0.4572 (8.9%)$^{\blacktriangle}$	11
16	FASILKOM02	0.2454	0.4141	0.2718 (10.8%)$^{\blacktriangle}$	0.4508 (8.9%)$^{\blacktriangle}$	11	0.2489 (1.4%)	0.4201 (1.5%)$^{\triangle}$	1
17	IBMLTR	0.2630	0.4136	0.2734 (4.0%)$^{\triangle}$	0.4441 (7.4%)$^{\blacktriangle}$	10	0.2703 (2.8%)$^{\triangle}$	0.4346 (5.1%)$^{\blacktriangle}$	8
18	otM12ihe	0.2995	0.4124	0.2969 (-0.9%)	0.4322 (4.8%)$^{\blacktriangle}$	7	0.2900 (-3.2%)$^{\nabla}$	0.4239 (2.8%)$^{\triangle}$	3
19	FASILKOM03	0.2716	0.4124	0.2859 (5.3%)$^{\blacktriangle}$	0.4452 (8.0%)$^{\blacktriangle}$	14	0.2740 (0.9%)	0.4270 (3.5%)$^{\blacktriangle}$	7
20	FASILKOM04	0.2461	0.4113	0.2575 (4.6%)$^{\blacktriangle}$	0.4294 (4.4%)$^{\blacktriangle}$	9	0.2414 (-1.9%)$^{\nabla}$	0.4220 (2.6%)$^{\triangle}$	5
21	IBMLTRFuture	0.2731	0.4090	0.2808 (2.8%)	0.4311 (5.4%)$^{\blacktriangle}$	10	0.2785 (2.0%)$^{\triangle}$	0.4415 (8.0%)$^{\blacktriangle}$	14
22	uiucGSLIS01	0.2445	0.4073	0.2575 (5.3%)$^{\blacktriangle}$	0.4260 (4.6%)$^{\blacktriangle}$	9	0.2478 (1.4%)	0.4233 (3.9%)$^{\triangle}$	7
23	PKUICST4	0.2786	0.4062	0.2909 (4.4%)$^{\triangle}$	0.4514 (11.1%)$^{\blacktriangle}$	18	0.2832 (1.7%)$^{\blacktriangle}$	0.4491 (10.6%)$^{\blacktriangle}$	18
24	uogTrLsE	0.2909	0.4028	0.2977 (2.3%)	0.4282 (6.3%)$^{\blacktriangle}$	9	0.3131 (7.6%)$^{\blacktriangle}$	0.4484 (11.3%)$^{\blacktriangle}$	19
25	otM12ih	0.2777	0.3989	0.2810 (1.2%)	0.4175 (4.7%)$^{\blacktriangle}$	10	0.2752 (-0.9%)	0.4119 (3.3%)$^{\triangle}$	5
26	ICTWDSERUN4	0.1877	0.3887	0.1985 (5.8%)$^{\blacktriangle}$	0.4164 (7.1%)$^{\blacktriangle}$	10	0.2040 (8.7%)$^{\blacktriangle}$	0.4220 (8.6%)$^{\blacktriangle}$	11
27	uwatrrfall	0.2620	0.3881	0.2812 (7.3%)$^{\blacktriangle}$	0.4136 (6.6%)$^{\blacktriangle}$	9	0.2942 (12.3%)$^{\triangle}$	0.4314 (11.2%)$^{\blacktriangle}$	16
28	cmuPhrE	0.2731	0.3842	0.2797 (2.4%)	0.4136 (7.7%)$^{\blacktriangle}$	12	0.2972 (8.8%)$^{\triangle}$	0.4352 (13.3%)$^{\blacktriangle}$	19
29	AIrun1	0.2237	0.3842	0.2339 (4.6%)$^{\blacktriangle}$	0.4102 (6.8%)$^{\blacktriangle}$	5	0.2285 (2.2%)$^{\triangle}$	0.4157 (8.2%)$^{\blacktriangle}$	13
30	PKUICST3	0.2118	0.3825	0.2318 (9.4%)$^{\blacktriangle}$	0.4119 (7.7%)$^{\blacktriangle}$	14	0.2363 (11.6%)$^{\blacktriangle}$	0.4415 (15.4%)$^{\blacktriangle}$	23
	Average			3.3%	5.3%	7.3	2.0%	6.2%	7.8

model to learn tree-edit operations on dependency parse trees. They cast the problem into the framework of structured output learning with latent variables. The model of Yao et al., 2013 [37] applies linear chain CRFs with features derived from TED to automatically learn associations between questions and candidate answers. Severyn and Moschitti [26] applied SVM with tree kernels to shallow syntactic representation, which provide automatic feature engineering. Yih et al. [33] use distributional models based on lexical semantics to match semantic relations of aligned words in QA pairs.

More recently, Bordes et al. [5] used siamese networks for learning to project question and answer pairs into a joint space whereas Iyyer et al. [16] modelled semantic composition with a recursive neural network for a question answering task. The work closest to ours is [38], where they apply deep learning to learn to match question and answer sentences. However, their sentence model to map questions and answers to vectors operates only on unigrams or bigrams. Our sentence model is based on a convolutional neural network with the state-of-the-art architecture, we use a relatively large width of the convolution filter (5), thus allowing the network to capture longer range dependencies. Moreover, the architecture of deep learning model along with the question-answer similarity score also encodes question and answer vector representations in the model. Hence, our model constructs and learns a richer representation of the question-answer pairs, which results in superior results on the answer sentence selection dataset. Finally, our deep learning reranker is trained end-to-end, while in [38] they use the output of their neural network in a separate logistic scoring model.

Regarding learning to rank systems applied to TREC microblog datasets, recently [27] have shown that richer linguistic representations of tweets can improve upon state of the art systems in TMB-2011 and TMB-2012. We directly compare with their system, showing that our deep learning model without any changes to its architecture (we only pre-train word embeddings) is on the par with their reranker. This is remarkable, since different from [27], which requires additional pre-proccesing using syntactic parsers to construct syntactic trees, our model requires no expensive pre-processing and does not rely on any external resources.

8. CONCLUSIONS

In this paper, we propose a novel deep learning architecture for reranking short texts. It has the benefits of requiring no manual feature engineering or external resources, which may be expensive or not available. The model with the same architecture can be successfully applied to other domains and tasks.

Our experimental findings show that our deep learning model: (i) greatly improves on the previous state-of-the-art systems and a recent deep learning approach in [38] on answer sentence selection task showing a 3% absolute improvement in MAP and MRR; (ii) our system is able to improve even the best system runs from TREC Microblog 2012 challenge; (iii) is comparable to the syntactic reranker in [27], while our system requires no external parsers or resources.

Acknowledgments. This work has been supported by the EC project CogNet, 671625 (H2020-ICT-2014-2, Research and Innovation action). The first author was supported by the Google Europe Doctoral Fellowship Award 2013.

REFERENCES

[1] A. Agarwal, H. Raghavan, K. Subbian, P. Melville, D. Gondek, and R. Lawrence. Learning to rank for robust question answering. In *CIKM*, 2012.

[2] J. W. Antoine Bordes and N. Usunier. Open question answering with weakly supervised embedding models. In *ECML*, Nancy, France, September 2014.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[4] R. Berendsen, M. Tsagkias, W. Weerkamp, and M. de Rijke. Pseudo test collections for training and tuning microblog rankers. In *SIGIR*, 2013.

[5] A. Bordes, S. Chopra, and J. Weston. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 615–620, Doha, Qatar, October 2014. Association for Computational Linguistics.

[6] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 129–136, New York, NY, USA, 2007. ACM.

[7] Y. Chen, M. Zhou, and S. Wang. Reranking answers from definitional QA using language models. In *ACL*, 2006.

[8] R. Collobert and J. Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*, pages 160–167, 2008.

[9] H. Cui, M. Kan, and T. Chua. Generic soft pattern models for definitional QA. In *SIGIR*, Salvador, Brazil, 2005. ACM.

[10] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 1990.

[11] M. Denil, A. Demiraj, N. Kalchbrenner, P. Blunsom, and N. de Freitas. Modelling, visualising and summarising documents with a single convolutional neural network. Technical report, University of Oxford, 2014.

[12] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[13] A. Echihabi and D. Marcu. A noisy-channel approach to question answering. In *ACL*, 2003.

[14] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. In *ICML*, pages 1319–1327, 2013.

[15] M. Heilman and N. A. Smith. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *NAACL*, 2010.

[16] M. Iyyer, J. Boyd-Graber, L. Claudino, R. Socher, and H. Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 633–644, Doha, Qatar, October 2014. Association for Computational Linguistics.

[17] J. Jeon, W. B. Croft, and J. H. Lee. Finding similar questions in large question and answer archives. In *CIKM*, 2005.

[18] 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.

[19] Y. Kim. Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751, Doha, Qatar, October 2014.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119, 2013.

[21] A. Moschitti, S. Quarteroni, R. Basili, and S. Manandhar. Exploiting syntactic and shallow semantic kernels for question/answer classification. In *ACL*, 2007.

[22] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[23] I. Ounis, C. Macdonald, J. Lin, and I. Soboroff. Overview of the TREC-2011 microblog track. In *TREC*, 2011.

[24] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. *CoRR*, 2006.

[25] Y. Sasaki. Question answering as question-biased term extraction: A new approach toward multilingual qa. In *ACL*, 2005.

[26] A. Severyn and A. Moschitti. Automatic feature engineering for answer selection and extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 458–467, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[27] A. Severyn, A. Moschitti, M. Tsagkias, R. Berendsen, and M. de Rijke. A syntax-aware re-ranker for microblog retrieval. In *SIGIR*, 2014.

[28] D. Shen and M. Lapata. Using semantic roles to improve question answering. In *EMNLP-CoNLL*, 2007.

[29] I. Soboroff, I. Ounis, J. Lin, and I. Soboroff. Overview of the TREC-2012 microblog track. In *TREC*, 2012.

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[31] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to rank answers to non-factoid questions from web collections. *Comput. Linguist.*, 37(2):351–383, June 2011.

[32] J. Suzuki, Y. Sasaki, and E. Maeda. Svm answer selection for open-domain question answering. In *COLING*, 2002.

[33] W. tau Yih, M.-W. Chang, C. Meek, and A. Pastusiak. Question answering using enhanced lexical semantic models. In *ACL*, August 2013.

[34] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010.

[35] M. Wang and C. D. Manning. Probabilistic tree-edit models with structured latent variables for textual entailment and question answer- ing. In *ACL*, 2010.

[36] M. Wang, N. A. Smith, and T. Mitaura. What is the jeopardy model? a quasi-synchronous grammar for qa. In *EMNLP*, 2007.

[37] P. C. Xuchen Yao, Benjamin Van Durme and C. Callison-Burch. Answer extraction as sequence tagging with tree edit distance. In *NAACL*, 2013.

[38] L. Yu, K. M. Hermann, P. Blunsom, and S. Pulman. Deep learning for answer sentence selection. *CoRR*, 2014.

[39] M. D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, 2012.

[40] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013.