Attentive Pooling Networks

Cicero dos Santos   CICERONS@US.IBM.COM
Ming Tan   MINGTAN@US.IBM.COM
Bing Xiang   BINGXIA@US.IBM.COM
Bowen Zhou   ZHOU@US.IBM.COM

IBM Watson, T.J. Watson Research Center, NY, USA

Abstract

In this work, we propose *Attentive Pooling* (AP), a two-way attention mechanism for discriminative model training. In the context of pair-wise ranking or classification with neural networks, AP enables the pooling layer to be aware of the current input pair, in a way that information from the two input items can directly influence the computation of each other's representations. Along with such representations of the paired inputs, AP jointly learns a similarity measure over projected segments (e.g. trigrams) of the pair, and subsequently, derives the corresponding attention vector for each input to guide the pooling. Our two-way attention mechanism is a general framework independent of the underlying representation learning, and it has been applied to both convolutional neural networks (CNNs) and recurrent neural networks (RNNs) in our studies. The empirical results, from three very different benchmark tasks of question answering/answer selection, demonstrate that our proposed models outperform a variety of strong baselines and achieve state-of-the-art performance in all the benchmarks.

1. Introduction

Neural networks (NN) with attention mechanisms have recently proven to be successful at different computer vision (CV) and natural language processing (NLP) tasks such as image captioning (Xu et al., 2015), machine translation (Bahdanau et al., 2015) and factoid question answering (Hermann et al., 2015). However, most recent work on neural attention models have focused on one-way attention mechanisms based on recurrent neural networks designed for generation tasks.

Another important family of machine learning tasks are centered around pair-wise ranking or classification, which have a broad set of applications, including but not limited to, question answering, entailment, paraphrasing and any other pair-wise matching problems. The current state-of-the-art models usually include NN-based representation for the input pair, followed by a discriminative ranking or classification models. For example, a convolution (or a RNN) and a max-pooling is used to independently construct distributed vector representations of the input pair, followed by a large-margin training (Hu et al., 2014; Weston et al., 2014; Shen et al., 2014; dos Santos et al., 2015).

The key contribution of this work is that we propose *Attentive Pooling* (AP), a two-way attention mechanism, that significantly improves such discriminative models' performance on pair-wise ranking or classification, by enabling a joint learning of the representations of both inputs as well as their similarity measurement.

Specifically, AP enables the pooling layer to be aware of the current input pair, in a way that information from the two input items can directly influence the computation of each other's representations. The main idea in AP consists of learning a similarity measure over projected segments (e.g. trigrams) of the two items in the input pair, and using the similarity scores between the segments to compute attention vectors in both directions. Next, the attention vectors are used to perform pooling.

There are a few key benefits of our model.

- Thanks to the two-way attention, our model projects the paired inputs, even though they may not be always semantically comparable for some applications (e.g., questions and answers in question answering), into a common representation space that they can be compared in a more plausible way.

- Our model is effective in matching pairs of inputs with significant length variations.

- The two-way attention mechanism is independent of the underlying representation learning. For example, AP can be applied to both CNNs and RNNs, which is in contrast to the one-way attention used in the generation models mostly based on recurrent nets.

In this work, we perform an extensive number of experiments on applying attentive pooling CNNs (AP-CNN) and biLSTMs (AP-biLSTM) for the answer selection task. In this task, given a question q and an candidate answer pool $P = \{a_1, a_2, \cdots, a_p\}$ for this question, the goal is to search for and select the candidate answer $a \in P$ that correctly answers q. We perform experiments with three publicly available benchmark datasets, which vary in data scale, complexity and length ratios between question and answers: InsuranceQA, TREC-QA and WikiQA. For the three datasets, AP-CNN and AP-biLSTM respectively outperform the CNN and the biLSTM that do not use attention. Additionally, AP-CNN achieves state-of-the-art results for the three datasets.

Our experimental results also demonstrate that attentive pooling makes the CNN more robust to large input texts. This is an important finding, since recent work have demonstrated that, in the context of semantically equivalent question retrieval, CNN based representations do not scale well with the size of the input text (dos Santos et al., 2015). Additionally, as AP-CNN does not rely only on the final vector representation to capture interactions between the input question and answer, it requires much less convolutional filters than the regular CNN. It means that AP-CNN-based representations are more compact, which can help to speed up the training process.

Although we demonstrate experimental results for NLP tasks only, AP is a general method that can be also applied to different types of NNs that perform matching of two inputs. Therefore, we believe that AP can be useful for different applications, such as computer vision and bioinformatics.

This paper is organized as follows. In Section 2, we describe two NN architectures for answer selection that have been recently proposed in the literature. In Section 3, we detail the attentive pooling approach. In Section 4, we discuss some related work. Sections 5 and 6 detail our experimental setup and results, respectively. In Section 7 we present our final remarks.

2. Neural Networks for Answer Selection

Different neural network architectures have been recently proposed to perform matching of semantically related text segments (Yu et al., 2014; Hu et al., 2014; dos Santos et al., 2015; Wang & Nyberg, 2015; Severyn & Moschitti, 2015; Tan et al., 2015). In this section we briefly review two NN

architectures that have previously been applied to the answer selection task: QA-CNN (Feng et al., 2015) and QA-biLSTM (Tan et al., 2015). Given a pair (q, a) consisting of a question q and a candidate answer a, both networks score the pair by first computing fixed-length independent continuous vector representations r^q and r^a, and then computing the cosine similarity between these two vectors.

In Figure 1 we present a joint illustration of these two neural networks. The first layer in both QA-CNN and QA-biLSTM transforms each input word w into a fixed-size real-valued word embedding $r^w \in \mathbb{R}^d$. Word embeddings (WEs) are encoded by column vectors in an embedding matrix $W^0 \in \mathbb{R}^{d \times |V|}$, where V is a fixed-sized vocabulary and d is the dimention of the word embeddings. Given the input pair (q, a), where the question q contains M tokens and the candidate answer a contains L tokens, the output of the first layer consists of two sequences of word embeddings $q^{emb} = \{r^{w_1}, ..., r^{w_M}\}$ and $a^{emb} = \{r^{w_1}, ..., r^{w_L}\}$. Next, QA-CNN and QA-biLSTM use different approaches to process these sequences. While QA-CNN process both q^{emb} and a^{emb} using a convolution, QA-biLSTM uses a Bidirectional Long Short-Term Memory RNN (Hochreiter & Schmidhuber, 1997) to process these sequences.
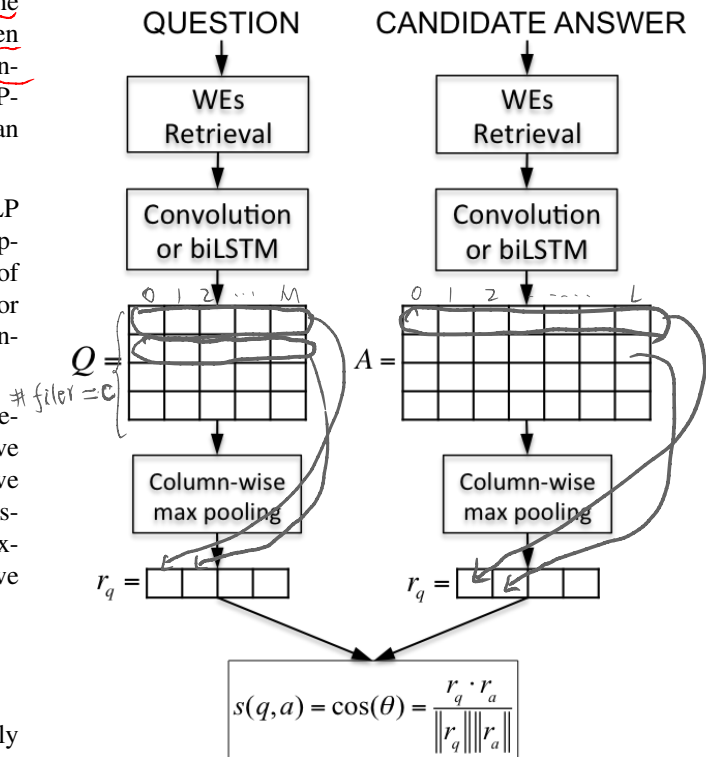

Figure 1. Joint illustration of QA-CNN and QA-biLSTM.

2.1. Convolution

Given the sequence $q^{emb} = \{r^{w_1}, ..., r^{w_M}\}$, let us define the matrix $Z^q = [z_1, ..., z_M]$ as a matrix where each column contains a vector $z_m \in \mathbb{R}^{dk}$ that is the concatenation of a sequence of k word embeddings centralized in the m-th word of the question. The output of the convolution with c filters over the question q is computed as follows:

$$Q = W^1 Z^q + b^1 \qquad (1)$$

where each column m in $Q \in \mathbb{R}^{c \times M}$ contains features extracted in a context window around the m-th word of q. The matrix W^1 and the vector b^1 are parameters to be learned. The number of convolutional filters c, and the size of the word context window k are hyper-parameters to be chosen by the user.

In a similar manner, and using the same NN parameters W^1 and b^1, we compute $A \in \mathbb{R}^{c \times L}$, the output of the convolution over the candidate answer a.

$$A = W^1 Z^a + b^1 \qquad (2)$$

2.2. Bidirectional LSTM (biLSTM)

Our LSTM implementation is similar to the one in (Graves et al., 2013) with minor modification. Given the sequence $q^{emb} = \{r^{w_1}, ..., r^{w_M}\}$, the hidden vector $\mathbf{h}(t)$ (with size H) at the time step t is updated as follows:

$$
\begin{aligned}
i_t &= \sigma(\mathbf{W}_i r^{w_t} + \mathbf{U}_i \mathbf{h}(t-1) + \mathbf{b}_i) & (3) \\
f_t &= \sigma(\mathbf{W}_f r^{w_t} + \mathbf{U}_f \mathbf{h}(t-1) + \mathbf{b}_f) & (4) \\
o_t &= \sigma(\mathbf{W}_o r^{w_t} + \mathbf{U}_o \mathbf{h}(t-1) + \mathbf{b}_o) & (5) \\
\tilde{C}_t &= \tanh(\mathbf{W}_m r^{w_t} + \mathbf{U}_m \mathbf{h}(t-1) + \mathbf{b}_m) & (6) \\
C_t &= i_t * \tilde{C}_t + f_t * C_{t-1} & (7) \\
\mathbf{h}_t &= o_t * \tanh(C_t) & (8)
\end{aligned}
$$

In the LSTM architecture, there are three gates (input i, forget f and output o), and a cell memory vector c. σ is the $sigmoid$ function. The input gate can determine how incoming vectors r^{w_t} alter the state of the memory cell. The output gate can allow the memory cell to have an effect on the outputs. Finally, the forget gate allows the cell to remember or forget its previous state. $\mathbf{W} \in R^{H \times d}$, $\mathbf{U} \in R^{H \times H}$ and $\mathbf{b} \in R^{H \times 1}$ are the network parameters.

Single direction LSTMs suffer a weakness of not utilizing the contextual information from the future tokens. Bidirectional LSTM utilizes both the previous and future context by processing the sequence on two directions, and generate two independent sequences of LSTM output vectors. One processes the input sequence in the forward direction, while the other processes the input in the reverse direction.

The output at each time step is the concatenation of the two output vectors from both directions, ie. $h_t = \overrightarrow{h_t} \parallel \overleftarrow{h_t}$. We define $c = 2 \times H$ for the notation consistency with the previous subsection. After computing the hidden state h_t for each time step t, we generate the matrices $Q \in \mathbb{R}^{c \times M}$ and $A \in \mathbb{R}^{c \times L}$, where the j-th column in Q (A) corresponds to j-th hidden state h_j that is computed by the biLSTM when processing q (a). The same network parameters are used to process both questions and candidate answers.

2.3. Scoring and Training Procedure

Given the matrices Q and A, we compute the vector representations $r^q \in \mathbb{R}^c$ and $r^a \in \mathbb{R}^c$ by applying a column-wise max-pooling over Q and A, followed by a non-linearity. Formally, the j-th elements of the vectors r^q and r^a are compute as follows:

$$[r^q]_j = \tanh\left(\max_{1 < m < M} [Q_{j,m}]\right) \qquad (9)$$

$$[r^a]_j = \tanh\left(\max_{1 < l < L} [A_{j,l}]\right) \qquad (10)$$

The last layer in QA-CNN and QA-biLSTM scores the input pair (q,a) by computing the cosine similarity between the two representations:

$$s(q, a) = \frac{r^q . r^a}{\|r^q\| \|r^a\|} \qquad (11)$$

Both networks are trained by minimizing a pairwise ranking loss function over the training set D. The input in each round is two pairs (q, a^+) and (q, a^-), where a^+ is a ground truth answer for q, and a^- is an incorrect answer. As in (Weston et al., 2014; Hu et al., 2014), we define the training objective as a hinge loss:

$$L = \max\{0, m - s_\theta(q, a^+) + s_\theta(q, a^-)\} \qquad (12)$$

where m is constant margin, $s_\theta(q, a^+)$ and $s_\theta(q, a^-)$ are scores generated by the network with parameter set θ. During training, for each question we randomly sample 50 negative answers from the entire answer set, but only use the one with the highest score to update the model.

We use stochastic gradient descent (SGD) to minimize the loss function with respect to θ. The backpropagation algorithm is used to compute the gradients of the network.

3. Attentive Pooling Networks for Answer Selection

Attentive pooling is an approach that enables the pooling layer to be aware of the current input pair, in a way that information from the question q can directly influence

学习输入对的 segment 的相似度，
然后根据相似度分值计算 attentive vector.

the computation of the answer representation r^a, and vice versa. The main idea consists of learning a similarity measure over the projected segments in the input pairs, and uses the similarity scores between the segments to compute attention vectors. When AP is applied to CNN, which we call AP-CNN, the network learns the similarity measure over the convolved input sequences. When AP is applied to biLSTM, which we call AP-biLSTM, the network learns the similarity measure over the hidden states produced by the biLSTM when processing the two input sequences. We use a similarity measure that has a bilinear form but followed by a non-linearity.

In Fig. 2, we illustrate the application of AP over the output of the convolution or the biLSTM to construct the representations r^q and r^a. Consider the input pair (q, a) where the question has size M and the answer has size L[1]. After we compute the matrices $Q \in \mathbb{R}^{c \times M}$ and $A \in \mathbb{R}^{c \times L}$, either by convolution or biLSTM, we compute the matrix $G \in \mathbb{R}^{M \times L}$ as follows:

双线性:
$$G = \tanh\left(Q^T U A\right) \tag{13}$$

where $U \in \mathbb{R}^{c \times c}$ is a matrix of parameters to be learned by the NN. When the convolution is used to compute Q and A, the matrix G contains the scores of a *soft alignment* between the convolved k-size context windows of q and a. When the biLSTM is used to compute Q and A, the matrix G contains the scores of a *soft alignment* between the hidden vectors of each token in q and a.

Next, we apply column-wise and row-wise max-poolings over G to generate the vectors $g^q \in \mathbb{R}^M$ and $g^a \in \mathbb{R}^L$, respectively. Formally, the j-th elements of the vectors g^q and g^a are computed as follows:

q中的第j个词对 answer的重要性

$$[g^q]_j = \max_{1 < m < M} [G_{j,m}] \tag{14}$$

$$[g^a]_j = \max_{1 < l < L} [G_{l,j}] \tag{15}$$

answer中的第j个词对 question的重要性

We can interpret each element j of the vector g^a as an *importance score* for the context around the j-th word in the candidate answer a with regard to the question q. Likewise, each element j of the vector g^q can be interpreted as the importance score for the context around the j-th word in the question q with regard to the candidate answer a.

Next, we apply the softmax function to the vectors g^q and g^a to create attention vectors σ^q and σ^a. For instance, the j-th element of the vector σ^q is computed as follows:

$$[\sigma^q]_j = \frac{e^{[g^q]_j}}{\sum\limits_{1 < l < M} e^{[g^q]_l}} \tag{16}$$

---

[1]In Fig. 2, q has a size of five and a has a size of seven.

Finally, the representations r^q and r^a are computed as the dot product between the attention vectors σ^q and σ^a and the output of the convolution (or biLSTM) over q and a, respectively:

$$r^q = Q\sigma^q \tag{17}$$

$$r^a = A\sigma^a \tag{18}$$

Like in QA-CNN and QA-biLSTM, the final score is also computed using the cosine similarity between r^q and r^a. We use SGD to train AP-CNN and AP-biLSTM by minimizing the same pairwise loss function used in QA-CNN and QA-biLSTM.

4. Related Work

Traditional work on answer selection have normally used feature engineering, linguistic tools, or external resources (Yih et al., 2013; Wang & Manning, 2010; Wang et al., 2007). Recently, deep learning (DL) approaches have been exploited for this task and achieved significant outperformance compared to traditional non-DL methods. For example, in (Yu et al., 2014; Feng et al., 2015; Severyn & Moschitti, 2015), the authors generate the representations of questions and answers separately, and score a QA pair using a similarity metric on top of these representations. In Wang & Nyberg (2015), first a joint feature vectors is learned from a joint long short-term memory (LSTM) model connecting questions and answers, and then the task is converted into a learning-to-rank problem.

At the same time, attention-based systems have shown very promising results on a variety of NLP tasks, such as machine translation (Bahdanau et al., 2015; Sutskever et al., 2014), caption generation (Xu et al., 2015) and factoid question answering (Hermann et al., 2015). Such models learn to focus their attention to specific parts of their input.

Some recently-proposed approaches introduce attention mechanisms in the answer selection task. Tan et al. (2015) developed an attentive reader based on bidirectional long short-term memory, which emphasizes certain part of the answer according to the question embedding. Unlike (Tan et al., 2015), in which attention is imposed only on answer embedding generation, AP-CNN and AP-biLSTM consider the interdependence between questions and answers.

In the context of two-way attention, two very recent work are related to ours. Rocktäschel et al. (2015), propose a two-way attention method that is inspired by bidirectional LSTMs that read a sequence and its reverse for improved encoding. Their approach, which is designed for RNNs only, differs in many aspects from the approach described in this work, which can be easily applied for CNNs and RNNs. Yin et al. (2015) present a two-way attention mechanism that is tailored to CNNs. Some of the main differ-

QUESTION

CANDIDATE ANSWER

WEs Retrieval

WEs Retrieval

Convolution or biLSTM

Convolution or biLSTM

$Q =$

$A =$

$\tanh\left(Q^T U A\right)$

$G =$ (MxL)

$g^q \in 1 \times M$

$g^a \in 1 \times L$

Column-wise max pooling

Row-wise max pooling

$r_q = Q \cdot \text{softmax}$ (dxM)

$r_a = A \cdot \text{softmax}$ (dxL)

(dx1)

(dx1)

(Mx1)

(Lx1)
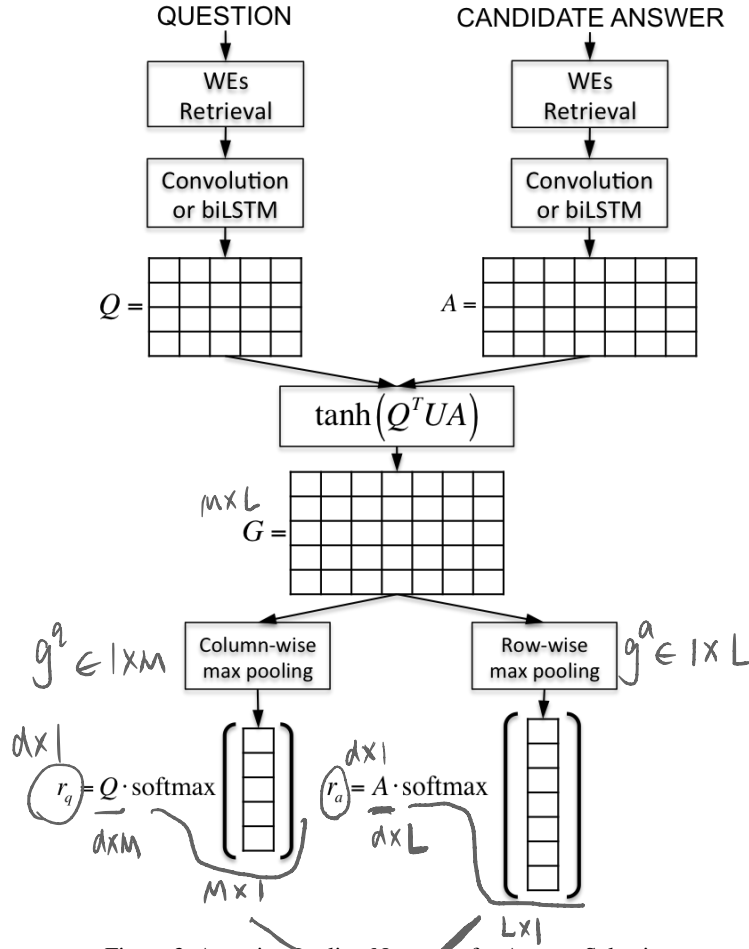
Figure 2. Attentive Pooling Networks for Answer Selection.

$s(q,a) = \cos\left(r_q, r_a\right)$ (1)

ences between their approach and this work are: (1) they use a simple Euclidean distance to compute the interdependence between the two input texts, while in this work we apply similarity metric learning, which has the potential to learn better ways to measure the interaction between segments of the input items; (2) the models in (Yin et al., 2015) compute the attention vector using sum-pooling over the alignment matrix and use the convolutional outputs updated by the attention as the input for another level of convolutional layer. In this work we use max-pooling over the alignment matrix plus softmax, in order to explicitly create an attention vector that is used to perform the pooling. Experimental results show that such difference yields substantial improvement of performance on WikiQA dataset.

5. Experimental Setup

5.1. Datasets

We apply AP-CNN, AP-biLSTM, QA-CNN and QA-biLSTM to three different answer selection datasets: Insur-

anceQA, TREC-QA and WikiQA. These datasets contain text of different domains and have different caracteristics. Table 1 presents some statistics about the datasets, including the number of questions in each set, average length of questions (M) and answers (L), average number of candidate answers in the dev/test sets and the average ratio between the lengths of questions and their ground-truth answers.

InsuranceQA[2] is a recently released large-scale non-factoid QA dataset from the insurance domain. This dataset provides a training set, a validation set, and two test sets. We do not see obvious categorical differentiation between questions of the two test sets. For each question in dev/test sets, there is a set of 500 candidate answers, which include the ground-truth answers and randomly selected negative answers. More details can be found in (Feng et al., 2015).

TREC-QA[3] was created by Wang et al. (2007) based on

---

[2]git clone https://github.com/shuzi/insuranceQA.git
[3]The data is obtained from (Yao et al., 2013)

Table 1. Answer Selection Datasets.

DATASET	TRAIN	DEV	TEST	AVG. M	AVG. L	AVG. # CAND. ANS.	AVG. L/M
INSURANCEQA	12887	1000	1800x2	7	95	500	13.8
TREC-QA	1162	65	68	8	28	38	4.2
WIKIQA	873	126	243	6	25	9	5.0

Text REtrieval Conference (TREC) QA track (8-13) data. We follow the exact approach of train/dev/test questions selection in (Wang & Nyberg, 2015), in which all questions with only positive or negative answers are removed. Finally, we have 1162 training questions, 65 development questions and 68 test questions.

WikiQA[4] is an open domain question-answering dataset. We use the subtask that assumes that there is at least one correct answer for a question. The corresponding dataset consists of 20,360 question/candidate pairs in train, 1,130 pairs in dev and 2,352 pairs in test. We adopt the standard setup of only considering questions that have correct answers for evaluation.

5.2. Word Embeddings

In order to fairly compare our results with the ones in previous work, we use two different sets of pre-trained word embeddings. For the InsuranceQA dataset, we use the 100-dimensional vectors that were trained by Feng et al. (2015) using word2vec (Mikolov et al., 2013). Following Wang & Nyberg (2015), Tan et al. (2015) and Yin et al.(2015), for the TREC-QA and the WikiQA datasets we use the 300-dimensional vectors that were trained using word2vec and are publicly available on the website of this tool[5].

5.3. Neural Networks Setup

In Table 2, we show the selected hyperparameter values, which were tuned using the validation sets. We try to use as much as possible the same hyperparameters for all the three datasets. The size of the word embeddings is different due to the different pre-trained versions that we used for InsuranceQA and the other two datasets. We use a context window of size 3 for InsuranceQA, while we set this parameter to 4 for TREC-QA and WikiQA. Using the selected hyperparameters, the best results are normally achieved using between 15 and 25 training epochs. For AP-CNN, AP-biLSTM and QA-LSTM, we also use a learning rate schedule that decreases the learning rate λ according to the training epoch t. Following dos Santos & Zadrozny (2014), we

[4]The data is obtained from (Yang et al., 2015)

[5]https://code.google.com/p/word2vec/

set the learning rate for epoch t, λ_t, using the equation: $\lambda_t = \dfrac{\lambda}{t}$.

In our experiments, the four NN architectures QA-CNN, AP-CNN, QA-biLSTM and AP-biLSTM are implemented using Theano (Bergstra et al., 2010).

6. Experimental Results

6.1. InsuranceQA

In Table 3, we present the experimental results of the four NNs for the InsuranceQA dataset. The results are in terms of accuracy, which is equivalent to precision at top one. On the bottom part of this table, we can see that AP-CNN outperforms QA-CNN by a large margin in both test sets, as well as in the dev set. AP-biLSTM also outperforms the QA-biLSTM in all the three sets. AP-CNN and AP-biLSTM have similar performance.

On the top part of Table 3 we present the results of two state-of-the-art systems for this dataset. In (Feng et al., 2015), the authors present a CNN architecture that is similar to QA-CNN, but that uses a different similarity metric instead of cosine similarity. In (Tan et al., 2015), the authors use a biLTSM architecture that employs unidirectional attention. Both AP-CNN and AP-biLSTM outperform the state-of-the-art systems.

Table 3. Accuracy of different systems for InsuranceQA

System	Dev	Test1	Test2
(FENG ET AL., 2015)	65.4	65.3	61.0
(TAN ET AL., 2015)	68.4	68.1	62.2
QA-CNN	61.6	60.2	56.1
QA-BILSTM	66.6	66.6	63.7
AP-CNN	**68.8**	69.8	66.3
AP-BILSTM	68.4	**71.7**	**66.4**

One important characteristic of AP-CNN is that it requires less convolutional filters than QA-CNN. For the InsuranceQA dataset, AP-CNN uses 10x less filters (400) than QA-CNN (4000). Using 800 filters in AP-CNN produces very similar results as using 400. On the other hand, as also found in (Feng et al., 2015), QA-CNN requires at least 2000 filters to achieve more than 60% accuracy on Insur-

Table 2. Neural Network Hyper-Parameters

Hyp.	Hyperpar. Name	AP-CNN	QA-CNN	AP-biLSTM	QA-biLSTM
d	WORD EMB. SIZE	100/300	100/300	100/300	100/300
c	CONV. FILTERS / HID. VEC. SIZE	400	4000	141x2	141x2
k	CONTEXT WINDOW SIZE	③/④ trec/wiki	2	1	1
mbs	MINIBATCH SIZE	20	1	20	20
m	LOSS MARGIN	0.5	0.009	0.2	0.1
λ	INIT. LEARNING RATE	1.1	0.05	1.1	1.1

anceQA. AP-CNN needs less filters because it does not rely only on the final vector representation to capture interactions between the input question and answer. As a result, although AP-CNN has a more complex architecture, its training time is two times faster than QA-CNN. Using a Tesla K20Xm, our Theano implementation of AP-CNN takes about 16 minutes to complete one epoch (training + inference over validation set) for InsuranceQA, which consists on processing 1.5 million text segments.

In figures 3 and 4, we plot the aggregated accuracy of AP-CNN and QA-CNN for answers up to a certain length for the Test1 and Test2 sets, respectively. We can see in both plots that the performance of both system is better for shorter answers. However, while the performance of QA-CNN continues to drop as larger answers are considered, the performance of AP-CNN seems to be stable after reaching a length of ~90 tokens. These results give support to our hypothesis that attentive pooling helps the CNN to become robust to larger input texts.
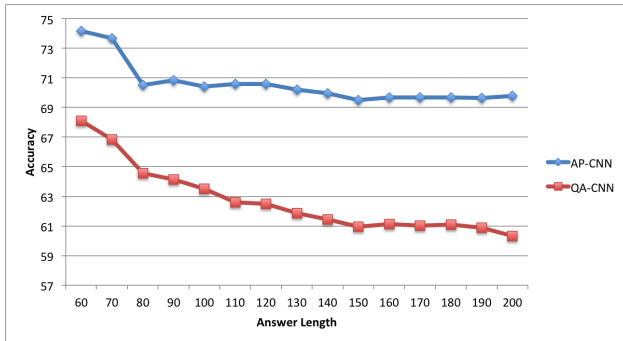
Figure 3. Aggregated accuracy for answers up to a certain length in the InsuranceQA Test1 set

6.2. TREC-QA

In Table 4, we present the experimental results of the four NNs for the TREC-QA dataset. The results are in terms of mean average precision (MAP) and mean reciprocal rank (MRR), which are the metric normally used in previous work with the same dataset. We use the official *trec_eval*
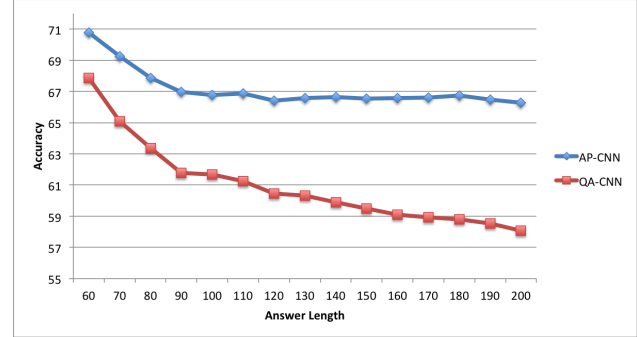

Figure 4. Aggregated accuracy for answers up to a certain length in the InsuranceQA Test2 set

scorer to compute MAP and MRR. We can see in Table 4 that AP-CNN outperforms QA-CNN by a large margin in both metrics. AP-biLSTM outperforms the QA-biLSTM, but its performance is not as good as the of AP-CNN.

On the top part of Table 4 we present the results of three recent work that use TREC-QA as a benchmark. In (Wang & Nyberg, 2015), the authors present an LTSM architecture for answer selection. Their best result consists of a combination of LSTM and the BM25 algorithm. In (Severyn & Moschitti, 2015), the authors propose an NN architecture where the representations created by a convolutional layer are the input to similarity measure learning. Wang & Ittycheriah (2015) propose a word-alignment-based method that is suitable for the FAQ-based QA task. AP-CNN outperforms the state-of-the-art systems in both metrics, MAP and MRR.

6.3. WikiQA

Table 5 shows the experimental results of the four NNs for the WikiQA dataset. Like in the other two datasets, AP-CNN outperforms QA-CNN, and AP-biLSTM outperforms the QA-biLSTM. The difference of performance between AP-CNN and QA-CNN is smaller than the one for the InsuranceQA dataset. We believe it is because the average size of the answers in WikiQA (25) is much smaller

Table 4. Performance of different systems for TREC-QA

System	MAP	MRR
WANG & NYBERG (2015)	0.7134	0.7913
SEVERYN & MOSCHITTI (2015)	0.7460	0.8080
WANG & ITTYCHERIAH (2015)	0.7460	0.8200
QA-BILSTM	0.6750	0.7723
QA-CNN	0.7147	0.8070
AP-BILSTM	0.7132	0.8032
AP-CNN	**0.7530**	**0.8511**

than in InsuranceQA (95). It is expected that attentive pooling bring more impact to the datasets that have larger answer/question lengths.

In Table 5 we also present the results of two recent work that use WikiQA as a benchmark. Yang et al. (2015), present a bigram CNN model with average pooling. In (Yin et al., 2015), the authors propose an attention-based CNN. In order to make a fair comparison, in Table 5 we include Yin et al.'s result that use word embeddings only[6]. AP-CNN outperforms these two systems in both metrics.

Table 5. Performance of different systems for WikiQA

System	MAP	MRR
YANG ET AL. (2015)	0.6520	0.6652
YIN ET AL. (2015)	0.6600	0.6770
QA-BILSTM	0.6557	0.6695
QA-CNN	0.6701	0.6822
AP-BILSTM	0.6705	0.6842
AP-CNN	**0.6886**	**0.6957**

6.4. Attentive Pooling Visualization

Figures 5 and 6 depict two heat maps of two test questions from InsuranceQA that were correctly answered by AP-CNN and whose answers are more than 100 words long. The stronger the color of a word in the question (answer), the larger the attention weight in σ^q (σ^a) of the trigram centered at that word. As we can see in the pictures, the attentive pooling mechanism is indeed putting more focus on the segments of the answer that have some interaction with the question, and vice-verse.

7. Conclusions

We present attentive pooling, a two-way attention mechanism for discriminative model training. The main contributions of the paper are: (1) AP is more general than

---

[6]Yin et al. (Yin et al., 2015) report 0.6921(MAP) and 0.7108(MRR) when using handcrafted features in addition to word embeddings.

Figure 5. Attention heat map from AP-CNN for a correctly selected answer.

recently proposed two-way attention mechanism because: (a) it learns how to compute interactions between the items in the input pair; and (b) it can be applied to both CNNs and RNNs; (2) we demonstrate that AP can be effectively used with CNNs and biLSTM in the context of the answer selection task, using three different benchmark datasets; (3) our experimental results demonstrate that AP helps the CNN to cope with large input texts; (4) we present new state-of-the-art results for InsuranceQA and TREC-QA datasets. (5) for the WikiQA dataset our results are the best reported so far for methods that do not use handcrafted features.

Acknowledgements

References

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference*, 2010.

dos Santos, Cicero, Barbosa, Luciano, Bogdanova, Dasha, and Zadrozny, Bianca. Learning hybrid representations

Figure 6. Attention heat map from AP-CNN for a correctly selected answer.

to retrieve semantically equivalent questions. In *ACL*, 2015.

dos Santos, Cicero Nogueira and Zadrozny, Bianca. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning, JMLR: W&CP volume 32*, Beijing, China, 2014.

Feng, Minwei, Xiang, Bing, Glass, Michael R, Wang, Lidan, and Zhou, Bowen. Applying deep learning to answer selection: A study and an open task. *arXiv preprint:1508.01585*, 2015.

Graves, Alex, Mohamed, Abdel-rahman, and Hinton, Geoffrey. Speech recognition with deep recurrent neural networks. *In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

Hermann, Karl Moritz, Kocisky, Tomas, Grefenstette, Edward, Espeholt, Lasse, Kay, Will, Suleyman, Mustafa, and Blunsom, Phil. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28*, pp. 1684–1692, 2015.

Hochreiter, Sepp and Schmidhuber, Jurgen. Long short-term memory. *Neural Computation*, 1997.

Hu, Baotian, Lu, Zhengdong, Li, Hang, and Chen, Qingcai. Convolutional neural network architectures for matching natural language sentences. *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S., and Dean, Jeff. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems (NIPS)*, 2013.

Rocktäschel, Tim, Grefenstette, Edward, Hermann, Karl Moritz, Kociský, Tomás, and Blunsom, Phil. Reasoning about entailment with neural attention. *CoRR*, abs/1509.06664, 2015.

Severyn, Aliaksei and Moschitti, Alessandro. Learning to rank short text pairs with convolutional deep neural networks. *Proceedings of SIGIR*, 2015.

Shen, Yelong, He, Xiaodong, Gao, Jianfeng, Deng, Li, and Mesnil, Grégoire. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pp. 101–110, 2014. ISBN 978-1-4503-2598-1.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 2014.

Tan, Ming, dos Santos, Cicero, Xiang, Bing, and Zhou, Bowen. Lstm-based deep learning models for non-factoid answer selection. *CoRR*, abs/1511.04108, 2015.

Wang, Di and Nyberg, Eric. A long short-term memory model for answer sentence selection in question answering. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, 2015.

Wang, Mengqiu and Manning, Christopher. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. *The Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, 2010.

Wang, Mengqiu, Smith, Noah, and Teruko, Mitamura. What is the jeopardy model? a quasi-synchronous grammar for qa. *The Proceedings of EMNLP-CoNLL*, 2007.

Wang, Zhiguo and Ittycheriah, Abraham. Faq-based question answering via word alignment. *arXiv preprint arXiv:1507.02628*, 2015.

Weston, Jason, Chopra, Sumit, and Adams, Keith. #tagspace: Semantic embeddings from hashtags. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Cho, Kyunghyun, Courville, Aaron C., Salakhutdinov, Ruslan, Zemel, Richard S., and Bengio, Yoshua. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 2048–2057, 2015.

Yang, Yi, Yih, Wen-tau, and Meek, Christopher. Wikiqa: A challenge dataset for open-domain question answering. *In Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

Yao, Xuchen, Durme, Benjamin, and Clark, Peter. Answer extraction as sequence tagging with tree edit distance. *Proceedings of NAACL-HLT*, 2013.

Yih, Wen-tau, Chang, Ming-Wei, Meek, Christopher, and Pastusiak, Andrzej. Question answering using enhanced lexical semantic models. *Proceedings of the 51st Annual Meeting of the Association for Computational Linguist (ACL)*, 2013.

Yin, Wenpeng, Schütze, Hinrich, Xiang, Bing, and Zhou, Bowen. ABCNN: attention-based convolutional neural network for modeling sentence pairs. *CoRR*, abs/1512.05193, 2015.

Yu, Lei, Hermann, Karl M., Blunsom, Phil, and Pulman, Stephen. Deep learning for answer sentence selection. *NIPS Deep Learning Workshop*, 2014.