ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs

Wenpeng Yin, Hinrich Schütze
Center for Information and Language Processing
LMU Munich, Germany
wenpeng@cis.lmu.de

Bing Xiang, Bowen Zhou
IBM Watson
Yorktown Heights, NY, USA
bingxia,zhou@us.ibm.com

Abstract

How to model a pair of sentences is a critical issue in many NLP tasks such as answer selection (AS), paraphrase identification (PI) and textual entailment (TE). Most prior work (i) deals with one individual task by fine-tuning a specific system; (ii) models each sentence’s representation separately, rarely considering the impact of the other sentence; or (iii) relies fully on manually designed, task-specific linguistic features. This work presents a general Attention Based Convolutional Neural Network (ABCNN) for modeling a pair of sentences. We make three contributions. (i) The ABCNN can be applied to a wide variety of tasks that require modeling of sentence pairs. (ii) We propose three attention schemes that integrate mutual influence between sentences into CNNs; thus, the representation of each sentence takes into consideration its counterpart. These interdependent sentence pair representations are more powerful than isolated sentence representations. (iii) ABCNNs achieve state-of-the-art performance on AS, PI and TE tasks. We release code at: https://github.com/yinwenpeng/Answer_Selection.

1 Introduction

How to model a pair of sentences is a critical issue in many NLP tasks such as answer selection (AS) (Yu et al., 2014; Feng et al., 2015), paraphrase identification (PI) (Madnani et al., 2012; Yin and Schütze, 2015a), textual entailment (TE) (Marelli et al., 2014a; Bowman et al., 2015a) etc.

Figure 1 demonstrates that each sentence of a pair partially determines which parts of the other sentence we must focus on. For AS, correctly answering \( s_0 \) requires attention on “gross”: \( s_1^+ \) contains a corresponding unit (“earned”) while \( s_1^- \) does not. For PI, focus should be removed from “today” to correctly recognize \( < s_0, s_1^+ > \) as paraphrases and \( < s_0, s_1^- > \) as non-paraphrases. For TE, we need to focus on “full of people” (to recognize TE for \( < s_0, s_1^+ > \)) and on “outdoors” / “indoors” (to recognize non-TE for \( < s_0, s_1^- > \)). These examples show the need for an architecture that computes different representations of \( s_i \) for different \( s_{1-i} (i \in \{0, 1\}) \).
Convolutional Neural Networks (CNNs) (LeCun et al., 1998) are widely used to model sentences (Kalchbrenner et al., 2014; Kim, 2014) and sentence pairs (Socher et al., 2011; Yin and Schütze, 2015a), especially in classification tasks. CNNs are supposed to be good at extracting robust and abstract features of input. This work presents the ABCNN, an attention-based convolutional neural network, that has a powerful mechanism for modeling a sentence pair by taking into account the interdependence between the two sentences. The ABCNN is a general architecture that can handle a wide variety of sentence pair modeling tasks.

Some prior work proposes simple mechanisms that can be interpreted as controlling varying attention; e.g., Yih et al. (2013) employ word alignment to match related parts of the two sentences. In contrast, our attention scheme based on CNNs models relatedness between two parts fully automatically. Moreover, attention at multiple levels of granularity, not only at word level, is achieved as we stack multiple convolution layers that increase abstraction.

Prior work on attention in deep learning (DL) mostly addresses long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997). LSTMs achieve attention usually in a word-to-word scheme, and word representations mostly encode the whole context within the sentence (Bahdanau et al., 2015; Rocktäschel et al., 2016). It is not clear whether this is the best strategy; e.g., in the AS example in Figure 1, it is possible to determine that “how much” in $s_0$ matches “$161.5 million” in $s_1$ without taking the entire sentence contexts into account. This observation was also investigated by Yao et al. (2013b) where an information retrieval system retrieves sentences with tokens labeled as DATE by named entity recognition or as CD by POS tagging if there is a “when” question. However, labels or POS tags require extra tools. CNNs benefit from incorporating attention into representations of local phrases detected by filters; in contrast, LSTMs encode the whole context to form attention-based word representations—a strategy that is more complex than the CNN strategy and (as our experiments suggest) performs less well for some tasks.

Apart from these differences, it is clear that attention has as much potential for CNNs as it does for LSTMs. As far as we know, this is the first NLP paper that incorporates attention into CNNs. Our ABCNNs get state-of-the-art in AS and TE tasks, and competitive performance in PI, then obtains further improvements over all three tasks when linguistic features are used.

2 Related Work

Non-DL on Sentence Pair Modeling. Sentence pair modeling has attracted lots of attention in the past decades. Many tasks can be reduced to a semantic text matching problem. In this paper, we adopt the arguments by Yih et al. (2013) who argue against shallow approaches as well as against semantic text matching approaches that can be computationally expensive:

Due to the variety of word choices and inherent ambiguities in natural language, bag-of-word approaches with simple surface-form word matching tend to produce brittle results with poor prediction accuracy (Bilotti et al., 2007). As a result, researchers put more emphasis on exploiting syntactic and semantic structure. Representative examples include methods based on deeper semantic analysis (Shen and Lapata, 2007; Moldovan et al., 2007), tree edit-distance (Punyakanok et al., 2004; Heilman and Smith, 2010) and quasi-synchronous grammars (Wang et al., 2007) that match the dependency parse trees of the two sentences. Instead of focusing on the high-level semantic representation, Yih et al. (2013) turn their attention to improving the shallow semantic component, lexical semantics, by performing semantic matching based on a latent word-alignment structure (cf. Chang et al. (2010)). Lai and Hockenmaier (2014) explore fine-grained word overlap and alignment between two sentences using negation, hypernym, synonym and antonym relations. Yao et al. (2013a) extend word-to-word alignment to phrase-to-phrase alignment by a semi-Markov CRF. However, such approaches often require more computational resources. In addition, employing syntactic or semantic parsers—which produce errors on many sentences— to find the best match between the structured representations of two sentences is not trivial.
**DL on Sentence Pair Modeling.** To address some of the challenges of non-DL work, much recent work uses neural networks to model sentence pairs for AS, PI and TE.

For AS, Yu et al. (2014) present a bigram CNN to model question and answer candidates. Yang et al. (2015) extend this method and get state-of-the-art performance on the WikiQA dataset (Section 5.1). Feng et al. (2015) test various setups of a bi-CNN architecture on an insurance domain QA dataset. Tan et al. (2016) explore bidirectional LSTMs on the same dataset. Our approach is different because we do not model the sentences by two independent neural networks in parallel, but instead as an interdependent sentence pair, using attention.

For PI, Blacoe and Lapata (2012) form sentence representations by summing up word embeddings. Socher et al. (2011) use recursive autoencoders (RAEs) to model representations of local phrases in sentences, then pool similarity values of phrases from the two sentences as features for binary classification. Yin and Schütze (2015a) similarly replace an RAE with a CNN. In all three papers, the representation of one sentence is not influenced by the other – in contrast to our attention-based model.

For TE, Bowman et al. (2015b) use recursive neural networks to encode entailment on SICK (Marelli et al., 2014b). Rocktäschel et al. (2016) present an attention-based LSTM for the Stanford natural language inference corpus (Bowman et al., 2015a). Our system is the first CNN-based work on TE.

Some prior work aims to solve a general sentence matching problem. Hu et al. (2014) present two CNN architectures, ARC-I and ARC-II, for sentence matching. ARC-I focuses on sentence representation learning while ARC-II focuses on matching features on phrase level. Both systems were tested on PI, sentence completion (SC) and tweet-response matching. Yin and Schütze (2015b) propose the MultiGranCNN architecture to model general sentence matching based on phrase matching on multiple levels of granularity and get promising results for PI and SC. Wan et al. (2016) try to match two sentences in AS and SC by multiple sentence representations, each coming from the local representations of two LSTMs. Our work is the first one to investigate attention for the general sentence matching task.

**Attention-Based DL in Non-NLP Domains.** Even though there is little if any work on attention mechanisms in CNNs for NLP, attention-based CNNs have been used in computer vision for visual question answering (Chen et al., 2015), image classification (Xiao et al., 2015), caption generation (Xu et al., 2015), image segmentation (Hong et al., 2016) and object localization (Cao et al., 2015).

Mnih et al. (2014) apply attention in recurrent neural networks (RNNs) to extract “information from an image or video by adaptively selecting a sequence of regions or locations and only processing the selected regions at high resolution.” Gregor et al. (2015) combine a spatial attention mechanism with RNNs for image generation. Ba et al. (2015) investigate attention-based RNNs for recognizing multiple objects in images. Chorowski et al. (2014) and Chorowski et al. (2015) use attention in RNNs for speech recognition.

**Attention-Based DL in NLP.** Attention-based DL systems have been applied to NLP after their success in computer vision and speech recognition. They mainly rely on RNNs and end-to-end encoder-decoders for tasks such as machine translation (Bahdanau et al., 2015; Luong et al., 2015) and text reconstruction (Li et al., 2015; Rush et al., 2015). Our work takes the lead in exploring attention mechanisms in CNNs for NLP tasks.

### 3 BCNN: Basic Bi-CNN

We now introduce our basic (non-attention) CNN that is based on the Siamese architecture (Bromley et al., 1993), i.e., it consists of two weight-sharing CNNs, each processing one of the two sentences, and a final layer that solves the sentence pair task. See Figure 2. We refer to this architecture as the BCNN. The next section will then introduce the ABCNN, an attention architecture that extends the BCNN. Table 1 gives our notational conventions.

In our implementation and also in the mathematical formalization of the model given below, we pad the two sentences to have the same length $s = \max(s_0, s_1)$. However, in the figures we show different lengths because this gives a better intuition of how the model works.

We now describe the BCNN’s four types of layers: input, convolution, average pooling and output.
Table 1: Notation

<table>
<thead>
<tr>
<th>symbol</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$, $s_0$, $s_1$</td>
<td>sentence or sentence length</td>
</tr>
<tr>
<td>$v$</td>
<td>word</td>
</tr>
<tr>
<td>$w$</td>
<td>filter width</td>
</tr>
<tr>
<td>$d_i$</td>
<td>dimensionality of input to layer $i+1$</td>
</tr>
<tr>
<td>$W$</td>
<td>weight matrix</td>
</tr>
</tbody>
</table>

Figure 2: BCNN: ABCNN without Attention

**Input layer.** In the example in the figure, the two input sentences have 5 and 7 words, respectively. Each word is represented as a $d_0$-dimensional precomputed word2vec (Mikolov et al., 2013) embedding, $d_0 = 300$. As a result, each sentence is represented as a feature map of dimension $d_0 \times s$.

**Convolution layer.** Let $v_1, v_2, \ldots, v_s$ be the words of a sentence and $c_i \in \mathbb{R}^{w \cdot d_0}$, $0 < i < s + w$, the concatenated embeddings of $v_{i-w+1}, \ldots, v_i$ where embeddings for $v_j$ are set to zero when $j < 1$ or $j > s$. We then generate the representation $p_i \in \mathbb{R}^{d_1}$ for the phrase $v_{i-w+1}, \ldots, v_i$ using the convolution weights $W \in \mathbb{R}^{d_1 \times w \cdot d_0}$ as follows:

$$p_i = \text{tanh} (W \cdot c_i + b)$$

where $b \in \mathbb{R}^{d_1}$ is the bias.

**Average pooling layer.** Pooling (including min, max, average pooling) is commonly used to extract robust features from convolution. In this paper, we introduce attention weighting as an alternative, but use average pooling as a baseline as follows.

For the output feature map of the last convolution layer, we do column-wise averaging over all columns, denoted as all-ap. This generates a representation vector for each of the two sentences, shown as the top “Average pooling (all-ap)" layer below “Logistic regression" in Figure 2. These two vectors are the basis for the sentence pair decision.

For the output feature map of non-final convolution layers, we do column-wise averaging over windows of $w$ consecutive columns, denoted as $w$-ap; shown as the lower “Average pooling (w-ap)" layer in Figure 2. For filter width $w$, a convolution layer transforms an input feature map of $s$ columns into a new feature map of $s + w - 1$ columns; average pooling transforms this back to $s$ columns. This architecture supports stacking an arbitrary number of convolution-pooling blocks to extract increasingly abstract features. Input features to the bottom layer are words, input features to the next layer are short phrases and so on. Each level generates more abstract features of higher granularity.

The last layer is an output layer, chosen according to the task; e.g., for binary classification tasks, this layer is logistic regression (see Figure 2). Other types of output layers are introduced below.

We found that in most cases, performance is boosted if we provide the output of all pooling layers as input to the output layer. For each non-final average pooling layer, we perform $w$-ap (pooling over windows of $w$ columns) as described above, but we also perform all-ap (pooling over all columns) and forward the result to the output layer. This improves performance because representations from different layers cover the properties of the sentences at different levels of abstraction and all of these levels can be important for a particular sentence pair.

4 ABCNN: Attention-Based BCNN

We now describe three architectures based on the BCNN, the ABCNN-1, the ABCNN-2 and the ABCNN-3, that each introduces an attention mechanism for modeling sentence pairs; see Figure 3.

**ABCNN-1.** The ABCNN-1 (Figure 3(a)) employs an attention feature matrix $A$ to influence co-
Figure 3: Three ABCNN architectures

(a) One block in ABCNN-1

(b) One block in ABCNN-2

(c) One block in ABCNN-3
volution. Attention features are intended to weight those units of $s_1$ more highly in convolution that are relevant to a unit of $s_{1-i}$ ($i \in \{0, 1\}$); we use the term “unit” here to refer to words on the lowest level and to phrases on higher levels of the network. Figure 3(a) shows two unit representation feature maps in red: this part of the ABCNN-1 is the same as in the BCNN (see Figure 2). Each column is the representation of a unit, a word on the lowest level and a phrase on higher levels. We first describe the attention feature matrix $A$ informally (layer “Conv input”), middle column, in Figure 3(a)). $A$ is generated by matching units of the left representation feature map with units of the right representation feature map such that the attention values of row $i$ in $A$ denote the attention distribution of the $i$-th unit of $s_0$ with respect to $s_1$, and the attention values of column $j$ in $A$ denote the attention distribution of the $j$-th unit of $s_1$ with respect to $s_0$. $A$ can be viewed as a new feature map of $s_0$ (resp. $s_1$) in row (resp. column) direction because each row (resp. column) is a new feature vector of a unit in $s_0$ (resp. $s_1$). Thus, it makes sense to combine this new feature map with the representation feature maps and use both as input to the convolution operation. We achieve this by transforming $A$ into the two blue matrices in Figure 3(a) that have the same format as the representation feature maps. As a result, the new input of convolution has two feature maps for each sentence (shown in red and blue). Our motivation is that the attention feature map will guide the convolution to learn “counterpart-biased” sentence representations.

More formally, let $F_{i,r} \in \mathbb{R}^{d \times s}$ be the representation feature map of sentence $i$ ($i \in \{0, 1\}$). Then we define the attention matrix $A \in \mathbb{R}^{s \times s}$ as follows:

$$A_{i,j} = \text{match-score}(F_{0,r}[;i], F_{1,r}[;j])$$  \hspace{1cm} (1)

The function match-score can be defined in a variety of ways. We found that $1/(1+|x-y|)$ works well where $|\cdot|$ is Euclidean distance.

Given attention matrix $A$, we generate the attention feature map $F_{i,a}$ for $s_i$ as follows:

$$F_{0,a} = W_0 \cdot A^\top, \quad F_{1,a} = W_1 \cdot A$$

The weight matrices $W_0 \in \mathbb{R}^{d \times s}$, $W_1 \in \mathbb{R}^{d \times s}$ are parameters of the model to be learned in training.\(^{1}\)

We stack the representation feature map $F_{i,r}$ and the attention feature map $F_{i,a}$ as an order 3 tensor and feed it into convolution to generate a higher-level representation feature map for $s_i$ ($i \in \{0, 1\}$). In Figure 3(a), $s_0$ has 5 units, $s_1$ has 7. The output of convolution (shown in the top layer, filter width $w = 3$) is a higher-level representation feature map with 7 columns for $s_0$ and 9 columns for $s_1$.

**ABCNN-2.** The ABCNN-1 computes attention weights directly on the input representation with the aim of improving the features computed by convolution. The ABCNN-2 (Figure 3(b)) instead computes attention weights on the output of convolution with the aim of reweighting this convolution output. In the example shown in Figure 3(b), the feature maps output by convolution for $s_0$ and $s_1$ (layer marked “Convolution” in Figure 3(b)) have 7 and 9 columns, respectively; each column is the representation of a unit. The attention matrix $A$ compares all units in $s_0$ with all units of $s_1$. We sum all attention values for a unit to derive a single attention weight for that unit. This corresponds to summing all values in a row of $A$ for $s_0$ ("col-wise sum", resulting in the column vector of size 7 shown) and summing all values in a column for $s_1$ ("row-wise sum", resulting in the row vector of size 9 shown).

More formally, let $A \in \mathbb{R}^{s \times s}$ be the attention matrix, $a_{0,j} = \sum A[j, :]$ the attention weight of unit $j$ in $s_0$, $a_{1,j} = \sum A[:, j]$ the attention weight of unit $j$ in $s_1$ and $F^c_{i,r} \in \mathbb{R}^{d \times (s_i+w-1)}$ the output of convolution for $s_i$. Then the $j$-th column of the new feature map $F^p_{i,r}$ generated by $w$-ap is derived by:

$$F^p_{i,r}[;j] = \sum_{k=0}^{w-1} a_{i,k} F^c_{i,r}[;k], \quad j = 1 \ldots s_i$$

Note that $F^p_{i,r} \in \mathbb{R}^{d \times s_i}$, i.e., ABCNN-2 pooling generates an output feature map of the same size as the input feature map of convolution. This allows us to stack multiple convolution-pooling blocks to extract features of increasing abstraction.

There are three main differences between the ABCNN-1 and the ABCNN-2. (i) Attention in the ABCNN-1 impacts convolution indirectly while attention in the ABCNN-2 influences pooling through direct attention weighting. (ii) The ABCNN-1 requires the two matrices $W_i$ to convert the attention matrix into attention feature maps; and the input to
convolution has two times as many feature maps. Thus, the ABCNN-1 has more parameters than the ABCNN-2 and is more vulnerable to overfitting. (iii) As pooling is performed after convolution, pooling handles larger-granularity units than convolution; e.g., if the input to convolution has word level granularity, then the input to pooling has phrase level granularity, the phrase size being equal to filter size \( w \). Thus, the ABCNN-1 and the ABCNN-2 implement attention mechanisms for linguistic units of different granularity. The complementarity of the ABCNN-1 and the ABCNN-2 motivates us to propose the ABCNN-3, a third architecture that combines elements of the two.

**ABCNN-3** (Figure 3(c)) combines the ABCNN-1 and the ABCNN-2 by stacking them; it combines the strengths of the ABCNN-1 and -2 by allowing the attention mechanism to operate (i) both on the convolution and on the pooling parts of a convolution-pooling block and (ii) both on the input granularity and on the more abstract output granularity.

### 5 Experiments

We test the proposed architectures on three tasks: answer selection (AS), paraphrase identification (PI) and textual entailment (TE).

**Common Training Setup.** Words are initialized by 300-dimensional word2vec embeddings and not changed during training. A single randomly initialized embedding is created for all unknown words by uniform sampling from \([-0.01, 0.01]\). We employ Adagrad (Duchi et al., 2011) and \( L_2 \) regularization.

**Network Configuration.** Each network in the experiments below consists of (i) an initialization block \( b_1 \) that initializes words by word2vec embeddings, (ii) a stack of \( k-1 \) convolution-pooling blocks \( b_2, \ldots, b_k \), computing increasingly abstract features, and (iii) one final **LR layer** (logistic regression layer) as shown in Figure 2.

The input to the LR layer consists of \( kn \) features – each block provides \( n \) similarity scores, e.g., \( n \) cosine similarity scores. Figure 2 shows the two sentence vectors output by the final block \( b_k \) of the stack (“sentence representation 0”, “sentence representation 1”); this is the basis of the last \( n \) similarity scores. As we explained in the final paragraph of Section 3, we perform *all-ap* pooling for all blocks.

<table>
<thead>
<tr>
<th>#CL</th>
<th>AS</th>
<th>PI</th>
<th>TE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( w)</td>
<td>( w)</td>
<td>( w)</td>
</tr>
<tr>
<td>ABCNN-1 1</td>
<td>.08 4</td>
<td>.08 3</td>
<td>.08 3</td>
</tr>
<tr>
<td>ABCNN-1 2</td>
<td>.085 4</td>
<td>.085 3</td>
<td>.085 3</td>
</tr>
<tr>
<td>ABCNN-2 1</td>
<td>.05 4</td>
<td>.05 3</td>
<td>.05 3</td>
</tr>
<tr>
<td>ABCNN-2 2</td>
<td>.06 4</td>
<td>.06 5</td>
<td>.06 5</td>
</tr>
<tr>
<td>ABCNN-3 1</td>
<td>.05 4</td>
<td>.05 3</td>
<td>.05 3</td>
</tr>
<tr>
<td>ABCNN-3 2</td>
<td>.06 4</td>
<td>.06 5</td>
<td>.06 5</td>
</tr>
</tbody>
</table>

Table 2: Hyperparameters. \( \#CL \): number convolution layers. \( w \): filter width. The number of convolution kernels \( d_i (i > 0) \) is 50 throughout. Not just for \( b_k \). Thus we get one sentence representation each for \( s_0 \) and \( s_1 \) for each block \( b_1, \ldots, b_k \). We compute \( n \) similarity scores for each block (based on the block’s two sentence representations). Thus, we compute a total of \( kn \) similarity scores and these scores are input to the LR layer.

Depending on the task, we use different methods for computing the similarity score: see below.

**Layerwise Training.** In our training regime, we first train a network consisting of just one convolution-pooling block \( b_2 \). We then create a new network by adding a block \( b_3 \), initialize its \( b_2 \) block with the previously learned weights for \( b_2 \) and train \( b_3 \) keeping the previously learned weights for \( b_2 \) fixed. We repeat this procedure until all \( k-1 \) convolution-pooling blocks are trained. We found that this training regime gives us good performance and shortens training times considerably. Since similarity scores of lower blocks are kept unchanged once they have been learned, this also has the nice effect that “simple” similarity scores (those based on surface features) are learned first and subsequent training phases can focus on complementary scores derived from more complex abstract features.

**Classifier.** We found that performance increases if we do not use the output of the LR layer as the final decision, but instead train a linear SVM or a logistic regression with default parameters\(^2\) directly on the input to the LR layer (i.e., on the \( kn \) similarity scores that are generated by the \( k \)-block stack after network training is completed). Direct training of SVMs/LR seems to get closer to the global optimum than gradient descent training of CNNs.

Table 2 shows hyperparameters, tuned on dev.

We use addition and LSTMs as two shared baselines for all three tasks, i.e., for AS, PI and TE. We

now describe these two shared baselines.

(i) **Addition.** We sum up word embeddings element-wise to form each sentence representation. The classifier input is then the concatenation of the two sentence representations. (ii) **A-LSTM.** Before this work, most attention mechanisms in NLP were implemented in recurrent neural networks for text generation tasks such as machine translation (e.g., Bahdanau et al. (2015), Luong et al. (2015)). Rocktäschel et al. (2016) present an attention-LSTM for natural language inference. Since this model is the pioneering attention based RNN system for sentence pair classification, we consider it as a baseline system (“A-LSTM”) for all our three tasks. The A-LSTM has the same configuration as our ABCNNs in terms of word initialization (300-dimensional word2vec embeddings) and the dimensionality of all hidden layers (50).

### 5.1 Answer Selection

We use WikiQA, an open domain question-answer 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 where we adopt the standard setup of only considering questions with correct answers in test. Following Yang et al. (2015), we truncate answers to 40 tokens.

The task is to rank the candidate answers based on their relatedness to the question. Evaluation measures are mean average precision (MAP) and mean reciprocal rank (MRR).

**Task-Specific Setup.** We use cosine similarity as the similarity score for AS. In addition, we use sentence lengths, `WordCnt` (count of the number of non-stopwords in the question that also occur in the answer) and `WgtWordCnt` (reweight the counts by the IDF values of the question words). Thus, the final input to the LR layer has size $k + 4$: one cosine for each of the $k$ blocks and the four additional features.

We compare with seven baselines. The first three are considered by Yang et al. (2015): (i) `WordCnt`; (ii) `WgtWordCnt`; (iii) `CNN-Cnt` (the state-of-the-art system): combine CNN with (i) and (ii). Apart from the baselines considered by Yang et al. (2015), we compare with two Addition baselines and two LSTM baselines. Addition and A-LSTM are the shared baselines described before. We also combine both with the four extra features; this gives us two additional baselines that we refer to as Addition(+) and A-LSTM(+).

**Results.** Table 3 shows performance of the baselines, of the BCNN and of the three ABCNNs. For CNNs, we test one (one-conv) and two (two-conv) convolution-pooling blocks. The non-attention network BCNN already performs better than the baselines. If we add attention mechanisms, then the performance further improves by several points. Comparing the ABCNN-2 with the ABCNN-1, we find the ABCNN-2 is slightly better even though the ABCNN-2 is the simpler architecture. If we combine the ABCNN-1 and the ABCNN-2 to form the ABCNN-3, we get further improvement.

This can be explained by the ABCNN-3’s ability to take attention of finer-grained granularity into consideration in each convolution-pooling block while the ABCNN-1 and the ABCNN-2 consider attention only at convolution input or only at pooling input, respectively. We also find that stacking two convolution-pooling blocks does not bring consistent improvement and therefore do not test deeper architectures.

<table>
<thead>
<tr>
<th>Baselines</th>
<th>Two-conv</th>
<th>One-conv</th>
<th>MAP</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>0.5888</td>
<td>0.5929</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addition(+)</td>
<td>0.6381</td>
<td>0.6537</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-LSTM</td>
<td>0.5347</td>
<td>0.5483</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-LSTM(+)</td>
<td>0.6381</td>
<td>0.6537</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCNN</td>
<td>0.6629</td>
<td>0.6813</td>
<td></td>
<td></td>
</tr>
<tr>
<td>two-conv</td>
<td>0.6593</td>
<td>0.6738</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABCNN-1</td>
<td>0.6810*</td>
<td>0.6979*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>one-conv</td>
<td>0.6855*</td>
<td>0.7023*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>two-conv</td>
<td>0.6885*</td>
<td>0.7054*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABCNN-2</td>
<td>0.6879*</td>
<td>0.7068*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>one-conv</td>
<td>0.6914*</td>
<td>0.7127*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>two-conv</td>
<td>0.6921*</td>
<td>0.7108*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Results on WikiQA. Best result per column is bold. Significant improvements over state-of-the-art baselines (underlined) are marked with * ($t$-test, $p < .05$).
5.2 Paraphrase Identification

We use the Microsoft Research Paraphrase (MSRP) corpus (Dolan et al., 2004). The training set contains 2753 true / 1323 false and the test set 1147 true / 578 false paraphrase pairs. We randomly select 400 pairs from train and use them as dev; but we still report results for training on the entire training set. For each triple (label, \(s_0, s_1\)) in the training set, we also add (label, \(s_1, s_0\)) to the training set to make best use of the training data. Systems are evaluated by accuracy and \(F_1\).

**Task-Specific Setup.** In this task, we add the 15 MT features from (Madnani et al., 2012) and the lengths of the two sentences. In addition, we compute ROUGE-1, ROUGE-2 and ROUGE-SU4 (Lin, 2004), which are scores measuring the match between the two sentences on (i) unigrams, (ii) bigrams and (iii) unigrams and skip-bigrams (maximum skip distance of four), respectively. In this task, we found transforming Euclidean distance into similarity score by \(1/(1 + |x - y|)\) performs better than cosine similarity. Additionally, we use dynamic pooling (Yin and Schütze, 2015a) of the attention matrix \(A\) in Equation (1) and forward pooled values of all blocks to the classifier. This gives us better performance than only forwarding sentence-level matching features.

We compare our system with representative DL approaches: (i) A-LSTM; (ii) A-LSTM(+): A-LSTM plus handcrafted features; (iii) RAE (Socher et al., 2011), recursive autoencoder; (iv) Bi-CNN-MI (Yin and Schütze, 2015a), a bi-CNN architecture; and (v) MPSSM-CNN (He et al., 2015), the state-of-the-art NN system for PI, and the following four non-DL systems: (vi) Addition; (vii) Addition(+): Addition plus handcrafted features; (viii) MT (Madnani et al., 2012), a system that combines machine translation metrics;\(^5\) (ix) MF-TF-KLD (Ji and Eisenstein, 2013), the state-of-the-art non-NN system.

**Results.** Table 4 shows that the BCNN is slightly worse than the state-of-the-art whereas the ABCNN-1 roughly matches it. The ABCNN-2 is slightly above the state-of-the-art. The ABCNN-3 outperforms the state-of-the-art in accuracy and \(F_1.\)\(^6\) Two convolution layers only bring small improvements over one.

5.3 Textual Entailment

SemEval 2014 Task 1 (Narelli et al., 2014a) evaluates system predictions of textual entailment (TE) relations on sentence pairs from the SICK dataset (Marelli et al., 2014b). The three classes are entailment, contradiction and neutral. The sizes of SICK train, dev and test sets are 4439, 495 and 4906 pairs, respectively. We call this dataset ORIG.

We also create NONOVER, a copy of ORIG in which words occurring in both sentences are removed. A sentence in NONOVER is denoted by the special token \(<\empty>\) if all words are removed. Table 5 shows three pairs from ORIG and their transformation in NONOVER. We observe that focusing on the non-overlapping parts provides clearer hints for TE than ORIG. In this task, we run two copies of each network, one for ORIG, one for NONOVER; these two networks have a single common LR layer.

Like Lai and Hockenmaier (2014), we train our final system (after fixing hyperparameters) on train and dev (4934 pairs). Eval measure is accuracy.

**Task-Specific Setup.** We found that for this task forwarding two similarity scores from each block performs the state-of-the-art in accuracy and \(F_1.\)\(^6\) Two convolution layers only bring small improvements over one.

<table>
<thead>
<tr>
<th>method</th>
<th>acc</th>
<th>(F_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>majority voting</td>
<td>66.5</td>
<td>79.9</td>
</tr>
<tr>
<td>RAE</td>
<td>76.8</td>
<td>83.6</td>
</tr>
<tr>
<td>Bi-CNN-MI</td>
<td>78.4</td>
<td>84.6</td>
</tr>
<tr>
<td>MPSSM-CNN</td>
<td>78.6</td>
<td>84.7</td>
</tr>
<tr>
<td>MT</td>
<td>76.8</td>
<td>83.8</td>
</tr>
<tr>
<td>MF-TF-KLD</td>
<td>78.6</td>
<td>84.6</td>
</tr>
<tr>
<td>Addition</td>
<td>70.8</td>
<td>80.9</td>
</tr>
<tr>
<td>Addition(+)</td>
<td>77.3</td>
<td>84.1</td>
</tr>
<tr>
<td>A-LSTM</td>
<td>69.5</td>
<td>80.1</td>
</tr>
<tr>
<td>A-LSTM(+)</td>
<td>77.1</td>
<td>84.0</td>
</tr>
</tbody>
</table>

Table 4: Results for PI on MSRP

---

\(^5\)For better comparability of approaches in our experiments, we use a simple SVM classifier, which performs slightly worse than Madnani et al. (2012)’s more complex meta-classifier.

\(^6\)Improvement of .3 (acc) and .1 (\(F_1\)) over state-of-the-art is not significant. The ABCNN-3 (two-conv) without “linguistic” features (i.e., MT and ROUGE) achieves 75.1/82.7.
neural network developed for this SICK task.

and TrRNTN (Bowman et al., 2015b), a recursive further compare with the top-3 systems in SemEval

tons.

number of word pairs in s: length s

contradictory (PAP) in NONOVER. We identify the matched pairs. We exclude synonyms and hy-

onyms and antonyms in the pairs. But we do this on NONOVER (not on ORIG) to focus on what

is critical for TE. Specifically, feature SYN is the number of word pairs in s0 and s1 that are syn-

onyms. HYP0 (resp. HYP1) is the number of words in s0 (resp. s1) that have a hypernym in s1 (resp.

s0). In addition, we collect all potential antonym pairs (PAP) in NONOVER. We identify the matched

chucks that occur in contradictory and neutral, but not in entailed pairs. We exclude synonyms and hy-

pernyms and apply a frequency filter of n = 2. In contrast to Lai and Hockenmaier (2014), we con-

strain the PAP pairs to cosine similarity above 0.4 in word2vec embedding space as this discards many

oise pairs. Feature ANT is the number of matched PAP antonyms in a sentence pair. As before we use

sentence lengths, both for ORIG (LEN0: length s0, LEN1O: length s1) and for NONOVER (LEN0N:

length s0, LEN1N: length s1).

On the whole, we have 24 extra features: 15 MT metrics, NEG, SYN, HYP0, HYP1, ANT, LEN0, LEN1O, LEN0N and LEN1N.

Apart from the Addition and LSTM baselines, we further compare with the top-3 systems in SemEval and TrRNTN (Bowman et al., 2015b), a recursive neural network developed for this SICK task.

Table 5: SICK data: Converting the original sentences (ORIG) into the NONOVER format

<table>
<thead>
<tr>
<th>ORIG</th>
<th>NONOVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>children in red shirts are children red shirts</td>
</tr>
<tr>
<td></td>
<td>playing in the leaves playing</td>
</tr>
<tr>
<td></td>
<td>three kids are sitting in the leaves three kids sitting</td>
</tr>
<tr>
<td>1</td>
<td>three boys are jumping in the leaves boys</td>
</tr>
<tr>
<td></td>
<td>three kids are jumping in the leaves kids</td>
</tr>
<tr>
<td>2</td>
<td>a man is jumping into an empty pool an empty</td>
</tr>
<tr>
<td></td>
<td>a man is jumping into a full pool a full</td>
</tr>
</tbody>
</table>

Table 6: Results on SICK. Significant improvements over (Lai and Hockenmaier, 2014) are marked with * (test of equal proportions, p < .05).

<table>
<thead>
<tr>
<th></th>
<th>method</th>
<th>acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sem-Eval Top3</td>
<td>(Jimenez et al., 2014)</td>
<td>83.1</td>
</tr>
<tr>
<td></td>
<td>(Zhao et al., 2014)</td>
<td>83.6</td>
</tr>
<tr>
<td>TrRNTN</td>
<td>(Bowman et al., 2015b)</td>
<td>84.6</td>
</tr>
<tr>
<td></td>
<td>(Lai and Hockenmaier, 2014)</td>
<td>84.6</td>
</tr>
<tr>
<td>Addition</td>
<td>no features</td>
<td>73.1</td>
</tr>
<tr>
<td></td>
<td>plus features</td>
<td>79.4</td>
</tr>
<tr>
<td></td>
<td>no features</td>
<td>78.0</td>
</tr>
<tr>
<td>A-LSTM</td>
<td>plus features</td>
<td>81.7</td>
</tr>
<tr>
<td></td>
<td>two-conv</td>
<td>85.0</td>
</tr>
<tr>
<td></td>
<td>one-conv</td>
<td>84.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCNN</td>
<td>one-conv</td>
<td>85.6</td>
</tr>
<tr>
<td></td>
<td>two-conv</td>
<td>85.8</td>
</tr>
<tr>
<td>ABCNN-1</td>
<td>one-conv</td>
<td>85.7</td>
</tr>
<tr>
<td></td>
<td>two-conv</td>
<td>85.8</td>
</tr>
<tr>
<td>ABCNN-2</td>
<td>one-conv</td>
<td>85.7</td>
</tr>
<tr>
<td></td>
<td>two-conv</td>
<td>85.8</td>
</tr>
<tr>
<td>ABCNN-3</td>
<td>one-conv</td>
<td>86.0*</td>
</tr>
<tr>
<td></td>
<td>two-conv</td>
<td>86.2*</td>
</tr>
</tbody>
</table>

We use the following linguistic features. Negation is important for detecting contradiction. Feature NEG is set to 1 if either sentence contains “no”, “not”, “nobody”, “isn’t” and to 0 otherwise. Following Lai and Hockenmaier (2014), we use WordNet (Miller, 1995) to detect nyms: synonyms, hypernyms and antonyms in the pairs. We use the following linguistic features. Negation is important for detecting contradiction. Feature NEG is set to 1 if either sentence contains “no”, “not”, “nobody”, “isn’t” and to 0 otherwise. Following Lai and Hockenmaier (2014), we use WordNet (Miller, 1995) to detect nyms: synonyms, hypernyms and antonyms in the pairs. We use the following linguistic features. Negation is important for detecting contradiction. Feature NEG is set to 1 if either sentence contains “no”, “not”, “nobody”, “isn’t” and to 0 otherwise. Following Lai and Hockenmaier (2014), we use WordNet (Miller, 1995) to detect nyms: synonyms, hypernyms and antonyms in the pairs. We can see that words in s with semantic equivalents in s1 get high attention while words without semantic equivalents get low attention, e.g., “walking” and “mural” in s0 and “front” and “colorful” in s1. This behavior seems reasonable for the unigram level.

Rows/columns of the attention matrix in Figure 4 (middle) correspond to phrases of length three since filter width w = 3. High attention values generally correlate with close semantic correspondence: the phrase “people are” in s0 matches “several people are” in s1; both “are walking outside” and “walking outside the” in s0 match “are in front” in s1; “the building that” in s0 matches “a colorful building” in

Visual Analysis. Figure 4 visualizes the attention matrices for one TE sentence pair in the ABCNN-2 for blocks b1 (unigrams), b2 (first convolutional layer) and b3 (second convolutional layer). Darker shades of blue indicate stronger attention values.

In Figure 4 (top), each word corresponds to exactly one row or column. We can see that words in s3 with semantic equivalents in s1 get high attention while words without semantic equivalents get low attention, e.g., “walking” and “mural” in s0 and “front” and “colorful” in s1. This behavior seems reasonable for the unigram level.

Visual Analysis. Figure 4 visualizes the attention matrices for one TE sentence pair in the ABCNN-2 for blocks b1 (unigrams), b2 (first convolutional layer) and b3 (second convolutional layer). Darker shades of blue indicate stronger attention values.

In Figure 4 (top), each word corresponds to exactly one row or column. We can see that words in s3 with semantic equivalents in s1 get high attention while words without semantic equivalents get low attention, e.g., “walking” and “mural” in s0 and “front” and “colorful” in s1. This behavior seems reasonable for the unigram level.

If we run the ABCNN-3 (two-conv) without the 24 linguistic features, performance is 84.6.
several people are in front of a colorful building

Figure 4: Attention visualization for TE. Top: unigrams, $b_1$. Middle: conv1, $b_2$. Bottom: conv2, $b_3$.

$s_1$. More interestingly, looking at the bottom right corner, both “on it” and “it” in $s_0$ match “building” in $s_1$; this indicates that ABCNNs are able to detect some coreference across sentences. “building” in $s_1$ has two places in which higher attentions appear, one is with “it” in $s_0$, the other is with “the building that” in $s_0$. This may indicate that ABCNNs recognize that “building” in $s_1$ and “the building that” / “it” in $s_0$ refer to the same object. Hence, coreference resolution across sentences as well as within a sentence both are detected. For the attention vectors on the left and the top, we can see that attention has focused on the key parts: “people are walking outside the building that” in $s_0$, “several people are in” and “of a colorful building” in $s_1$.

Rows/columns of the attention matrix in Figure 4 (bottom, second layer of convolution) correspond to phrases of length 5 since filter width $w = 3$ in both convolution layers ($5 = 1 + 2 \times (3 - 1)$). We use “…” to denote words in the middle if a phrase like “several…front” has more than two words. We can see that attention distribution in the matrix has focused on some local regions. As granularity of phrases is larger, it makes sense that the attention values are smoother. But we still can find some interesting clues: at the two ends of the main diagonal, higher attentions hint that the first part of $s_0$ matches well with the first part of $s_1$; “several murals on it” in $s_0$ matches well with “of a colorful building” in $s_1$, which satisfies the intuition that these two phrases are crucial for making a decision on TE in this case. This again shows the potential strength of our system in figuring out which parts of the two sentences refer to the same object. In addition, in the central part of the matrix, we can see that the long phrase “people are walking outside the building” in $s_0$ matches well with the long phrase “are in front of a colorful building” in $s_1$.

6 Summary

We presented three mechanisms to integrate attention into CNNs for general sentence pair modeling tasks.

Our experiments on AS, PI and TE show that attention-based CNNs perform better than CNNs without attention mechanisms. The ABCNN-2 generally outperforms the ABCNN-1 and the ABCNN-3 surpasses both.

In all tasks, we did not find any big improvement of two layers of convolution over one layer. This is probably due to the limited size of training data. We expect that, as larger training sets become available, deep ABCNNs will show even better performance.

In addition, linguistic features contribute in all three tasks: improvements by 0.0321 (MAP) and 0.0338 (MRR) for AS, improvements by 3.8 (acc) and 2.1 ($F_1$) for PI and an improvement by 1.6 (acc) for TE. But our ABCNNs can still reach or surpass state-of-the-art even without those features in AS and TE tasks. This indicates that ABCNNs are generally strong NN systems.
Attention-based LSTMs are especially successful in tasks with a strong generation component like machine translation (discussed in Sec. 2). CNNs have not been used for this type of task. This is an interesting area of future work for attention-based CNNs.

Acknowledgments

We gratefully acknowledge the support of Deutsche Forschungsgemeinschaft (DFG): grant SCHU 2246/8-2.

We would like to thank the anonymous reviewers for their helpful comments.

References


