Sentence Similarity Measurement with Convolutional Neural Networks Using Semantic and Syntactic Features

Shiru Zhang[1], Zhiyao Liang[1, *] and Jian Lin[2]

Abstract: Calculating the semantic similarity of two sentences is an extremely challenging problem. We propose a solution based on convolutional neural networks (CNN) using semantic and syntactic features of sentences. The similarity score between two sentences is computed as follows. First, given a sentence, two matrices are constructed accordingly, which are called the syntax model input matrix and the semantic model input matrix; one records some syntax features, and the other records some semantic features. By experimenting with different arrangements of representing the syntactic and semantic features of the sentences in the matrices, we adopt the most effective way of constructing the matrices. Second, these two matrices are given to two neural networks, which are called the sentence model and the semantic model, respectively. The convolution process of the neural networks of the two models is carried out in multiple perspectives. The outputs of the two models are combined as a vector, which is the representation of the sentence. Third, given the representation vectors of two sentences, the similarity score of these representations is computed by a layer in the CNN. Experiment results show that our algorithm (SSCNN) surpasses the performance MPCPP, which noticeably the best recent work of using CNN for sentence similarity computation. Comparing with MPCNN, the convolution computation in SSCNN is considerably simpler. Based on the results of this work, we suggest that by further utilization of semantic and syntactic features, the performance of sentence similarity measurements has considerable potentials to be improved in the future.

Keywords: Sentence similarity, neural network, convolutional neural networks.

1 Introduction

Computation of semantic similarity between sentences has broad application areas. For example, we want to build an Intelligent Question Answering (IQA) system. When a user asks a question, if we can find a similar question in the knowledge base, then the existing answer can be provided. In this article, we present a solution for computing semantic similarity of sentences which is based on convolution neural networks (CNN) [Kim (2014)] and modeling sentences using their syntactic and semantic features.

[1] Faculty of Information Technology, Macau University of Science and Technology, Macau.

[2] Management Information Systems, University of Houston-Clear Lake, Houston, 77058, USA.

[*] Corresponding Author: Zhiyao Liang. Email: zyliang@must.edu.mo.

Received: 10 October 2019; Accepted: 05 November 2019.

1.1 Related work

Because language expressions are multitudinous and the corresponding sentence semantics are complex, calculating the semantic similarity of two sentences is extremely challenging. Before researchers started using machine learning on natural language processing (NLP), the traditional techniques of NLP have significant achievements such as the understanding of rules of sentence syntax. However, most of the recent exciting performance improvements of solutions to NLP problems are based on machine learning techniques.

For the area of sentence similarity computation, the most noticeable recent researches are using deep neural networks. Frameworks of deep neural networks include Convolutional Neural Network (CNN) [Goodfellow, Bengio and Courville (2016)] and Recurrent Neural Network (RNN) [Siegelmann and Son-tag (1995)]. Long Short-Term Memory (LSTM) [Hochreiter and Schmid-huber(1997)] has become a more applied form of RNN in NLP research areas.

Zeng et al. [Zeng, Liu, Lai et al. (2014)] presented a well-known paper using CNN to classify relationships between words and to extract sentence features.

Tai et al. [Tai, Socher and Manning (2015)] proposed the Tree-LSTM model, which achieved very good results of computing sentence similarity.

Mueller et al. [Mueller and Thyagarajan (2015)] have presented a Siamese adaptation of LSTM called MaLSTM, which achieved state-of-art results using the SemEval 2014 dataset [Marelli, Bentivogli, Baroni et al. (2014)], and outperformed the Tree-LSTM model.

He et al. [He, Gimpel and Lin (2015)] have proposed Multi-Perspective CNN (MPCNN) for modelling of sentence similarity. With MPCNN, sentence features can be extracted from more prospects that can improve the accuracy of sentence similarity computation. MPCNN also achieved state-of-art results and outperforms the Tree-LSTM model.

The design idea of modeling sentences with more features in the work of this paper is like MPCNN. Besides other new features of the proposed solution, we have unique modeling considerations of sentence syntactic and semantic features.

Choosing the underlying neural network architecture is a dimension of consideration for designing an algorithm. It is difficult to say which choice is better, CNN or RNN (LSTM), for solving an NLP problem. In this work, we choose the research direction of using CNN. The experience that we gained in this research, including the integration of syntactic of semantic features of sentences, could have positive influences on designing better algorithms using other neural network architecture, like LSTM, with similar insights or techniques for performance improvements.

2 Overview of the sentence similarity computation process

A sentence model is designed to compute sentence representations. The proposed model consists of two parts. One is the syntax model that computes using the syntactic features of a sentence, the other is the semantic model that computes using the semantic features of the words in a sentence. The computation of the model has multiple ways of convolution and pooling like those used by the MPCNN algorithm. This model doesn't require a dictionary environment like WordNet; thus, the dependence on detailed definitions of every word is avoided. It needs word embeddings, which can be obtained from published resources. Comparing to the previously published works, the main differences of our solution lie in our unique ways of constructing the input matrices of the neural network, and the ways of conducting the convolutions.

As shown in Fig. 1, a sentence is processed in parallel by the computation of the semantic and syntactic models. Through the full connection layer, based on the outputs of the two models, a synthesized sentence representation is computed, which is a vector. Given two sentence representation vectors, their similarity is computed by the similarity computation layer the CNN. The output of the similarity computation layer is a similarity score, which is a value between 0 and 1 and can be scaled up to some range like [0, 5].

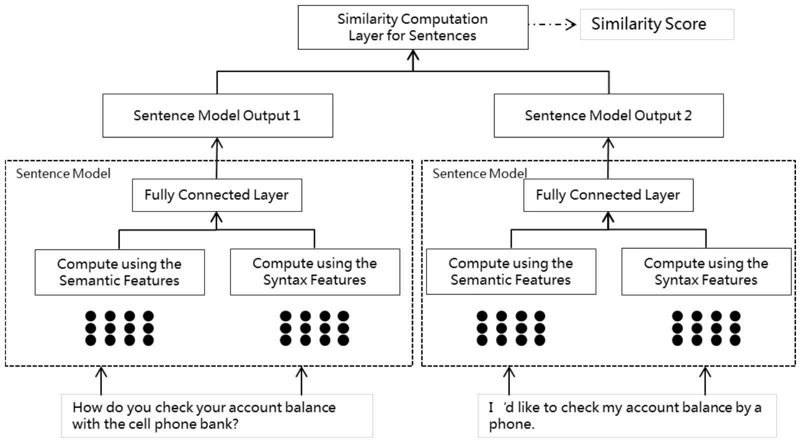
Similarity Computation Layer for Sentences
Similarity Score
Sentence Model Output 1
Sentence Model Output 2
Sentence Model
Fully Connected Layer
Compute using the Semantic Features
Compute using the Syntax Features
Sentence Model
Fully Connected Layer
Compute using the Semantic Features
Compute using the Syntax Features
How do you check your account balance with the cell phone bank?
I 'd like to check my account balance by a phone.

Figure 1: The process of computing similarity scores

3 Text segmentation and vectorization

Given a sentence, we want to partition its sequence of characters into a sequence of tokens (words), which is the text segmentation process. Once the token list of a sentence is obtained, we want to obtain a vector (embeddings) of each word that can represent some semantic features of the word in multiple dimensions. This is the word vectorization process and can be done using the popular software package Word2Vec [Bojanowski, Grave, Joulin et al. (2017)]. The distance between two vectors of two words should correspond to the semantic similarity between the two words.

Although the presented experiment results are purely for English sentences, we have also experimented with Chinese sentences, since we are also interested in solving NLP problems with Chinese sentences. For Chinese sentences, we used the software package JIEBA to obtain the tokens. For English sentences, we can use the NLTK [Wagner (2010)] packages to do text segmentation. In the final stage of our experiments, whose results are presented in this paper, we used the Stanford CoreNLP package [Marneffe and Manning (2008)] to do text segmentation for English sentences, and we used the pre-trained Word2Vec file provided by the Stanford GloVe [Pennington, Socher and Manning (2014)] website.

4 Sentence model

4.1 Semantic model

4.1.1 Input matrix to the semantic model

Given a sentence, after the word segmentation and vectorization process, suppose there are n words, we can construct a $n \times n$ matrix M as follows. The item at the row j and column k, denoted as $M[j, k]$, where $j \neq k$, is the cosine distance (shown as Eq. (1)) between the vectors of the jth word and the kth word.

Given two vectors A and B, the cosine distance between them is a value between 0 and 1 computed by the following equation.

$$\text{similarity} = \text{cosSimi}\,(A,\ B) = \frac{A \cdot B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\,\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

The vector of the jth word in a sentence is denoted as $vec(s[j])$; in our expedient, it is an existing value already computed by GloVe. Therefore, the input matrix of a sentence s for the semantic model can be defined as follows:

$$M[j][k] = \text{cosSimi}\big(\text{vec}(s[j]), \text{vec}(s[\text{k}])\big) \tag{2}$$

4.1.2 The convolution layer of the semantic model

The convolution computation is shown in Fig. 2. We propose two ways of convolution that are called the convolution on sub-sentences and convolution on words.

The convolution on sub-sentences is shown in the upper part of Fig. 2. Each square marked with a red border is the shifting window of a convolution filter. Given a sentence with *len* words, the windows of the filters of the convolution on sub-sentences all have a width 2, but with different heights ranging from *len* to 2.

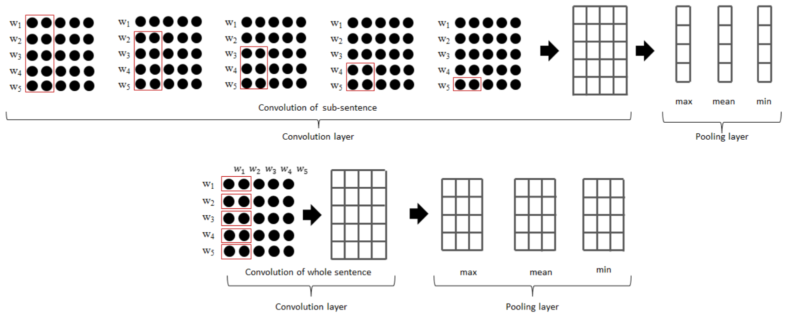

Figure 2: The semantic model

A convolution filter F is defined as a tuple $<t_F, w_F, b_F, h>$, where t_F is the height of the shifting window of F whose width is always 2; $w_F \in R^{t \times 2}$ is the weight matrix for the filter; $b_F \in R$ is the bias of the filter; h is the activation function that is the same for all filters in our model. In our experiments, the choice of the activation function h is the sigmoid function, which is shown in Eq. (3), where z is an input value, and $h\ (z)$ is a value between 0 and 1.

$$h(z) = \frac{1}{1+e^{(-z)}} \tag{3}$$

For a sentence with len words, the output of the convolution using a filter F, denoted as out_F, is a vector of $len-1$ values, since the shifting window of the filter will appear in $len-1$ positions from left to right. The jth entry of out_F is defined by Eq. (4), where $X[j] \in$ t × 2 is the matrix of the window of the filter at the jth shifting position.

$$out_F[j] = h(W_F \cdot X[j]) + b_F \tag{4}$$

Given a filter F, out_F represents some potentially valuable summary of the semantic information of the sub-sentence at F; hence, it is called the convolution on sub-sentences. Since there are $len-1$ filters, and each filter outputs a vector or $len-1$ entries, the output of the convolution on sub-sentences using all the $len-1$ filters will be a matrix of $(len-1) \times (len-1)$ entries.

The other way of convolution is the convolution on words, as shown in the lower part of Fig. 2, where each filter has its height 1 and width 2. The computation output at each window is the same as Eq. (4), while the shifting window size is different from those for the convolution on sub-sentences. In the convolution on words, a filter will shift its window from left to right, and produce a vector of $len-1$ values, which represent some potentially valuable summary of the semantic meaning of a word against each word in the sentence, therefore the name "convolution on words". The output of this convolution using all the len filters will be a matrix of $len \times (len-1)$.

4.1.3 The pooling layer of the semantic model

For the output matrix M (with $len\text{-}1$ rows), in the pooling layer, three functions of max, mean, and min are applied to M. For a function $p \in \{\max, \min, \text{mean}\}$, $p(M)$ is a vector of $len-1$ entries, whose jth entry is computed by applying p to the jth row of m, denoted as $p(M[j])$. Therefore, 3 vectors of $len-1$ values are obtained, as shown in the upper-right part of Fig. 2.

For the $len \times len-1$ output matrix of the convolution of words, we choose each small area of 2×2 in the output matrix to apply the max, min, and mean functions. Therefore, we obtain three matrices, each has $(len\text{-}1) \times (len\text{-}2)$ entries.

4.2 Syntactic model

We believe that when a sentence is parsed according to some deep understanding of NLP, the syntactic association between words can show important logic and meanings between words, which are important aspects of semantics. Therefore, we construct a neural network especially designed for exploiting the syntactic information of the words in a sentence. Other researches on sentence similarity do not use syntactic information as we do; for example, the work of the MPCNN paper [He, Gimpel and Lin (2015)] does not consider the syntax of a sentence at all. Utilizing syntactic information should be a reason that supports the advantages of the performance of our solution.

4.2.1 The Stanford CoreNLP syntactic relationships between words

We use the software package provided by the *Stanford CoreNLP* library [Manning, Surdeanu, Bauer et al. (2014)] to lexically analyze a sentence to obtain its tokens and syntactically analyze the tokens to label the syntactic relationships between them. There are more than 50 relationships between two words that *Stanford CoreNLP* can identify; some of them are listed in Tab. 1 as examples. Further descriptions of these relationships and their labels can be found on the website of *Stanford CoreNLP*[1].We use these relationships to construct the input matrix to the syntax model.

Table 1: Some syntactic relationships between words in a sentence

Name	Description
amod	Adjectival modifier; An adjectival modifier of an NP is any adjectival phrase that serves to modify the meaning of the NP.
nsubj	Nominal subject; A nominal subject is a noun phrase which is the syntactic subject of a clause. The governor of this relationship might not always be a verb: when the verb is a copular verb, the root of the clause is the complement of a copular verb, which can be an adjective or noun.
nmod	Modification of compound nouns.
case	Object pointing preposition.
det	Determiner; Decision words, such as articles, etc.

Fig. 3 shows an example of the syntactic relationship between the words of a sentence that are labelled by the analysis of the *Stanford CoreNLP* software package.

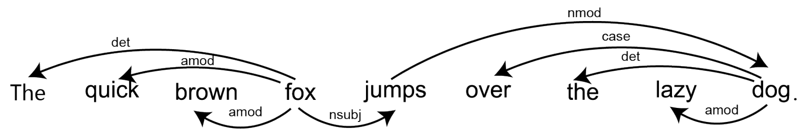

Figure 3: An example showing the syntactic relationships labeled by Stanford CoreNLP between the tokens in a sentence

4.2.2 Input matrix to the syntactic model

Given a sentence, afterword segmentation, suppose it has *len* words, we use the CoreNLP package to obtain the labeled relationships between the words of the sentence. Suppose the *j*th word and the *k*th word of the sentence have some relationship that is labeled as *L*, this fact is denoted as:

$$rela[j][k] = L$$

Otherwise, if there is no relationship labeled between the two words, it is denoted as:

$$\text{rela}[j][k] = \varepsilon$$

[1] *stanfordnlp.github.io/CoreNLP*

The items at the diagonal of the matrix are specially chosen as the TF-IDF values [Wu, Luk, Wong et al. (2008)] of each word in the sentence so that an item $M\ [j, j]$, for some j, is a value between 0 and 1 that represents the semantic weight of the jth token in the sentence with the consideration of the entire corpus in the background.

The computation formula of a TF-IDF value is described in the following. Considering the jth word of a sentence s, denoted as $s[j]$, its frequency in s, denoted as tf (j, s), is the number of occurrences of the word $s[j]$ in s divided by the number of words in s. Given a corpus c, the number of sentences in c is denoted as $|c|$. Here we consider each sentence is a document in c. The document frequency of a word x in a corpus c is the number of documents (sentences) in c where x appears, denoted as df (x, c). The Inverse Document Frequency of s[j] in c is computed by Eq. (5):

$$\mathrm{idf}(j, s, c) = log_e\left(\frac{|c|}{\mathrm{df}(s[j], c)}\right) \tag{5}$$

The TF-IDF value of the s[j] of a corpus c is computed by Eq. (6):

$$\mathrm{tf\_idf}(j, s, c) = \mathrm{tf}(j, s) \times \mathrm{idf}(j, s, c) \tag{6}$$

We consider using TF-IDF values at the diagonal is much more meaningful than using the cosine distance values between words that will always be 1 at the diagonal. This special choice of diagonal values makes a remarkable contribution to the performance of our algorithm. The related experiment details will be elaborated in Section 7.1.

A $len \times len$ matrix can be constructed for the sentence as follows:

$$\mathrm{M[j][k]} = \begin{cases} \mathrm{if}_{\mathrm{idf}(sen, j)}, & \mathrm{if}(j = k) \\ \mathrm{cosSim}(\mathrm{vec}(\mathrm{sen}[j]), \mathrm{vec}(\mathrm{sen}[k])), & \mathrm{if}\big(j \neq k, and, (\mathrm{rela}[j][k] \neq \varepsilon, or, \mathrm{rela}[k][j] \neq \varepsilon)\big) \\ 0, & \mathrm{else} \end{cases} \tag{7}$$

In the example of Fig. 3, there are 9 tokens, so, the input matrix M to the syntactic model is (9×9). Since there is a relationship from the second word to the first word, $M[1][2]=M[2][1]= \mathrm{cosSim}(\mathrm{vec}(\mathrm{sen}[1]), \mathrm{vec}(\mathrm{sen}[2]))$. Since there is no relationship between the 3rd and the 9th words, $M\ [3][9]=M[9][3]=0$. Again, the diagonal entries of the input matrix will be the TF-IDF values of the tokens.

4.2.3 The convolution layer of the syntactic model

The convolution process of the syntactic model is shown in Fig. 4. Comparing to the semantic model, the differences of the convolution process in the syntactic model is that, there are no multiple convolutions on different perspectives since there is only one filter used, which has a shifting window of size 2×2. Therefore, given a sentence with len tokens, the convolution of the syntactic model will output a matrix of size $(len-1)\times(len-1)$. The pooling computation is the same as the pooling computation after the word convolution of the semantic model (the lower part of Fig. 2), where a window of 2×2 is used to apply the three pooling functions. Therefore, the pooling layer will output 3 matrices of size $(len-2)\times(len-2)$.

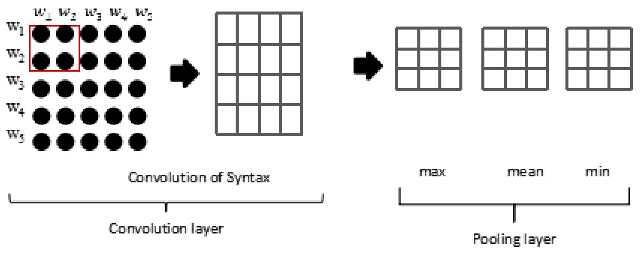

Figure 4: The convolution and pooling layers of the syntactic model

5 The similarity computation of sentence representations

Given a sentence, the output of the syntactic model and semantic model includes vectors and matrices. A long vector is formed by concatenating all these vectors, and the columns of these matrices, following a certain order. This is done by the full connection layer in the neural network. Given two sentences, if they have different numbers of tokens, then their vectors will have different lengths. We add trailing 0's to the shorter one to make the two vectors equal in length. We call such a long vector computed based on a sentence the representation vector of the sentence. We apply the cosine similarity formula (shown in Eq. (1)) to the two representation vectors of the two sentences to obtain the similarity score between them.

6 Comparing the model computation of SSCNN with MPCNN

The convolution in multiple perspectives in the semantic model is designed to capture the potential semantic meanings of the sentences, which is similar to the MPCNN algorithm [He, Gimpel and Lin (2015)]. The sub-sentence filters of our solution are similar to the holistic filer mentioned in the MPCNN paper, but we choose sub-sentence filters with different heights and a fixed-width, while the holistic filters in the MPCNN paper have fixed height but with variable widths. Our word filters are similar to the per-dimensional filters in the MPCNN paper, but our filters have a fixed width, while the per-dimensional filters have variable widths.

The way that we choose the pooling functions is also similar to the MPCNN paper. There are two pooling functions, *max* and *min*, used by MPCNN, while we have three, with *mean* as the added one.

Comparing with MPCNN, the computation of SSCNN is much simpler. In the MPCNN paper, convolutions are done separately to each different pooling function, while in our work the pooling functions all based on the same convolution computation. This simply implies that our solution is easier and faster to be implemented and executed. Experiments show that our solution, while simpler, has even better performance than MPCNN.

Most of all, the input matrix to the neural networks of our work is completely different from the input matrix of the MPCNN paper, in which each column is the word vector of a word. We consider the special design of the input matrix in our work is crucial for the excelling performance of our solution.

7 Experiments

In this section, we introduce the settings and results of the experiments, and how we designed experiments to optimize the algorithm.

7.1 Implementation and comparison methods

The authors of the original MPCNN paper published their code and data at an open repository on GitHub. However, the code is based on the PyTorch platform, while we prefer the TensorFlow platform for its popular technical convenience.

There is a recently published project on GitHub that implements the algorithm of MPCNN on TensorFlow; we call it the MPCNN2018[1] project. We found that the quality of MPCNN2018 is notably good comparing with other recent implementations of MPCNN. We have scrutinized the code of MPCNN2018 and provided a considerable amount of updates to make sure that the model computation process of the MPCNN algorithm is exactly implemented on the TensorFlow platform. We have avoided some mistakes in the setting that of MPCNN2018, such as including the testing data as part of the training data. We used the same dataset that is used in the MPCNN2018 project. The code of our implementation of MPCNN, which we call the MPCNN2019 project[2], has been uploaded to the open repository on GitHub.

Our experimental setting has differences comparing with the original MPCNN paper. For example, our training data sets are different; also, some customized loss functions are used in the experiment setting of the original MPCNN paper, while we used a common Mean Square Error (MSE) loss function. These differences do not affect the validity of our experiment results regarding to comparing the performance of the algorithm of MPCNN and our algorithm, since the code of MPCNN2019 has correctly implemented the convolutional neural network algorithm of MPCNN, and in our experiments the two algorithms, MPCNN2019 and SSCNN, are tested with the same data sets and the same experimental settings.

We implemented the neural networks using the framework of TensorFlow [Abadi, Barham, Chen et al. (2016)] with version 1.4. TensorFlow has a feature that significantly simplifies the implementation work; for example, it provides the default implementation of the standard computations of CNN, such as the gradient descent computation, the loss function, etc.

7.2 Preparing the source data

We prefer to use some larger and more recent data sets than those used in the MPCNN project where the used data set is SemEval 2014; in comparison, we used all the data sets of SemEval from 2012 to 2017.

We downloaded the files of the pretrained word vectors of English words from the GloVe [Pennington, Socher and Manning (2014)] website. We used the vectors of the words in a sentence to construct the input matrices to the neural networks.

[1] https://github.com/Fengfeng1024/MPCNN

[2] https://gitlab.com/rudyshine/SSCNN

The same set of word vectors are applied to the SSCNN and MPCNN models in the experiments for fairness of the comparison. The choice of using the pretrained word embeddings of GloVe provides some technical convenience and is sufficient to show the advantage of our model in the comparison. The output of Word2Vec varies on different datasets. Since the word embeddings are used in the two parts of the sentence model, we consider that the usage of word embeddings is more extensive in our model than other models, and our model will be more sensitive to the quality of word embeddings. If we compute the word embeddings using the Word2Vec algorithm directly on our experiment datasets and obtain word embeddings with better quality, we expect that the advantage of our model could be more obvious.

The training data of the neural networks in our experiment are gathered from the SemEval system[1], which includes all the training data that are published by SemEval for semantic similarity analysis from 2012 to 2017, totally about more than 20,000 pairs of sentences. For each pair of them, a similarity score from 0.0 to 5.0 is already marked.

The testing data file is published by SemEval in 2017 (2017.dev), which includes more than 1000 pairs of sentences.

7.3 Choosing the best configuration of input matrices

In the early stage of the experiments, we trained the SSCNN model with 20,000 pairs of sentences without special considerations of the diagonals at the input matrices; i.e., the diagonals of input matrices of the semantic and syntactic models have all entries as 1.

With the model trained like this, we tried different combinations to construct the input matrices for the convolutional neural networks. One choice is about to use the semantic model only, or the syntax model only, or both. Another choice is about putting the TF-IDF values on the diagonal (left-top to lower-bottom) of which input matrix; we can choose to do it only to the input matrix to the semantic model, or only to the matrix to the syntactic model, or both, or none. For a matrix whose diagonal does not have the TF-IDF values, all entries on the diagonal are 1.

Therefore, 8 different settings of these choices are tried. For each setting, we tested it with 400 pairs of sentences, and we record how many pairs of sentences have the correct similarity score (comparing the closest integers of the computed score and the already marked value). The results are shown in Tab. 2. The entry marked with the description string "(*semantic*)+(*syntactic diagonal*)" means that we chose to run both the semantic model (without the TF-IDF diagonal on its matrix), and the syntactic model (with the TF-IDF diagonal). The settings of the other entries in Tab. 2 are marked with the corresponding description strings.

From Tab. 2, the best result belongs to the first row, which means that the best setting could be that we run both the syntactic and semantic models and put the TF-IDF diagonal only on the semantic model. Therefore, the SSCNN models are settled with this optimized setting and trained again. All the later comparison experiments are based on the optimized models, whose results are described in the rest of this section.

[1] https://en.wikipedia.org/wiki/SemEval

Table 2: Experimenting different settings of the input matrices

Model setting	**Total sentence pair num**	**Success sentence pair num**	**Precision rate**
(semantic diagonal)+(syntactic)	400	348	87%
(semantic)+(syntactic diagonal)	400	332	83%
(semantic)+(syntactic)	400	324	81%
(semantic diagonal)	400	312	78%
(semantic)	400	299	74%
(semantic diagonal)+(syntactic diagonal)	400	154	38%
(syntactic diagonal)	400	120	30%
(semantic)	400	103	26%

7.4 Experiment results and analysis

7.4.1 Results of running SSCNN

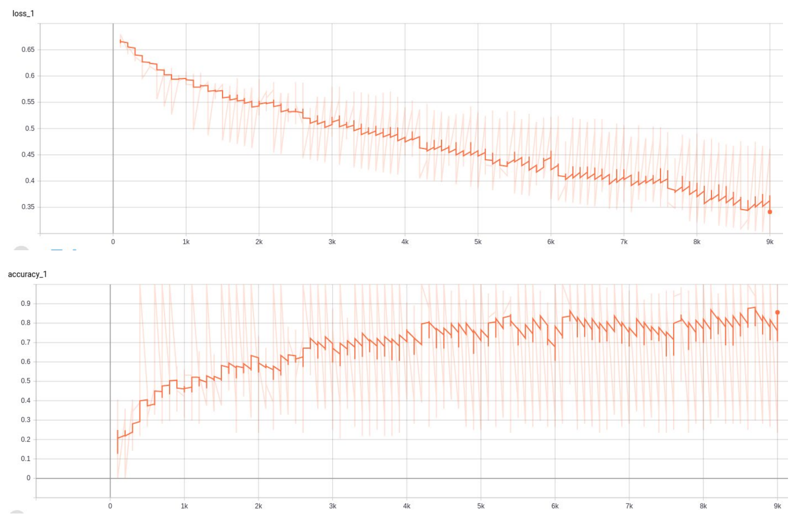

Figure 5: Results of training the SSCNN model

Fig. 5 shows the results of training the SSCNN model with 20,000 pairs of sentences in 9,000 steps. The pictures are automatically generated by the TensorBoard module of the TensorFlow system. For each step, a set of 100 (the batch size) pairs of sentences are randomly chosen for the training computation. Among these 100 pairs of sentences, the computation results of 20 pairs of sentences are randomly chosen to draw the graph. The dark lines show the average values and the light lines show the highest and lowest values of every range of steps, with some chosen small range size.

Given a batch of n sentence pairs, for the t^{th} pair among them, for some number t, its labeled similarity value and computed similarity value are denoted as O_t and P_t

respectively. The Mean Square Error of the computation on a batch of *n* sentence pairs is described by the following equation:

$$\mathrm{MSE} = \frac{1}{N}\sum_{t=1}^{n} (\mathrm{O}_t - \mathrm{P}_t)^2 \tag{8}$$

The upper part of Fig. 5 shows the loss values at different steps computed by Eq. (8). A loss value shows some overall status of the errors of all the experimental samples (at a step). Therefore, with some higher loss value, the model is more error-prone at the step.

The lower part of Fig. 5 shows the accuracy rate, which is the number of pairs of sentences with correct computed similarity scores divided by the total number of samples (100).

We can see that for the SSCNN model, at the beginning of the training the model produces many errors. At the 9000*th* step, the results become stabilized at the best level.

7.4.2 Results of running the MPCNN model

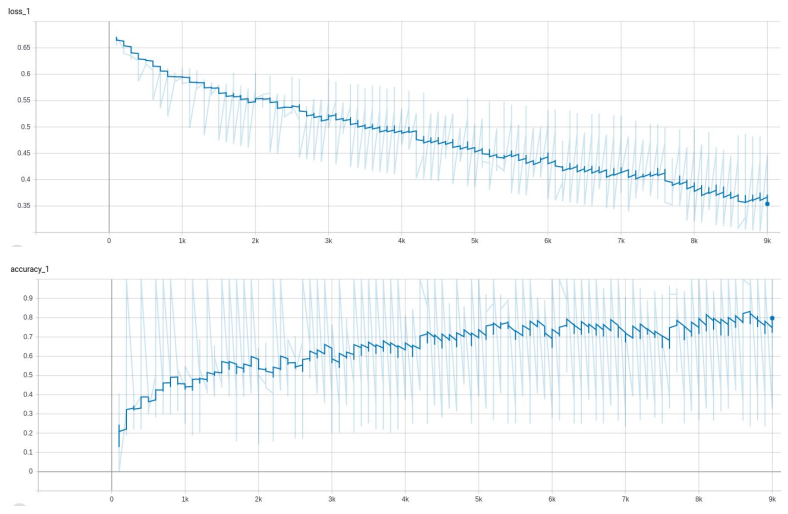

Figure 6: Results of training the MPCNN model

Fig. 6 shows the results of running the MPCNN model (code of the MPCNN2019 project) using the same experiment setting for the SSCNN model (Fig. 5), which means the same training data sets, same experiment parameters, and the same text segmentation process and Word2Vec library provided by Stanford CoreNLP.

7.4.3 Comparing the performance of the two models

The experiment results in Figs. 5 and 6 are summarized in Tab 3.

Table 3: Experiment results of comparing the two models

Condition of collected experiment data	Number of correct sentence pairs in training	Accuracy of training	Number of correct sentence pairs in testing	Accuracy of testing
Both models are correct	24611/29239	84.17%	808/1217	66.39%
MPCNN is correct	26073/29239	89.17%	889/1217	73.05%
SSCNN is correct	26504/29239	90.65%	927/1217	76.17%

Comparing the two models, we can see that the SSCNN model has the following advantages over MPCNN:

● More accurate. The SSCNN model stabilizes at the accuracy rate of around 76.17%, while the stabilized accuracy of the MPCNN model is around 73.05%.

● Less variance. If we print the values of the samples as dots the graph, we can see that the track of dots for SSCNN is more condense, while MPCNN's graph is sparser. It suggests that the performance of SSCNN has less variance and more stable.

● Faster. SSCNN surpasses MPCNN on accuracy at the step 2,000, when the achieved accuracy of SSCNN is about 60%. In contrast, MPCNN needs about 2700 steps to achieve 60% accuracy. It suggests that less computation time is required by SSCNN to obtain some nearly optimal results. Also note that, since the convolution of SSCNN is much simpler than MPCNN, the computation time needed for each step is significantly less than MPCNN.

8 Summary and future work

We designed an algorithm (SSCNN) based on CNN for sentence similarity computation. Comparing with MPCNN, which is the most notable CNN-based algorithm for sentence similarity computation, SSCNN has simpler computation and better performance in terms of accuracy.

In the future, we want to improve SSCNN by better utilization of syntax features. Although the algorithm has achieved positive effects by utilizing syntactic features of the sentences, there are a lot of useful syntactic features that are ignored by the algorithm.
The direction and type of the relationship between two tokens are totally ignored in the input matrix to the syntactic model. We believe that if we can use these syntactic features more sufficiently, we can construct some more effective algorithms.

We expect that SSCNN has potentials that are especially suitable for an IQA environment [Cui, Xiao, Wang et al. (2017)], for the following reasons: First, SSCNN requires less amount of computation, which can support the demand of quick response of Human-Computer Interaction in an IQA environment. Second, in an IQA environment, the question sentences of users can be restricted to, or translated to, some certain forms of grammar structure for the convenience of the computation of an IQA system. According to our observation, when the grammar features of sentences are similar, the performance of SSCNN to compute the semantic similarity of the sentences is especially good. Third, for an IQA, the question sentences are likely to relate to a certain domain of purpose or knowledge. In this scenario, if some computation method is designed with the consideration of the whole corpus, such as the TF-IDF formula adopted by the SSCNN model, its performance will be especially good.

This research is partly motived by solving some practical problems of an IQA environment. We plan to integrate the algorithm deeply in an IQA with features of perpetual learning [Du (2018)]. Such a system will be updated manually or automatically upon the failure and success cases of processing question sentences. For example, if two words have similar meanings while the system judges them dissimilar, a library of similar word pairs can be updated to mitigate this problem. The SSCNN algorithm can evolve in an environment of an IQA system, which can work as an experimenting platform for the SSCNN algorithm. When the IQA and the SSCNN are deployed, the IQA will keep on processing more data through interaction with users, which means that the SSCNN model will keep on improving itself by continuous training.

We expect that our approach to design SSCNN based on CNN can be similarly applied in a future task to design a better algorithm based on another model of neural networks, such as LSTM.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A. et al. (2016): TensorFlow: a system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation*, pp. 265-283.

Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. (2017): Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135-146.

Cui, W.; Xiao, Y.; Wang, H.; Song, Y.; Hwang, S. W. et al. (2017): KBQA: learning question answering over QA corpora and knowledge bases. *Proceedings of the VLDB Endowment*, vol. 10, no. 5, pp. 565-576.

Du, Z. (2018): From one-off machine learning to perpetual learning: a STEP perspective. *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 17-23.

Goodfellow, I.; Bengio, Y.; Courville, A. (2016): Deep Learning. *MIT Press*.

He, H.; Gimpel, K.; Lin, J. (2015): Multi-Perspective sentence similarity modeling with convolutional neural networks. *Conference on Empirical Methods in Natural Language Processing*, pp. 1576-1586.

Hochreiter, S.; Schmidhuber, J. (1997): Long short-term memory. *Neural Computation*, vol. 9, no. 8, pp. 1735-1780.

Kim, Y. (2014): Convolutional neural networks for sentence classification. *Empirical Methods in Natural Language Processing*, pp. 1746-1751.

Manning, C.; Surdeanu, M.; Bauer, J.; Finke l, J.; Bethard, S. et al. (2014): The Stanford CoreNLP natural language processing toolkit. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60.

Marelli, M.; Bentivogli, L.; Baroni, M.; Bernardi, R.; Menini, S. et al. (2014): SemEval-2014 Task 1: evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. *8th International Workshop on Semantic Evaluation*, pp. 1-8.

Marneffe, M. C. D.; Manning, C. D. (2008): The Stanford typed dependencies representation. *Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pp. 1-8.

Mueller, J.; Thyagarajan, A. (2015): Siamese recurrent architectures for learning sentence similarity. *Thirtieth Aaai Conference on Artificial Intelligence*, pp. 2786-2792.

Pennington, J.; Socher, R.; Manning, C. (2014): Glove: global vectors for word representation. *Proceedings of the Empirical Methods in Natural Language Processing*, pp. 1532-1543.

Siegelmann, H. T.; Sontag, E. D. (1995): On the computational power of neural nets. *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 132-150.

Tai, K. S.; Socher, R.; Manning, C. D. (2015): Improved semantic representations from tree-structured long short-term memory networks. *Accepted for publication at ACL 2015*, vol. 1, pp. 556-1566.

Wagner, W. (2010): Natural language processing with python, analyzing text with the natural language toolkit. *Language Resources and Evaluation*, vol. 44, no. 4, pp. 421-424.

Wu, H. C.; Luk, R. W. P.; Wong, K. F.; Kwok, K. L. (2008): Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems*, vol. 26, no. 3, pp. 1-37.

Zeng, D.; Liu, K.; Lai, S.; Zhou, G.; Zhao, J. (2014): Relation classification via convolutional deep neural networks. *25th International Conference on Computational Linguistics*, pp. 2335-2344.