Semantic Similarity Detection Using Convolutional Neural Network(CNN) and LSTM

Quach Manh Cuong
quachmanhcuong03@gmail.com

Vu Bao Chau
baochauvu124@gmail.com

Abstract

Semantic similarity detection is a crucial task in natural language processing that involves measuring the likeness between two pieces of text. In this research, we explore and compare the effectiveness of two distinct models for semantic similarity detection: Long Short-Term Memory (LSTM) and a combination model comprising Convolutional Neural Network (CNN) and LSTM.

1 Introduction

Semantic Similarity Detection (SSD) holds significance in various Natural Language Processing (NLP) applications, including information retrieval, classification, extraction, question answering, and plagiarism detection. The SSD task evaluates the similarity between two texts by assigning a continuous score between 0 and 1. A score closer to 0 suggests greater independence in the semantics of the sentences, whereas a score closer to 1 indicates likely semantic equivalence. SSD poses a challenge due to the multitude of ambiguities and synonymous expressions in languages, coupled with the variability in sentence lengths and complex structures.

Hence, fundamental models like bag-of-words or TF-IDF models (as exemplified by arXiv:1810.10641v1 [cs.CL] on October 24, 2018) are limited by their specificities, which overlook the significance of word order and neglect both syntactic and semantic relationships. Recent advancements in sentence similarity employ Neural Networks, including Recurrent Neural Networks (RNNs) as seen in works by Mueller & Thyagarajan (2016), Kiros et al. (2015), Tai et al. (2015), and Convolutional Neural Networks (CNNs) like the approach by He et al. (2015). Neural Networks (NNs) conduct in-depth analyses of sentences and words, enabling a more comprehensive consideration of both sentence semantics and structure for improved sentence similarity prediction.

In this paper, we describe our technique based on NNs to measure similarity. Siamese CNN will help analyze the local context of words in a sentence and generate a representation of the relevance of a word and its neighborhood. Siamese LSTM analyzes the entire sentence based on its words and its local contexts. Manhattan distance is used as a function to calculate the similarity of the pair of sentences.

This paper is organized as follows: we make an overview of relevant work for SSD in Section 1.1. Next, we detail our experiments in Section 2, and 3 as Preprocessing Data, Models, and Evaluation. The conclusion will be presented in Section 4. All the references will be noted in Section 5.

1.1 Related Work

CNNs have demonstrated remarkable success in tasks such as classification (Kim, 2014) and other natural language processing (NLP) applications (Collobert et al., 2011). He et al. (2015) developed a sentence embedding approach using a Siamese CNN architecture, incorporating various convolution and pooling operations to capture different levels of information granularity. Their convolutional filters analyze entire word embeddings and each dimension of word embeddings across multiple window sizes. Multiple pooling types (max, mean, and min) are applied to the output of the convolution operation. Ultimately, they predict sentence similarity by comparing local regions of sentence representation through various measurements, including horizontal and vertical comparisons.

2 PreProcessing

2.1 Data

The source of our test data is the Microsoft Research Paraphrase Corpus (MRPC) dataset consisting of 5801 sentence pairs collected from news. Each pair has an annotation indicating paraphrased or non-paraphrased pairs. The

whole dataset is split into 3576/500/1725 for training/dev(validation)/test subset.

The data file has five columns, they are Quality, ID1, ID2, Sentence1, and Sentence2 respectively.

quality	id1	id2	sentence1	sentence2
1	702876	702977	Amrozi accused his brother, whom he called "th...	Referring to him as only "the witness", Amrozi...
0	2108705	2108831	Yucaipa owned Dominick's before selling the ch...	Yucaipa bought Dominick's in 1995 for $693 mil...
1	1330381	1330521	They had published an advertisement on the Int...	On June 10, the ship's owners had published an...
0	3344667	3344648	Around 0335 GMT, Tab shares were up 19 cents, ...	Tab shares jumped 20 cents, or 4.6%, to set a ...
1	1236820	1236712	The stock rose $2.11, or about 11 percent, to ...	PG&E Corp. shares jumped $1.63 or 8 percent to...

Figure 1: Dataset

2.2 Tokenization

Tokenization is a fundamental step in NLP that involves breaking down a text into individual units, typically words or subword units. In this experiment, we will be using the Tokenizer class in the spaCy library. The model 'en_core_web_lg' allows the tokenizer to intelligently segment text into tokens, considering linguistic intricacies like contractions, punctuation, and special characters.

tokenized_sentence1	tokenized_sentence2
[Amrozi, accused, his, brother,, whom, he, cal...	[Referring, to, him, as, only, "the, witness",...
[Yucaipa, owned, Dominick's, before, selling, ...	[Yucaipa, bought, Dominick's, in, 1995, for, $...
[They, had, published, an, advertisement, on, ...	[On, June, 10,, the, ship's, owners, had, publ...
[Around, 0335, GMT,, Tab, shares, were, up, 19...	[Tab, shares, jumped, 20, cents,, or, 4.6%,, t...
[The, stock, rose, $2.11,, or, about, 11, perc...	[PG&E, Corp., shares, jumped, $1.63, or, 8, pe...

Figure 2: Tokens

In our models, stop words will be eliminated as they considered to be of little value in conveying meaningful information. Stop words are frequently used words that generally do not contribute much to the overall meaning of a sentence. Examples of stop words in English include "the," "and," "is," "of," and "in."

After tokenization, the next step is encoding - converting these tokens into numerical indices in the vocabulary that we created before. Each unique token is assigned a unique index, and the sequence of tokens in a text is represented as a sequence of corresponding indices. This transformation allows the model to work with the numerical input.

Zero padding is the next step we do. In the context of sequence data processing, such as in natural language processing tasks, it is a common practice to pad sequences to ensure uniform length. Padding

tokenized_sentence1	tokenized_sentence2
[22607, 2273, 2809, 5482, 6009, 22608, 7963, 2...	[22611, 3187, 2814, 4030, 6009, 22612, 10256, ...
[83, 6774, 366, 598, 22614, 166, 203, 146, 132...	[22615, 1327, 71, 1975, 14946, 784, 190, 5604,...

Figure 3: Encoding

involves adding elements of a specific value (usually zero) to sequences that are shorter than the maximum length, ensuring that all sequences have the same length. In this case, all sequences are padded to match with the length of the longest sequences which is 30.

```
[22611  3187  2814  4030  6009 22612 10256  3187  1409  4030  7963 22613
   299  2362 22610    18     0     0     0     0     0     0     0     0
     0     0     0     0     0     0]
```

Figure 4: After-padding example

2.3 Embedding

Embedding is a fundamental concept in natural language processing and machine learning that plays a pivotal role in transforming raw, discrete data into continuous vector representations. In the context of text and language, embedding refers to the process of mapping words, phrases, or entire documents onto high-dimensional vectors, capturing semantic relationships and contextual nuances.

In this research, we use pre-trained word vectors from Word2Vec to create an embedding matrix that serves as a lookup table for words in a model, enabling the model to represent words as continuous vectors with semantic meaning. The matrix will be used in the embedding layer in our models.

3 Models

The models we used in this experiment are Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN).

LSTM *(Sepp Hochreiter and Jürgen Schmidhuber, 1997)* is a type of Recurrent Neural Network (RNN) that is specifically designed to handle sequential data, such as time series, speech, and text. LSTM networks are capable of learning long-term dependencies in sequential data. LSTMs introduces a memory cell, which is a container that can hold information for an extended period of time. The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell.

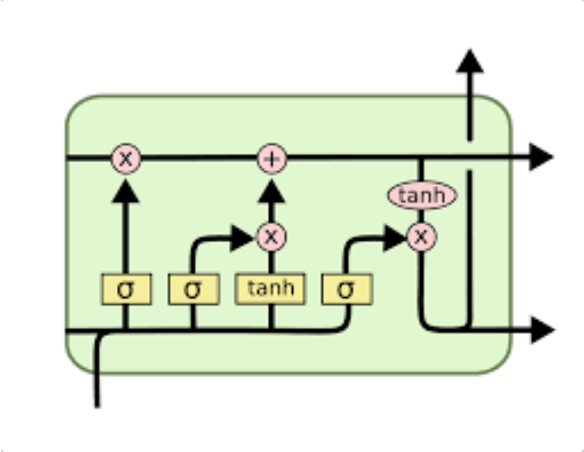


Figure 5: LSTM network

Convolutional neural network (CNN) is a regularized type of feed-forward neural network that learns feature engineering by itself via filters (or kernel) optimization. Vanishing gradients and exploding gradients, seen during backpropagation in earlier neural networks, are prevented by using regularized weights over fewer connections. CNNs have accomplished excellent outcomes in classification and other NLP tasks. CNNs uses filters that analyze entire word embeddings and each dimension of word embeddings with multiple window sizes.

Various similarity metrics, including cosine, Euclidean, and Manhattan distances, were examined, and the most favorable result was obtained using the Manhattan distances.

3.1 LSTM

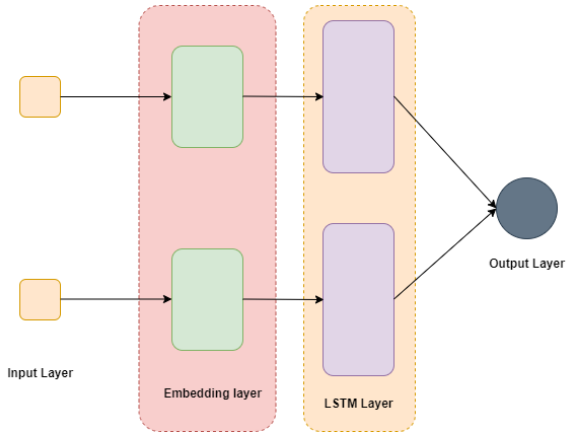


Figure 6: MaLSTM

The input layer contains two sequences as two 30-dimensional vectors. After taking the input as sequences after preprocessing, the input will pass

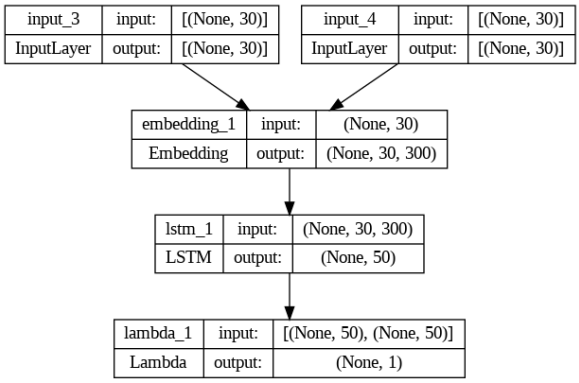


Figure 7: Structure of MaLSTM

to the embedding layer where it will convert them to a matrix with a size of (30,300). Each row represents a word with an embedding vector - 300 dimension so that a sentence (in MRPC data) will be represented as a matrix (30x300).

After that, the two matrices representing two input sequences then go through the LSTM layer which has 50 units - the number of memory cells or neurons in the LSTM layer.

The last layer has a 1-value output which is the Manhattan distance of the two sequences. The output has the range of (0,1) and will be compared to the target which is 1 or 0.

The model uses Adam optimizer, MSE loss, and Accuracy metric and is trained with batch size = 32, 20 epochs.

3.2 Combination of CNN, LSTM

Kim trained a simple CNN on top of pre-trained word vectors for the sentence classification task (Kim, 2014). His simple model composed of one layer of convolution achieved excellent results on multiple benchmarks. Inspired by the good results of CNNs in sentence classification (Kim, 2014), we use a Siamese CNN to generate local contexts for each word in a sentence from its previous and following words. We utilize pre-trained word embeddings Word2Vec to represent these words. Let $w_{ei} \in R^k$ be the k-dimensional word vector corresponding to the i-th word in a sentence. A local context of length l (e.g., $l = 5$) is represented as:

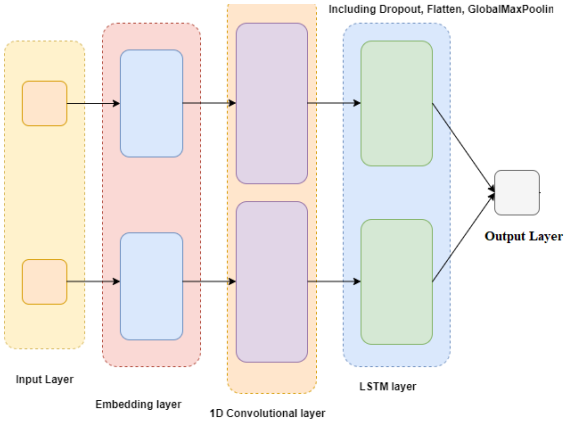
$$x_{li} = x_{i-2} \oplus x_{i-1} \oplus x_i \oplus x_{i+1} \oplus x_{i+2} \quad (1)$$

where $\oplus$ is the concatenation operator. Our convolution operation involves a filter $w \in R^{lk}$, which is applied to a window of l words to produce a local

context. In more detail, our CNN generates the local context of word i by:

$$lc_i = f(w \cdot x_{li} + b) \quad (2)$$

where b is a bias term and f is the hyperbolic tangent function. This filter is connected to every sequence of words in a sentence to deliver a local context for all words.



3.3 Evaluation values

[h]

In this experiment, we evaluate our model's performance by recording its accuracy and F1 score. The formula for these are as shown below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

where

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

3.4 Results

First, we executed the MaLSTM model and found that the model performed poorly on the test set. So we decided to create the CNNLSTM which is more complex in structure by adding a convolutional layer before it.

The second model show a significant improvement when comparing to the first one. The accuracy and F1-score computed on the test set are shown in the table below.

To sum up, the local context of words refined the general context analysis. Our approach identified more details about the words and their local as well as general contexts, which usually leads to improved SSD results.

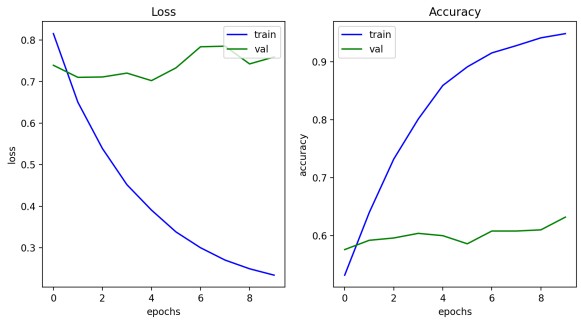


Figure 8: MaLSTM training

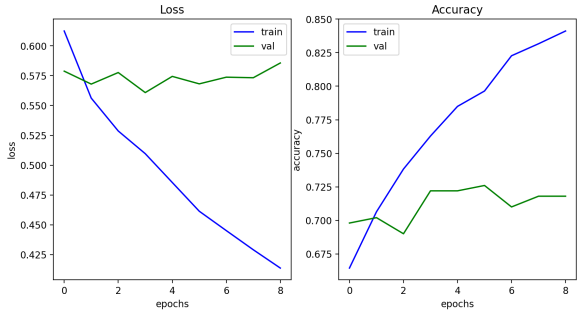


Figure 9: CNNLSTM training

	MaLSTM	CNNLSTM
Accuracy	0.607	0.700
F1 score	0.714	0.806

Table 1: Results on test set

4 Conclusion

Semantic Similarity Detection (SSD) holds significance in various Natural Language Processing (NLP) applications, including Automatic Text Summarization (ATS), Question-Answering, and Information Retrieval. Our system integrates CNN and LSTM structures to effectively analyze, identify, and preserve pertinent information within sentences and across entire sentences. Leveraging local context proves beneficial in obtaining additional information about words in a sentence, thereby enhancing sentence analysis. Through our experiments, the inclusion of local context has resulted in improved prediction of sentence similarity, char-

acterized by a reduction in mean squared error and an augmentation of correlation scores.

5 References

ELVYS L. P., STÉPHANE H. ANDRÉA C. L., JUAN-MANUE T. (2018). Predicting the Semantic Textual Similarity with Siamese CNN and LSTM.

COLLOBERT R., WESTON J., BOTTOU L., KARLEN M., KAVUKCUOGLU K. & KUKSA P. (2011). Natural language processing (almost) from scratch.

KIM Y. (2014). Convolutional neural networks for sentence classification.

KIM G. (2021). Spoiler Detection.

TAI K. S., SOCHER R. & MANNING C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks.

HE H., GIMPEL K. & LIN J. J. (2015). Multi-perspective sentence similarity modeling with convolutional neural networks.

MUELLER J. & THYAGARAJAN A. (2016). Siamese recurrent architectures for learning sentence similarity.

KIROS R., ZHU Y., SALAKHUTDINOV R., ZEMEL R. S., TORRALBA A., URTASUN R. & FIDLER S. (2015). Skip-thought vectors.