Semantic Textual Similarity

A Dissertation Submitted to the University of Hyderabad

in Partial fulfillment of the Degree of

Master of Technology

in

ARTIFICIAL INTELLIGENCE

by

Shivam Soni

18MCMI26

SCHOOL OF COMPUTER AND INFORMATION SCIENCES

UNIVERSITY OF HYDERABAD

(P.O.) Central University, Gachibowli,

Hyderabad – 500 046

Telangana, India

June 2020

CERTIFICATE

This is to certify that the dissertation entitled, **"Semantic Textual Similarity"** submitted by **Shivam Soni**, bearing Regd. No. **18MCMI26**, in partial fulfillment of the requirements for the award of **Master of Technology** in **Artificial Intelligence** is a bonafide work carried out by him under my supervision and guidance.

The dissertation has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Signature of the Supervisor

Prof. Kavi Narayana Murthy

School of Computer and Information Sciences

University of Hyderabad

Signature of the Dean

Prof. Chakravarthy Bhagvati

School of Computer and Information Sciences

University of Hyderabad

DECLARATION

I, **Shivam Soni**, hereby declare that this dissertation titled, **"Semantic Textual Similarity"** submitted by me under the guidance and supervision of **Prof. Kavi Narayana Murthy** is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or other University or Institution for the award of any degree or diploma.

Signature of the Student

Date: **Name:** Shivam Soni

Regd. No.: 18MCMI26

ACKNOWLEDGEMENT

I am extremely thankful to my supervisorsProfessor Kavi Narayana Murthy for his motivation and tireless efforts to help me to get deep knowledge of the research area and supporting me throughout the life cycle of my M. Tech. dissertation work. Especially, the extensive comments, healthy discussions, and fruitful interactions with the supervisors had a direct impact on the final form and quality of M. Tech. dissertation work.

I am also thankful to to Prof. Chakravarthy Bhagwati Head of Computer and Information Sciences, for his fruitful guidance through the early years of chaos and confusions. I wish to thank the faculty members and supporting staff of Computer Engineering Department for their full support and heartiest co-operation.

This thesis would not have been possible without the hearty support of my friends. My deepest regards to my Parents for their blessings, affection and continuous support. Also, Last but not the least, I thank to the GOD, the almighty for giving me the inner willingness, strength and wisdom to carry out this research work successfully.

(Shivam Soni)

ABSTRACT

Semantic Textual Similarity(STS) is the task of measuring the semantic similarity between two snippets of text. For instance, given two sentences, the idea is to compute upto what degree the two sentences are similar in meaning. The degree of similarity is measured on a scale of 0 to 5 (real values), according to the shared tasks competition held by SemEval [4] , where a score of '0' represents complete unrelatedness and '5' represents semantic equivalence. We have employed a hybrid architecture which combines various *Knowledge and Corpus Based techniques* [52] with Neural based work, which is basically a Convolutional Neural Network followed by a Fully-Connected-Layer inspired from [53]. Moreover, we have enhanced the *Knowledge and Corpus Based techniques* called *WordNet-Augmented Word Overlap* and *Vector Space Sentence* presented in [52]. In *WordNet-Augmented Word Overlap*, we have replaced *pathlen similarity*(used to calculate the similarity between the senses in the WordNet) with: *Jiang- Conrath similarity*(JCS) [24] and *Lin Similarity*(LS) [35] and in *Vector Space Sentence Similarity*, we have replaced LSA word vectors with vectors given by Google's pre-trained Word2Vec model [41]. We have also tried various modifications in CNN architecture like changing the pre-trained vectors, loss functions and number of hidden layers and hidden unit. After extracting the features for the input pair of sentences using our combined architecture, a regressor model was trained that outputs the final similarity score. With our final hybrid architecture we are able achieve better results than some of the benchmark results as per SemEval 2017 [13].

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Recent researches have shown a need of measuring semantic similarity between the sentences. There are various applications where STS task can be applied such as for Text Summarization, Machine Translation, Question Answering , Dialogue and Conversational Sytems and Text Generation. In text mining, it can be used to locate unseen text from the data [6]. STS is related to various other tasks such as Paraphrase Detection [64] [18], Textual Entailment [37] and Semantic relatedness [37].

A lot of work using Knowledge Based and Corpus technique(KACB) has been done in the past for finding the similarity between two sentences. The Knowledge Based techniques commonly use the lexical database WordNet [43] as the source and includes various semantic similarity measures such as *Pairwise Word Similarity* ([7], [39], [52]). [56] made use of word alignment technique which used Paraphrase Database (PPDB), which is a large lexical and phrasal database. The Corpus Based techniques involves measures such as Pointwise Mutual Information (PMI) [39], Vector Space Similairty [52], LSA models [32] etc. [7] in their work pointed out some limitaions in the earlier methods like [17] [16] [34] [32] [39] and one of the limitaions among them was that these earlier measures were under the assumption that a single measure can itself capture all the required information out of the sentences and were therefore unable to get high accuracy for computing the similarity. To overcome this limitation, they combined various *content based*, *style based* and *structure based* similarity measures and achieved one of the best performances in the STS task-2012.

Lately, various deep learning techniques became very competitive with earlier knowledge based and corpus based techniques. [21] in their work gave one top most results in 2015 by proposing a Multi- Perspective CNN (MPCNN) model. The same authors in [23] improved their technique by adding an interaction layer(based on the attention models) before passing the input sentences in MPCNN model. [22] proposed a combination of Bi- LSTMs and Deep CNNs for measuring the similarity between the sentences. Various other neural based techniques like [8], [44], [53] started taking over the older techniques and set the new benchmarks.

1.1 Our Contribution

As discussed above, [7] observed that the earlier methods like [17] [16] [34] [32] [39] had a limitation due their assumption, that a single measure in itself is enough to capture the semantic relations between sentences and therefore they combined various techniques in their proposed work. In our work, we have followed a similar notion of combining various techniques for the task of measuring similarity between sentences. We have proposed a hybrid architecture which combines various *Knowledge and Corpus Based techniques* (KACB) [52] with Convolutional Neural Network followed by a Fully-Connected-Layer inspired from [53]. The combined output was then finally passed into a regressor model to produce the final output. Our overall view of the system can be understood from the figure 7.1. Moreover, we have modified two techniques presented in [52] which are, *WordNet-Augmented Word Overlap* and *Vector Space Sentence Similairty.* In *WordNet-Augmented Word Overlap*, we have replaced *pathlen similarity*(used to calculate the similarity between the senses in the WordNet) with: *Jiang- Conrath similarity*(JCS) [24] and *Lin Similarity*(LS) [35]. The drawback of *pathlen similarity* is that it assigns same similarity score to different pairs of concepts without considering the generality or specificity of the pairs in WordNet heirarchy. Whereas, JCS and LS similarities rely on WordNet structure and adds probabilistic information derived from a corpus. Also, we have made various modifications in the CNN architecture by changing the loss functions and pre-trained word embeddings.

We have opted for $KACB$ technique presented in [52] because they have proposed one the best $KACB$ based measures and are still used among benchmark results among all KACB techniques. We chose CNNs over RNNS or LSTMs because, it has been observed through various researches (section 4.1.4) that CNNs have proved to work better in cases where we have a fixed length for input sentences, whereas RNNs or LSTMS works best when the input sentences have dynamic lengths. In the SemEval 2017 dataset [13] which we have considered in our work for the task measuring similarity between sentences, the lengths of the sentences among each other does not vary much, i.e., the standard deviation of the lengths of sentences is very less, so CNNs works better here . And we chose the CNN architecture presented by [53] among various others like [21], [23], [22] etc. because it showed that rather than having a deep complex architecture for computing semantic relation between sentences, a simple architecture can also work very well. Their results are among the top five benchmark results in 2017 SemEval dataset [13]. More details on why and how the various modification have been carried out in our work, are discussed in Section 7.

1.2 Thesis Organization

In the next section 2 we will discuss how the STS task is basically defined and what is the metric used for comparison. In section 3 we will discuss about the background of STS task. In section 4 we will discuss about the methods which we have followed in our proposed work(section 7). Also, before the proposed work we have discussed about various optimization techniques and word embeddings used in our work, in sections 5 and 6 respectively. At last we will discuss about all the experiments and conclude our work and talk about some future work in section 8.

Chapter 2

Problem Definition

Semantic Textual Similarity (STS) task follows the idea that, given two sentences, compute upto what degree the two sentences are similar. For instance, given two sentences, the idea is to compute upto what degree the two sentences are similar in meaning. The degree of similarity is measured on a scale of 0 to 5 (real values), according to the shared tasks competition held by SemEval (International Workshop on Semantic Evaluation) [4] , where a score of '0' represents complete unrelatedness and '5' represents semantic equivalence. For example (from [3]) :

- A score of '*5*' can be given if two sentences means exactly the same thing, as shown below:

  Bob is diving into the pool.

  Bob is jumping into the swimming pool.

- A score of '*3*' can be given if two sentences roughly means the same thing but some information is missing in one or the other, as shown below:

  Akshay said chapter 5 is very important than chapter 3 from the book.

  "Chapter 3 is not so important." Akshay said.

- A score of '*0*' can be given if two sentences are completely independent from each other, as shown below:

  John is going for a walk with his friends.

  There is bird flying in the sky.

The dataset released by SemEval for STS task from the year 2012-2017 contains gold standard labels for every pair of sentences which is annoted by various experienced annotators ([4] [3] [2] [1] [10] [13]). To measure the performance of any model, i.e, how well a particular model has captured the semantic similarity between two sentences , **Pearson Correlation Coefficient** is used as standard across all datasets. SemEval had also released various datasets for measuring similarity between cross-lingual sentenses [2] [1] [10] [13]. In this work we have worked on monolingual task of measuring similarity between two English sentences.

Chapter 3

LITERATURE REVIEW

3.1 A Quick Survey of the Evolution of the STS Task

A rise in the research on measuring similarity between two sentences mainly started when SemEval (International Workshop on Semantic Evaluation) announced Semantic Textual Similarity shared task, which successfully ran from the year 2012 to 2017 [4] [3] [2] [1] [10] [13]. Before the SemEval STS task was announced, a substantial amount of work had been done for measuring the similarity between the pair of texts but it was mainly focused on comparison between two long texts or documents [5] [20] [30] [38] and very few publications related to measuring similarity between short texts were reported [16] [34] [32] [39]. And the methods used for comparing long texts are generally inefficient when applied on short text because two similar long texts are usually dependent on degree of co-occuring words. Whereas, in short texts co-occuring words maybe less or null sometimes. Therefore,in this field, some of the earliest works can be considered to be given by [16] [34] [32] [39]. Also, [34] [32] were among the first to provide dataset for this task. Where [34] contributed 65 English sentence pairs and [32] contributed 50 document pairs suitable for the STS task. There are various different kinds of measures that has been used till now to compute the similarity between a pair of sentences, where, the most commonly used measures for a long period of time were based on knowledge based and corpus based techniques(Section 3.2). We have also discussed about word alignment technique in the same section. Recently, the impact of Deep Neural Nets based techniques proved quite effective, which is discussed in Section 3.3.

3.2 Knowledge Based, Corpus Based and Word Alignment techniques

The work done by [32] used of Binary, Count,and Latent Semantic Analysis (LSA) similarity models. The corpus was represented as a $m \times n$ matrix, where each value t_{ab} counts the number of times $a-th$ word or n-gram occurs in the $b-th$ document. For the Binary model value t_{ab} simply represented whether the a-th word or n-gram is present in the b-th document or not. And then used Common Features Model [31],Tversky's (1977) Ratio Model [60] and Distinctive Features [45] to to measure similarity between a pair of text or document in their case. And for the Count similarity model they used Jaccard, Cosine and Overlap correlation models to measure similarity between the pairs. For LSA models also they Cosine similarity to calculate similarity after the SVD decomposition of the word-document matrix. [39] in their work used *Corpus-based Measures* (which included Pointwise Mutual Information (PMI) [59] and LSA [29]) and *Knowledge-based Measures* which makes use word similarity metrics: Leacock and Chodorow [15], Lesk [33], Wu and Palmer [63], Resnik [49], Lin [35], Jiang and Conrath [24]. These word similarity metrics make use of lexical database (Wordnet database in thier work). The PMI technique and word similarity metrics were used to find similarity between each word from the first text segment and each word from the second text segment which had the same part-of-speech tag. Then using a similarity metric, which they proposedin their work [39] , they determined the similarity score between the two texts. [34] followed a step by step process, where they first created an *order vector* and a *raw semantic vector* by using a lexical database(they used Wordnet) and a joint word set formed by the distinct words from both the sentences. Then the *raw semantic vector* was converted into a *semantic vector* by using Brown corpus. Finally, the sentence similarity was computed by combining semantic similarity(calculated using two semantic vectors) and word order similarity (calculated using two word order).

There were some limitaions in the techniques discussed above [17] [16] [34] [32] [39] as observed by [7] in its paper. Firstly, these measures were under the assumption that a single measure can itself capture all the required information out of the sentences. Secondly, these measures assumed that the similarity between the two sentences are completely dependent on the contents of the sentences itself, without considering any other text characterstics. So, in their work they combined various *content based*, *style based* and *structure based* similarity measures which resulted in more than 300 score vectors (out of which 20 features were finally used), which when combined together, served as a feature set for a simple log-linear classifier . Their results gave one of the best performances in the STS task-2012. The *content based measure* includes character n-grams which are compared using the implemantaion given by Barron and word n-grams which are compared using Jaccard coefficient. It also included various *String Similariy Measures* like, *longest common substring* and *longest common subsequent* measure. Similar approach of combining various measures was followed by [52] , and by the combination of various knowledge based, corpus based and syntactical techniques which they proposed, a feature set of scores was constructed. These extracted feature set was used to train a SVR model, which finally predicted sentence similarity score. Some examples of their knowledge based techniques are: *WordNet-Augmented Word Overlap*, *Weighted Word Overlap etc.*, all them used *WordNet* as their lexical database. The corpus based techniques like : *Vector Space Sentence Similarity* and *Weighted Vector Space Similarity* used LSA vectors. In the year 2014, [56] gave one the best and successfull measure for the STS task using word alignment technique presented by them in [55]. Their work follows an alignment pipeline, where, in the first step, all the Identical Word Sequences are aligned and in the second step, all the Named Entities are aligned. In the third and the fourth step content words are aligned basically using two properties of the words: 1. if they are semantically similar, 2. if they occur in similar contexts in respective sentences. The context of the word pairs from the two respective sentences is defined by using syntactic dependencies in the third step and by taking a surrounding window of size 3 in the step four. The final score is then computed using harmonic mean of $prop_{Al}^{(1)}$ and $prop_{Al}^{(2)}$, where $prop_{Al}^{(1)}$ is the proportion of content words in $S(1)$ that are aligned and vice-versa for $prop_{Al}^{(2)}$.

3.3 Deep Neural Network Based Work

Lately, deep learning related work started giving competion to various syntactical and lexical based techniques.One of the top performances were given by [21]. They developed a *sentence model* using a Convolutional Neural Network(CNN) architecture in which they used two different types of filters. The first one was a holistic filter, which by the help of a particular window size matches the complete word vectors and the second one was a per-dimensional filter which independently match each dimension of word embeddings. The output of the sentence model after passing it through various different types of pooling layer was given into a *similarity measurement layer*, which compared different local region of the sentence representation, and finally a fully-connected layer was employed at the end to get the final result. In 2016, they released another STS measurement techniques [22], which gave even better results. Initially, a Bidirectional Long Short-Term Memory Nework (Bi-LSTMs) was used to model the context of the input sentences and then using a *Pairwise Word Interaction Model* each word from first sentence was compared with each word of the second sentence and 13 different compared values were computed, which resluted in a 3D matrix of size $13 \times |sent1| \times |sent2|$. This was finally passed through a *focus layer* and then into 19 layer Deep ConvNet to produce the final output. It was the first paper ever, which combined the use of both Bi-LSTMs and Deep CNNs for any task. The multi-perspective CNN model (MPCNN) [21] discussed above, was modified in [23] by adding an interaction layer before passing the input sentences in MPCNN model. This modification was mainly done to make the input sentences more inter-related, because earlier, the model lacked contextual interaction information between the sentences. [8] in 2017 presented an end-to-end LSTM system which was a version of Siamese LSTM [44]. In Siamese LSTM [44], an LSTM model is employed for each input sentence, weights are shared between layers and updated parallelly during learning phase. Whereas, [8] allowed two sentences to be given as an input into the same LSTM model and shared thw weights. This work by [8] also gave quality results. An another paper in 2017 [53], gave a simple CNN model without adding any dropout or batchnorm layer and passed the output of the CNN model into a Fully-Connected Neural Network (FCNN) to get the final scores.

Chapter 4

Methods

4.1 Convolutional Neural Network

Convolution in most simple terms is a mathematical operation on two functions which computes a third a fuction, expressing how a particular function can bring changes to the other function. It has been used in various fields like, *probability, statistics, signal processing, computer vision, natural language processing and differential equations.* In our work we have used CNNs for NLP task and we will be discussing about it in section 4.1.4. But before that we have discussed about 1-D, 2-D Convolutions and CNNs in following sections.

4.1.1 1-Dimensional Convoluation

In 1-D Convolution we compute the current value of the input based on the previous set of inputs. Which can be expressed as weighted average of all the given inputs:

$$s_t = \sum_{a=0}^{\infty} w_{-a} x_{t-a} = (x * w)_t \tag{4.1}$$

where t referes to the reference point or simple terms,a particular time step.a is the index of the weight, ranging from 0 for reference point to ∞. Although, in practice we would not take the reading up till -∞ , we can simply declare all the unwanted weights as zero. w refers to the set of weights which can be learned depending on the task in hand.

Some applications of 1-D Convolution can seen in 1-D Convolutional Neural Network(CNN), which can be used in situations where *the location of feature itself, is not important.* For example, evaluating signal data over a fixed- length period like an audio recording or evaluating time series of sensor data. Also , now 1-D CNN have proved working well in NLP tasks, where we have short length sentences.

4.1.2 2-Dimensional Convolution

The basic definition for 2D-Convolution is also the same,which is the weighted average of all the previous inputs to find the current value. Where we can learn the weights depending on the task and procedure to follow to learn them. But the above definition is easy to visualize in case of 1-D input but not in the case of 2-D.

Therefore, in 2-D we consider neighbors along the rows and columns, as shown in the following equation:

$$s_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i+a,j+b} * K_{a,b} \tag{4.2}$$

where, K refers to kernel or weights and I is the input. And $*$ is the convolution operation. a refers to the number of rows and b refers to the number of columns. m and n specify the size of the matrix.

The equation 4.2 looks different than 4.1, because here not only we are considering previous values but all the surrounding values for a particular point.

4.1.3 2-D Convolutional Neural Network

CNNs are a variety of different convolution layers with *non-linear activations* like ReLU, tanh or sigmoid computed on to the output of each of them. Earlier, when CNNS were not so popular, a lot of research in *computer vision* was focused on what kind of different filters can be used to extract features out of an image or any type of input given. But with CNNs, by the help of various loss functions, it learned the weights for the filters by its own and was able to perform much better.
CNNs are able to deal with various shortcomings of Deep Neural Nets (DNNs) with all fully-connected layers. In DNNs we have a multiple fully-connected layers which

results in too many parameters, because of which they are often prone to overfitting. So, the most important questions that arose was, can we have DNNs which can have many non-linearity, but have fewer parameters and hence less prone to overfitting? The answere is yes, and all these problems are resolved by CNNs. Formally, two main aspects of CNNs are **Sparce Connectivity** and **Weight Sharing**, which makes them perform better than DNNs. We discuss the aspects of CNNs below:
Sparse Connectivity: In traditional feedforward neural network, each neuron of the output layer is connected to each neuron of the previous layer.Which finally results in more number of parameters and hence overfitting. Whereas, in CNNs one particular neuron in the output is a result of convolution of filters with a specific region of the input. Therefore, it is connected only to some neighbouring neurons of the input, which provides sparse connectivity and results in *fewer number of parameters.*
Weight Sharing: In CNNs, **weights are nothing but the kernel**, and the **kernel remains constant as we pass over the entire input matrix**, creating different output values based on the neighborhood of inputs. One complete pass of the kernel over the input matrix constitutes one Convolutional output. As we know, the kernel is constant, thereby **the weights get shared for all the neurons in that output area.** Only the inputs vary.
Thereby, we are effectively reducing the number of parameters yet still retaining model complexity(many non-linearity) and overcoming one of the shortcomings of DNNs.

4.1.4 Why and How are CNNs related to NLP tasks

CNNs are usually used for images as input. But when used for texts, the document or sentence is represented in a matrix form. As an example, for a sentence of length 15 words and each word represented as a vector of dimension 300, the input would be a 10×300 matrix.
In NLP, generally, filters slide over full dimensions of words, i.e., they move only in 1-dimension, taking two words pr three words at a time depending on the size of window. A full picture of how it works has been shown in figure 4.1, which has been taken from [65]. We can see that, 3 different choices of window sizes for filter has been used ,1st with a window size of two, 2nd with a window size of three and

4th with a window size of four, with 2 filters for each. Therefore we get 6 outputs, 1 from each of the filters, which is then max-pooled and concatenated to give final classification among two categories. For intuition it can looked upon as extracting n-grams features out of a sentence.

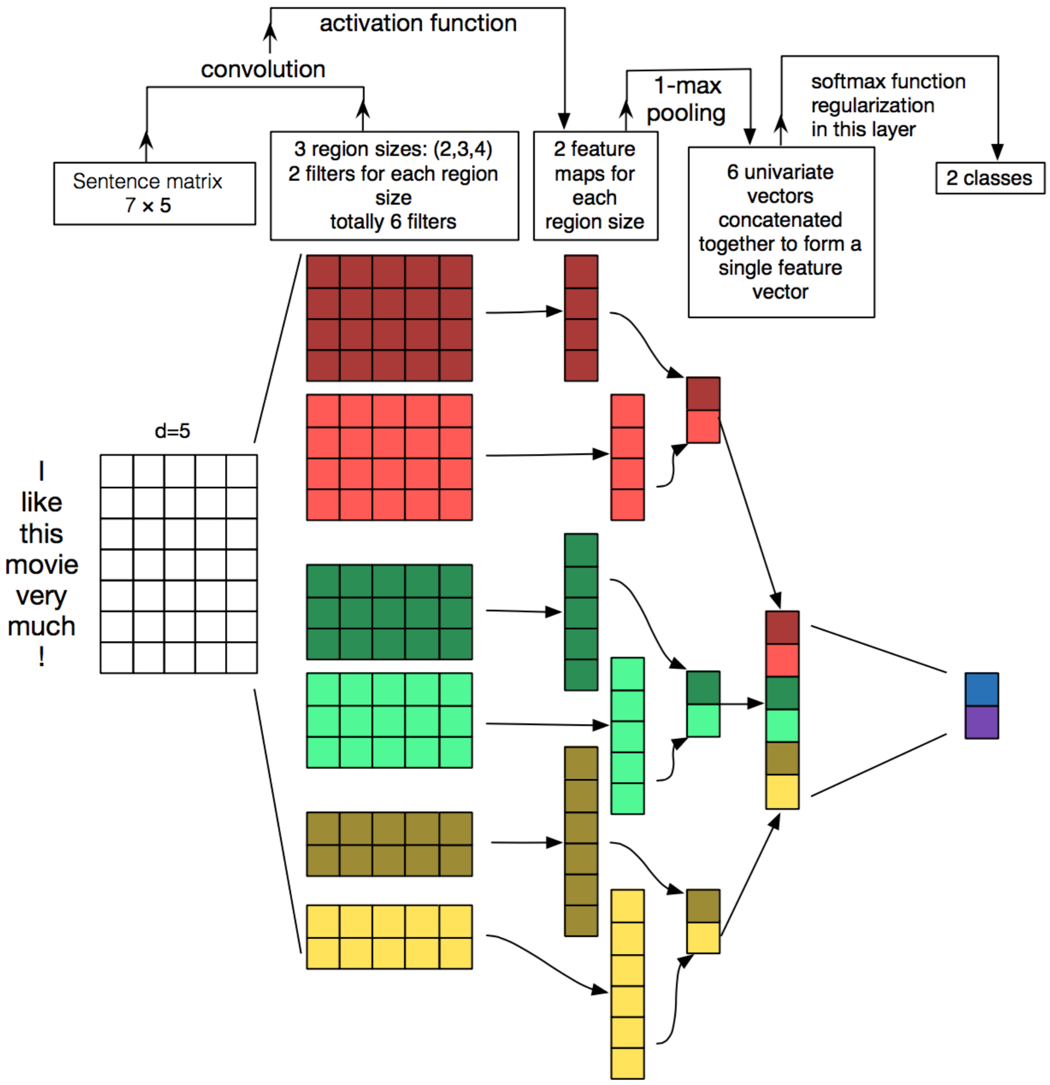

activation function
convolution
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
1-max pooling
6 univariate vectors concatenated together to form a single feature vector
softmax function regularization in this layer
2 classes
d=5
I
like
this
movie
very
much
!

Figure 4.1: CNNs for Sentence classificationc [65]

Yes, we can say that, there are not very clear intuitions of how CNNs provide benefits for local invariance and what exactly higher-level representation mean in case of NLP as in computer vision. So, it may seem that Recurrent Neural Networks (RNNs) works better as it is more intuitive in case of texts. But this does not mean that CNNs do not work at all. Many models might not work well but some have proved to be very usefull in recent studies. A big defence for CNNs is, they are very fast and are very comfortably used at hardware level on GPUs. If compared to n-grams , CNNs are more efficient in terms of representaion, because a large size of vocabulary is computionally expensive, even Googlge has not provided anything greater than 5-grams , but with CNNs we can even work with 6-grams or greater than that. The filters are automatically able to learn good representations, without a need of any large vocabulary. We have discussed below about different applications for CNNs on NLP tasks.

A Quick Survey on CNNs for NLP tasks

It has been observed that CNNs seems to work quite well for classification tasks, such as Sentimentlal analysis, Topic Categorization or Spam Detection. [28] in his work employed a very simple CNN architecture on different classification dataset and still got a very high performance on some of the dataset and state-of-the-art on some of them. They used word2vec word embeddings in their work. A similar approach has been followed by [27] and [61] but with a complex network. They introduced an additional layer for semantic clustering. [25] trained the CNN model without making use of any pretrained word embeddings, rather, trained the model from scratch. They also proposed a memory-efficient bag-of-word technique to represent the input, thereby, decreasing the number of overall parameters required to train the model. [26] prposed an unsupervised "region embedding", using which they extended their model. These region embeddings were learned by the CNN predicting the context of text region. This method which they followed worked well for long texts but for short sentences it did not. The reason behind it which they gave was, maybe, the pre-trained word embeddings,if it would have been used for the representation of words, it would have a better result for short sentences as well. [65] experimented with various different choises of hyper-parameters like: what kind embeddings for words can be used, what kind of poolings to be used and what activations would work best. Some of the important conclusions was, that max pooling works better than average pooling with CNNs on text and regularization techniques does not have a greater impact in NLP tasks. The Microsoft research team [19] [54] proposed a technique on how to learn meaningful semantic representation of sentences for Information Retrieval task using CNN models.

4.2 Knowledge Based and Corpus Based Techniques

The various knowledge and corpus based techniques which we have used in our work are inspired from [52], which we discuss as follows:

4.2.1 N - Gram Overlap

The ngram overlap between two sentences ($S_1\, and\, S_2$) is expressed as follows:

$$ngo(S_1, S_2) = 2.\left(\frac{|S_1|}{|S_1 \cap S_2|} + \frac{|S_2|}{|S_1 \cap S_2|}\right)^{-1} \quad (4.3)$$

where $|S_1|$ and $|S_2|$ is the length of unigrams, bigrams or trigrams in S_1 and S_2 respectively. $|S_1 \cap S_2|$ is the length of common unigrams, bigrams or trigrams between S_1 and S_2.

The idea behind ngram-overlap is, it computes that upto what extent sentences S_1 and S_2 covers each other and then finally computing a harmonic mean between the two quantities.

4.2.2 WordNet - Augmented Word Overlap(WAWO)

Wordnet augmented coverage between sentences ($S_1\, and\, S_2$) is expressed as follows:

$$wn(S_1, S_2) = \frac{1}{|S_2|} \sum_{w_1 \in S_1} score(w_1, S_2) \quad (4.4)$$

$$score(w, S) = \begin{cases} 1 & if\, w \in S \\ max_{\bar{w} \in S}\, sim_{pl}(w, \bar{w}) & otherwise \end{cases}$$

$wn(S_1, S_2)$ in 4.4 finds the wordnet coverage between S_1 and S_2, similarly $wn(S_2, S_1)$ is also calculated between S_2 and S_1, where, finally a harmonic mean is calculated between $wn(S_1, S_2)$ and $wn(S_2, S_1)$ to compute the final score for this method. $score(w_1, S_2)$ helps to find the score between each word from S_1 with each word from S_2. Where $sim_{pl}(w, \bar{w})$ computes the pathlen similarity between w and $\bar{w}$. And pathlen similarity between two concatenated c_1 and c_2 is calculated as:

$$sim_{pl}(c_1, c_2) = \frac{1}{Pathlen(c_1, c_2)}$$

$Pathlen(c_1, c_2)$ = No. of edges in the shortest path (in hypernym graph) between 2 concepts.

4.2.3 Weighted Word Overlap

The following equation computes weighted word coverage of the first sentence S_1 by the second sentence S_2:

$$wwc(S_2, S_1) = \frac{\sum_{w \in S_1 \cap S_2} IC(w)}{\sum_{\bar{w} \in S_2} IC(\bar{w})} \tag{4.5}$$

Similarly $wwc(S1, S2)$ is also calculated and a harmonic mean between $wwc(S_2, S_1)$ and $wwc(S1, S2)$ is computed to get the final score for *weighted Word Overlap* measure.

$IC(w)$ in the above equation 4.5 is the informtion content of a word w from a sentence, which is expressed as follows:

$$IC(w) = ln \frac{\sum_{\bar{w} \in C} freq(\bar{w})}{freq(w)}$$

where $freq(w)$ refers to the frequency of a word in a corpus and C is the a of words from the corpus.

4.2.4 Vector Space Sentence Similarity

In this method LSA vectors are used to represent a word in a sentence and the whole sentence is represented by taking the summation of all the word vectors present in the sentence, expressed as following:

$$\vec{v}(S) = \sum_{w \in S} u_w$$

where $\vec{v}(S)$ is the vector representation of the sentence and u_w is the LSA vector representation of a word w. An additional representation $\vec{v_W}$ is also used, where each word vector is assigned some weight, which is basically the information content of the word w and as expressed as following :

$$\vec{v_W}(S) = \sum_{w \in S} IC(w) u_w$$

Finally, vector space sentence similarity features between the two sentences is computed as $|cos(v(S_1), v(S_2))|$ and $|cos(v_W(S_1), v_W(S_2))|$.

4.2.5 Normalized Difference and Number Overlap

The normalized differences between the two sentences is calculated using three different ways: 1) using the chunks of noun, verb,predicate counts from sentences, 2)using lengths of the sentences and 3) by aggregating the information content of the words in a sentence.
Numbers Overlap includes three different features given as follows:
$(A \subseteq B \vee B \subseteq A)$, $(log(1 + |A| + |B|))$ and $(|A \cap B|)/(|A| + |B|)$
where A and B are the sets of numbers in S1 and S2.

4.2.6 Named Entity Feature

In this feature, they considered capitalized words as named entities and found the overlap between them as done equation 1. They also found the overlap between the stock index symbols as well. Stock index symbols are the tokens which are in all caps and begin with a period.

Chapter 5

Optimization

5.1 Gradient Descent

Gradient Descent(GD) is an algorithm which helps to find a local minimum of a differentiable function, which iterates into the direction proportional to negetive of the gradient from the current state, which eventually converges at a local minimum. The negetive of the gradients provide a direction towards the steepest descent for the current point. In Neural Network, GD is used to update the parametrs (weights and biases) by differentiating the loss function w.r.t parameters as seen in the equation 5.1. The size of each step is decided by the value of Learning Rate η. Therefore, the final equation for gradient descent and parameters w update is defined as:

$$w_{t+1} = w_t - \eta \bigtriangledown w_t \tag{5.1}$$

where, η is the learning rate in range (0,1), t is the current epoch, t+1 next epoch and $\bigtriangledown w_t$ is the derivative of the loss function with respect to the parameter w in current epoch t. This algorithm is used in Neural Networks to update the weights and biases during backpropogation.

5.1.1 Limitaions of Gradient Descent

To start the training of our model we have to initialize our parameters(weights w, biases b) randomly. And if the learning rate η is very large, though, each time larger steps are been taken but there is a risk of overshooting the lowest point and oscilating near that point. So, generally we choose a smaller value of η. But still there is a problem with choosing a lower value of η. The problem is, since we initialize our parameters randomly, it may happen that we get started in a plane or platue kind of surface, and because of a lower value of η the algorithm would need to run many epochs to get out of the plateau region.Therefore to put it together,**the main issue with Gradient Descent is that it takes a lot of time to navigate regions with gentle slopes, because the gradient is very small in these regions**. So, to overcome this problem *Momentum based Gradient Decsent*(5.2) was proposed.

5.2 Momentum based Gradient Decsent

An intuition behind the solution for the above discussed problem would be, that if **algorithm is repeatedly being asked to go in the same direction, then it should probably gain some confidence and start taking bigger steps in that direction.**

To put this intuition in working formula, a history variable v_t is introduced, which records the previous movements of the gradients and updates the present parameters w.r.t. history v_t (5.2) as well as the gradients $\eta \bigtriangledown w_t$ as shown 5.3 :

$$v_t = \gamma * v_{t-1} + \eta \bigtriangledown w_t \tag{5.2}$$

$$w_{t+1} = w_t - v_t \tag{5.3}$$

To look a deeper into how v_t works Consider every instance in time denoted by the subscript, ranging from 0 to t :

$$v_0 = 0$$

$$v_1 = \gamma * v_0 + \eta \bigtriangledown w_1$$

$$v_2 = \gamma * v_1 + \eta \bigtriangledown w_2 = \gamma.\eta \bigtriangledown w_1 + \eta \bigtriangledown w_2$$

$$.$$

$$v_t = \gamma * v_{t-1} + \eta \bigtriangledown w_t = \gamma^{t-1}.\eta \bigtriangledown w_1 + \gamma^{t-2}.\eta \bigtriangledown w_2 + ... + \eta \bigtriangledown w_t$$

Here, we take an Exponentially Decaying Weighted Sum, whereby as we move further and further into the series, the weight decays more. The intuition behind this is as we progress further down a series/direction, we can place lesser importance to the later gradients as we move along the same direction.

5.2.1 Limitations of Momentum Based GD

By making use of history variable to record past movements, though, moment based GD was able to move quickly through gentle slopes, but the problem occured when it entered steep slopes, where due to its quick movement, **starts oscilating in and out of the minima valley (u-turns).** But, despite these u-turns it still converges faster than vanilla gradient descent. Now, we will look at reducing the oscillations in Momentum based GD in the next section.

5.3 Nesterov accelerated gradient descent (NAG)

The Nesterov Accelerated Gradient Descent solves the problem of overshooting and multiple oscillations as follows:
1. Compute w_{temp} based on movement with history.

$$w_{temp} = w_t - \gamma * v_{t-1} \tag{5.4}$$

2. Move further in the direction of the derivative of w_{temp}.

$$w_{t+1} = w_{temp} - \eta \bigtriangledown w_{temp} \tag{5.5}$$

3. Update history with movement due to derivative of w_{temp}.

$$v_t = \gamma * v_{t-1} + \eta \bigtriangledown w_{temp} \tag{5.6}$$

In 5.2 for Momentum Based GD we saw that the movement occurs in two steps: the first is with the history-term $gamma * v_{t-1}$, the second is with the weight term $\eta \bigtriangledown w_t$.So, when moving both steps each time, it is possible to overshoot the minima between the two steps.
Therefore, in NAG the intuition is, **we can consider first moving with the history term, then calculate the second step from where we were located after the first step** w_{temp}. This is represented in equation 5.4, 5.5 and 5.6.

5.3.1 Limitations for NAG

It was observed that while training, NAG and also previously seen momentum based GD, the biases were first getting updated and then weights, which eventually was taking more time than it should have ideally taken. And the main reason behind it was, that since in many problems we have sparse features and biases are dense in nature, so the updates were first happening for biases first and then to the weights.

5.4 Adaptive Learning

Adaptive learning is an another extension on the way of updating weights or biases while backpropogation. In adaptive learning, different learning rate for each parameter(weights), which takes care of the frequency(sparsity/density) of features, is used.

The need for this type learning arose because, we know that the value of the input features plays a important role in the gradient calculation. For example: Let $f(x) = \sigma(wx+b)$, then $\bigtriangledown w^n = (f(x)-y)*f(x)*(1-f(x))*x^n$. Which shows that gradient is dependent on input features x.

In real world scenarios, many features in the data are **sparse** , i.e. they take on a 0 value for most of the training inputs. Therefore, the derivatives corresponding to these 0 valued points are also 0, and the weight update is going to be 0. To aid these sparse features, a **larger learning rate** can be applied to the **non- zero valued point** of these sparse features. Conversely, **dense** features are those with non-zero values for most of the data points. They must be dealt with by using a **lower learning rate.** So, we will look into some techniques for adaptive learning in some of the following sections.

5.4.1 Adagrad (Adaptive Gradients)

The Adagrad (Adaptive Gradient) is an algorithm which satisfies the above intuition discussed in section 5.4 with the equations shown below:

$$v_t = v_{t-1} + (\bigtriangledown w_t)^2 \tag{5.7}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{(v_t)} + \epsilon} \bigtriangledown w_t \tag{5.8}$$

In the equation 5.7, the value v_t increments based on the gradient of that particular iteration, i.e. the value of the feature is non-zero. In the case of **dense features**, it increments for most iterations, resulting in a larger v_t value. For **sparse features**, does not increment much as the gradient value is often 0, leading to a lower v_t value. In the equation 5.8, the denominator term $\sqrt{(v_t)}$ serves to regulate the learning rate η. For **dense features**, v_t is larger, $\sqrt{(v_t)}$ becomes larger thereby lowering η. For **sparse features**, v_t is smaller, $\sqrt{(v_t)}$ becomes smaller and lowers η to a smaller extent. The term ϵ is added to the denominator, to prevent a divide-by-zero error from occuring in the case of very sparse features i.e. where all the data points yield zero up till the measured instance.

Limitations For Adagrad

Yes, using Adagrad we were now able to solve problems caused by sparse feature as we discussed in above section. But **it resulted in a problem for dense features**, because the **learning rate decays very aggressively** and became zero or close to zero as the denominator grows. Consequently, at a particular point near the local minima, dense feature like bias was not able to get updated, therefore, the algorithm was not able to converge at local minima.

5.4.2 RMSProp

RMSProp overcome the problem of aggressively decaying learning rate in case of dense features, by introducing an exponentially decaying sum in the denominator as shown below:

$$v_t = \beta * v_{t-1} + (1 - \beta)(\bigtriangledown w_t)^2 \tag{5.9}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{(v_t)} + \epsilon} \bigtriangledown w_t \tag{5.10}$$

So , now instead simply taking the square of previous gradients for the history variable v_t(as seen in 5.7), RMSProp considers exponentially decaying sum(equation 5.9) in the denominator and is able to obtain a local minimum without getting stuck just near to it as in Adagrad.

5.4.3 Adam

We have already seen Momentum based GD (section 5.2) , where history (**??**) is used to calculate the current update and RMSProp (section 5.4.2), where history 5.9 is used to adjust the learning-rate. Adam basically **combines these two ideas.** which can be seen as follows:

1. m_t a running sum of all the updates done. This is very similar to the history that Momentum based GD maintains.

$$m_t = \beta_1 * v_{t-1} + (1 - \beta_1)(\bigtriangledown w_t) \tag{5.11}$$

2. v_t is used to regulate the learning-rate. This is similar to the history that RMSProp maintains.

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2)(\bigtriangledown w_t)^2 \tag{5.12}$$

3. Here, the first history m_t is used to make the update, ensuring that the history of derivatives is used to calculate the current update. The second derivative v_t is used to regulate the learning rate based on density or sparsity of the feature.

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{(v_t)} + \epsilon} m_t \tag{5.13}$$

5.5 Batch, Mini- Batch and Stochastic GD

So far we have seen what are the better update rules for GD. In this section, we see what combination data should we use for computing the gradients. In *batch GD* we take whole data points at once and keep on accumalating the gradients. Once all the data is exhausted, then we go for updating the parameters.

In Mini- Batch GD the whole data (say n samples), is divided into mini-batch of suppose x data samples. We keep on accumulating the gradients till the time all x data points are exhausted and then update the parameters. Therefore, in one epoch we update the parameters $\frac{n}{x}$ times.

Whereas, in Stochastic GD we update the parameters for every sample point. It is similar to mini-batch GD with mini-batch size=1.

5.6 Non-Linearities

The representation power of a deep Neural Net is due its *Non-Linear Activition functions*, without them NN is nothing but a linear transformation function. We will discuss about various different activations in the following sections.

5.6.1 Logistic Function

Logistic function is also called sigmoid function which is given as :

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5.14}$$

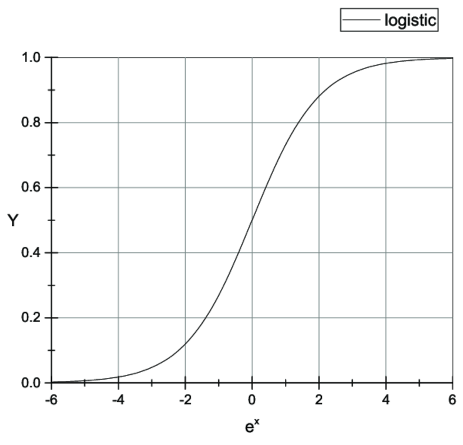


Figure 5.1: Logistic Function

Limitations of Logistic Function

There are three issues with logistic function:
1. **Saturation**: A logistic neuron is said to be saturated when it reaches its peak values. From figure 5.1 we can see that for any larger positive or negetive value the function reaches its peak i.e. 1 and 0 respectively.
We know that, derivative of this function is expressed as:

$$f'(x) = \frac{\partial f(x)}{\partial x} = f(x) * (1 - f(x)) \tag{5.15}$$

So, at peak values, i.e. , when *f(x)=0* or *f(x)=1*, $\mathbf{f'(x) = 0}$. Therefore, because of this saturation issue, when we are calculating the gradient w.r.t. a weight associated with a saturated neuron, the saturated neuron's derivative is 0, thus resulting in the entire gradient becoming 0. This is because the term associated with the saturated neuron in the chain rule for gradient calculation becomes 0, thus making the entire gradient 0. This is called the **Vanishing Gradient Problem** and due to this weights are not updated.

2. **Not Zero Centerd**: A function is said to be *zero centered* if it is spread out equidistant around the 0 point, i.e. it takes an equal number of positive and negative values. The logistic function ranges from 0 to 1, therefore, it is not zero-centered. Suppose, we have a scenario of neural network as shown in figure 5.2

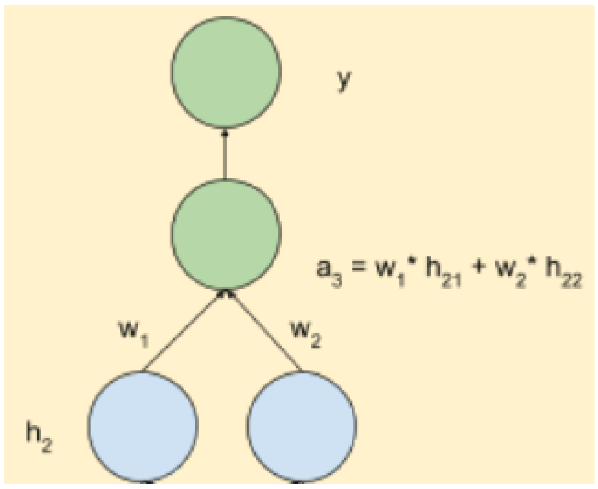
1 

Figure 5.2: NN example

Consider the following gradients:

$$\triangledown w_1 = \left(\frac{\partial L(w)}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} \right) * h_{21}$$

$$\triangledown w_2 = \left(\frac{\partial L(w)}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} \right) * h_{22}$$

Both h_{21} and h_{22} are outputs of the logistic function, so they are always positive (i.e. ranging from 0 to 1). Due to this, at all times, both $\triangledown w_1$ and $\triangledown w_2$ will always be of the same sign, either positive or negative. They cannot be different from each other since the bracketed part is common between them and the logistic function output is always positive. The gradients w.r.t all the weights connected to the same neuron are either all +ve or all -ve. **Thus, this limits the directions in which the weights can be updated** and we cannot arrive at the local minima as fast as possible by moving in all directions.

3. Also, logistic function is **computationally expensive** because of e^x.

5.6.2 Tanh

tanh is expressed as :

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.16}$$

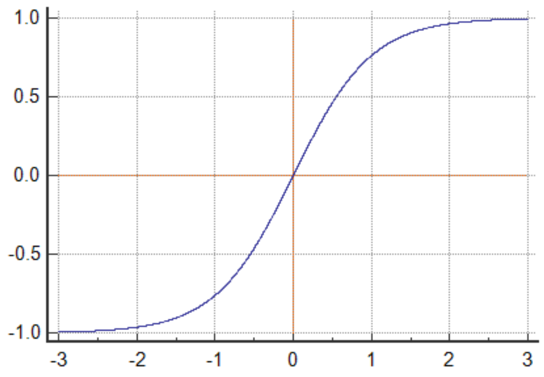


Figure 5.3: tanh Function

We will discuss about those three points which we discussed in section (5.6.1) as follows:

1. **Saturation**: Since, at larger positive or negetive values tanh function also reaches its peak, i.e, -1 and 1 respectively , problem of saturation, which eventually leads to *vanishing gradient problem* still persists.

As the derivative of this function is expressed as:

$$f'(x) = \frac{\partial f(x)}{\partial x} = (1 - (f(x))^2) \tag{5.17}$$

So, at peak values, i.e. , when *f(x)=-1* or *f(x)=1*, $\mathbf{f'(x) = 0}$, therefore, gradients gets vanished.

2. **Zero Centered**: *tanh* is a zero centered function. So, the problem of limiting the direction in which the weights get updated is resolved.

3. Since, e^x is there in the function,therefore, *tanh* is also **computationally expensive**.

However, it is still preferred over the logistic function.

5.6.3 ReLU

ReLU stands for Rectified Linear Unit, which is expressed as follows:

$$f(x) = max(0, x) \tag{5.18}$$

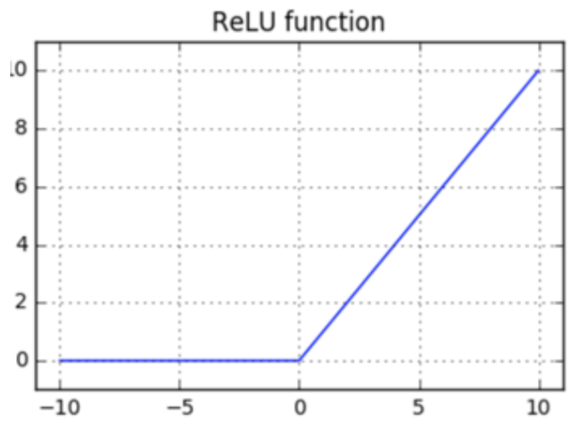


Figure 5.4: ReLU Function

Now the question arises, if still there is any caveat in using ReLU? The answere is yes, which is discussed as below:

1. **Saturation**: There is a problem of saturation of a ReLU neuron in the negative region but not in the positive region, let us see how:

Suppose, we have neuron in the first layer of a NN, and h_1 is the output of the output of the neuron after applying ReLU activation on linear transformation a_1 as shown below:

$$h_1 = ReLU(a_1) = max(0, a_1) = max(0, w_1x_1 + w_2x_2 + b)$$

And if b takes on a large negative value due to a large negative update $\bigtriangledown b$ at some point then:

$$w_1x_1 + w_2x_2 + b < 0[if b << 0]$$

Therefore, $h_1 = 0\,[deadneuron]$ and $\frac{\partial h_1}{\partial a_1} = 0$. This zero derivative is involved in the chain rule for computing the gradient w.r.t $\bigtriangledown w_1$. So, $\bigtriangledown w_1$ becomes 0 leading to the weight not being updated, as in the case of a **saturated neuron**.

2. Since, the function is **not zero centered**, the problem of limiting the direction in which the weights get updated, still persists.

3. But ReLU function is fast and **not computationally expensive.**

5.6.4 Leaky ReLU

It is an extended version of ReLU and expressed as follows:

$$f(x) = max(0.01x, x) \tag{5.19}$$

Leaky ReLu outputs the input value itself if it is positive, else it outputs a fraction of the input value, i.e. f(2) = 2, f(-2) = 0.02.

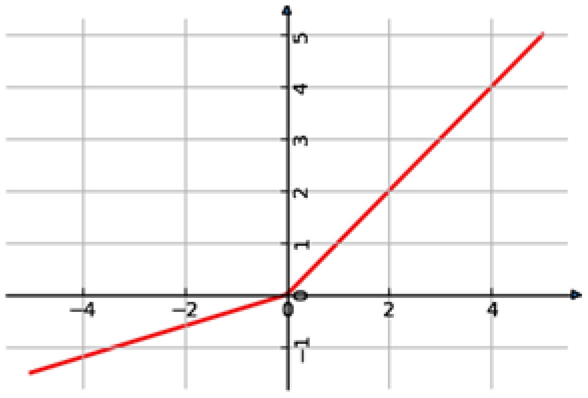


Figure 5.5: Leaky ReLU Function

1. **Saturation**: Leaky ReLU completely solves the problem of vanishing gradients, as there is no saturation issue in it. It does not saturate in the positive or negative region. Therefore, no problem in derivatives as well.

$$f'(x) = \frac{\partial f(x)}{\partial x} = 0.01 if x < 0 | 1\, if\, x > 0$$

The neuron will not die as well, because $0.01x$ ensures that at least a small gradient will flow through. this means that there isn't any 0 valued derivative, thereby ensuring that the gradients are all non-zero. Thus, the weights are always updated.
2. Close to **zero centered** outputs, therefore, no problem here as well.
3. It is easy to compute (no expensive e^x).
Therefore, Leaky ReLU resolves all the issues discussed above.

5.7 Loss Function

5.7.1 Pearson Correlation Coefficient

Pearson Correlation Coefficient (ρ) is a statistical measure that captures linear correlation between two quantitative, continous variables . Its range is between -1 to

+1, where, -1 represents a negetive linear correlation and +1 represents a positive linear correlation. ρ between two random variable (A and B) is given as follows:

$$\rho_{A,B} = \frac{cov(A,B)}{\sigma_A \sigma_B} \tag{5.20}$$

where, σ_A and σ_B represents the standard deviation of A and B respectively and cov is the *covariance* between A and B.
Pearson's correlation coefficient for sample data is represented as r_{xy} and is expressed as follows:

$$r_{ab} = \frac{\sum_{i=1}^{m}(a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{m}(a_i - \bar{a})^2}\sqrt{\sum_{i=1}^{m}(b_i - \bar{b})^2}} \tag{5.21}$$

where, m is the total number of samples, a_i, b_i are the individual points indexed with i and $\bar{a}$, $\bar{b}$ are sample mean.

5.7.2 Kullback-Leibler Divergence(KL-Divergence)

KL-Divergence capture differences between two probability distribution, i.e., it measures how on probability distribution is different from another. If there are no differences,i.e., if probability distribution perfectly matches the KL-Divergence gives a score of zero , otherwise it can take any value *0* and ∞ .
KL-Divergence between two probability distribution *A(i)* and *B(i)* (where *B(i)* is the approximation and *A(i)* is the true distribution is expressed as follows:

$$D_{KL}(A||B) = \sum_{i \in I} A(i) log\left(\frac{A(i)}{B(i)}\right) \tag{5.22}$$

Chapter 6

Word Embeddings

Word embeddings have become an essential part for the representation of words in a form, which can easily be interpreted and understood by our computers, scripts and models. Loosely speaking, it represents of a particular word in a vector form. There are also other forms of embeddings like: character level embeddings and sentence level embeddings, but in our work we have used word embeddings. Different types of text representation, in general, can be classified into two categories as: *Discrete Representation* and *Distributional Representation* of text. Discrete representation of word includes techniques such as : One-Hot encoding, Tf- idf, Count vectorizer. But some major drawbacks with frequency based techniques are, the final word vector representation is very large in dimension, ignores the location information of a word in sentence and is sparse in nature, thereby, lacks relationship between similar words. Whereas, distributional representation of words are mainly based on Neural Nets and are able to generate a dense representation of the final word vectors and with low dimensions. There are many examples of distributional word embeddings such as: word2vec [40], Global Vectors for Word Representation (GloVe) [46], Embedding from Language Models (ELMO) [48], Fasttext [9] etc. But in our work we have employed GloVe, word2vec and Fasttext, which are explained in following section.

6.1 word2vec

In 2013, [40] presented a new word embedding technique called *word2vec*. It actually contains two different methods to obtain the embeddings of the words, known as *Continous Bag of Words* and *Continous Skip- Gram model* which a contains a shallow neural net of just two layers. Some variations in their techniques were made and presented in [41] to improve the overall performance of their model.

6.1.1 Continous Bag-of-Words (CBoW)

The way CBoW works is, it predicts a center/target word w_c, given k words before and k words after the target word w_c, by defining a window of size k (as shown in figure 6.1). The context of the center word is defined as $(w_{c-k}, .., w_{c-1}, w_{c+1}, ..w_{c+k})$, where w_c is the target word. Therefore , the objective function of CBoW is to maximize the probabilty to predict the center word w_c given its context with a window size of k, shown as below:

$$J_\theta = \frac{1}{C}\sum_{c=1}^{C} log\, p(w_c|(w_{c-k}, .., w_{c-1}, w_{c+1}, ..w_{c+k}) \tag{6.1}$$

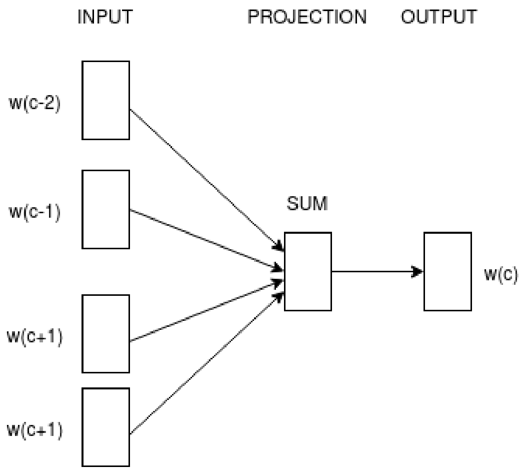


Figure 6.1: Continous Bag of words

6.1.2 Continous Skip-Gram Model

Skip-gram model actually reverse the procedure followed by CBoW to some extent, i.e., here it predicts the surrounding words, given the center word as input to the model, as shown in the figure 6.2 . Therefore, the objective function of the Skip-gram model is to maximize the summed up log probabilties of the context words given the center word w_c which can be seen in the following equation:

$$J_\theta = \frac{1}{C}\sum_{c=1}^{C}\sum_{-k\leq j\leq k,\neq 0} logp(w_{c+j}|w_c) \tag{6.2}$$

Where, the probabilty $p(w_{t+j}|w_t)$ is calculated using the softmax function as shown below:

$$p(w_{c+j}|w_c) = \frac{exp(v_{w_c}^T v'_{w_{c+j}})}{\sum_{w_i\in V} exp(v_{w_c}^T v'_{w_i})} \tag{6.3}$$

where, v_w represents input embeddings and v'_w represents output word embeddings.

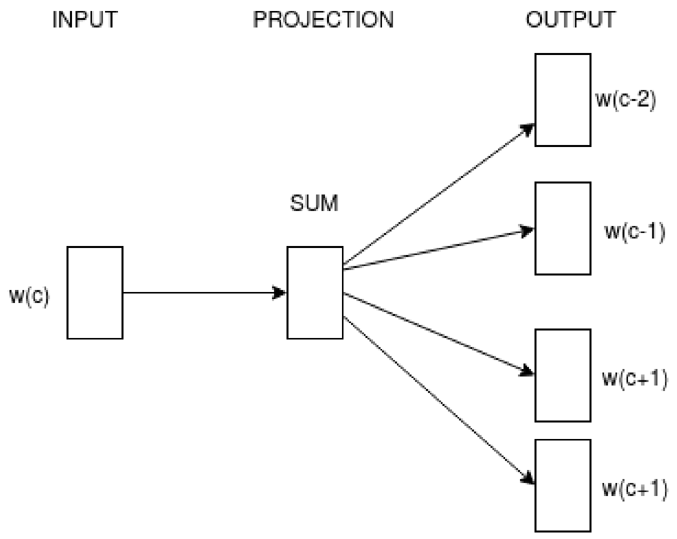


Figure 6.2: Skip Gram Model

6.2 Global Vectors for Word Representation (GloVe)

GloVe [46] uses a co-occurence matrix to calculate embeddings for the words. To know the mechanism of GloVe we first need to define some terms:

w_x= word vector of dimension (d×1)

$\tilde{w}_y$ = word vector of the probe word of dimension (d×1)

K_{xy} = frequency of word w_y occuring in the context of word w_x

In GloVe, to predict co-occurence counts between the words, we need to measure the *similarity of the hidden factors between them.* So, basically we want to create a vector representations that can predict co-occurrence counts from the corpus which is represented as follows:

$$w_x^T \tilde{w}_y + b_x + \tilde{b}_y = log(K_{xy}) \tag{6.4}$$

where, $w_x^T \tilde{w}_y$ represents the similarity of the hidden factors between w_x and w_y to predict the co-occurence between them. $log(K_{xy})$ is the log of actual co-occurence between w_x and w_y. b_x and b_y are the two biases term.
Following the intuition from equation 6.4, we can now define loss function for GloVe as shown below:

$$J = \sum_{x,y=1}^{V} f(K_{xy})(w_x^T \tilde{w}_y + b_x + \tilde{b}_y - log(K_{xy})^2 \tag{6.5}$$

$$f(k) = \begin{cases} (k/k_{max})^\alpha & if\, k < k_{max} \\ 1 & otherwise \end{cases}$$

This loss functions is basically *Mean Squared Error* between the predicted co-occurence counts of the word pairs and the ground truth. But since the predicted co-occurence can not always be accurate, so a function $f(x)$ is introduced to readjust the cost as shwon above. Where, x_{max} is some threshold value and α is a constant.

6.3 FastText

One of the major drawbacks of word2vec and GloVe word embeddings is in their inability to deal with the words which are not present in the vocabulary. In 2017, Facebook Research team proposed a new word embedding technique called *FastText* [9], which is based on continuous skip-gram model. In this technique each word is represented as a a bag of character n-grams, which in turn helps to deal with the words not present in the vocabulary. For example (taken from their work [9]), a word *where* with *n=3* for n-grams will be represented by character n-gramas as : <wh, whe, her, ere, re>and special sequence <where >. Now, the embedding vector for the word *where* will be summation of vector representation of all the n-gram characters.

This way, if a new word comes which is not in the corpus, but the training set has a vector representation for all its n-gram, so the average of all its constituent character n-gram would become the vector representation for this new word. Following the same intuition they obtained the scoring function:

$$s(w, c) = \sum_{n \in S_w} z_n^T v_c \tag{6.6}$$

where S_w represents the set of n-grams present in word w, z_n is the vector representation for each n-gram n and v_c is the context vector representation.

Chapter 7

Proposed Work

7.1 Architecture

We have employed a hybrid architecture by combining various knowledge based and corpus techniques (KACB) with a CNN architecture, as shown in figure 7.1. Each technique present in KAB computes a similarity score between sentences in range 0 to 1 (real values) when a sentence pair is given as an input to them. Also, CNN followed by an FCNN produces a similarity score between two sentences. All the results are then combined to train a regression model(SVR or MLP regressor), which produces the final score.

7.2 Knowledge Based and Corpus Techniques

The knowledge based and corpus based techniques have been taken from [52] (as discussed in section (4.2)) with some modifications in two techniques called *WordNet-Augmented Word Overlap* which is knowledge based technique and *Vector Space Sentence Similarity* which is a corpus based technique. We discuss the drawbacks and changes which are employed to overcome the drawbacks in following sections.

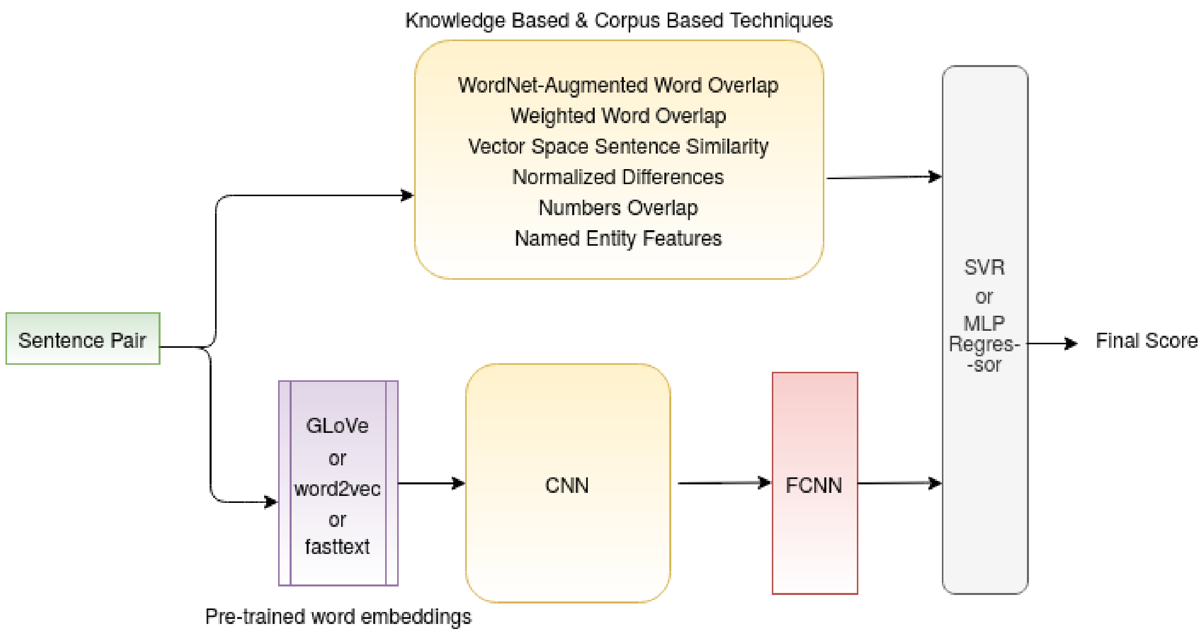

Knowledge Based & Corpus Based Techniques
WordNet-Augmented Word Overlap
Weighted Word Overlap
Vector Space Sentence Similarity
Normalized Differences
Numbers Overlap
Named Entity Features
Sentence Pair
GLoVe or word2vec or fasttext
Pre-trained word embeddings
CNN
FCNN
SVR or MLP Regres- -sor
Final Score

Figure 7.1: Overview of the sytem

7.2.1 WordNet - Augmented Word Overlap (WAWO)

The name of the technique in itself clarifies the knowledge based source used in this technique is the lexical database *WordNet* [43]. As discussed in section 4.2.2, WAWO makes use of *pathlen similarity* to measure the similarity between two concepts present in the wordnet hierarchy. But a drawback with pathlen similarity is, it assigns equal similarity scores to different pairs of concepts without considering the generality or specificity of the pairs in WordNet hierarchy. To understand it better, we can consider an example from the figure 7.2.

Let us consider two pairs of concepts (*nickel.n.02*, *dime.n.01*) and (*medium_of_exchange.n.01*, *scale.n.01*) from the figure 7.2. Both the pairs have a pathlen of 2 (i.e, the count of the edges in the shortest path between them), therefore *pathlen similarity* between them is 0.5 given by the equation 7.1.

$$sim_{pl} = \frac{1}{pathlen(C1, C2)} \tag{7.1}$$

But in actual, if we look intuitively, (*nickel.n.02*, *dime.n.01*) are more similar to each other than (*medium_of_exchange.n.01*, *scale.n.01*), also (*nickel.n.02*, *dime.n.01*) are

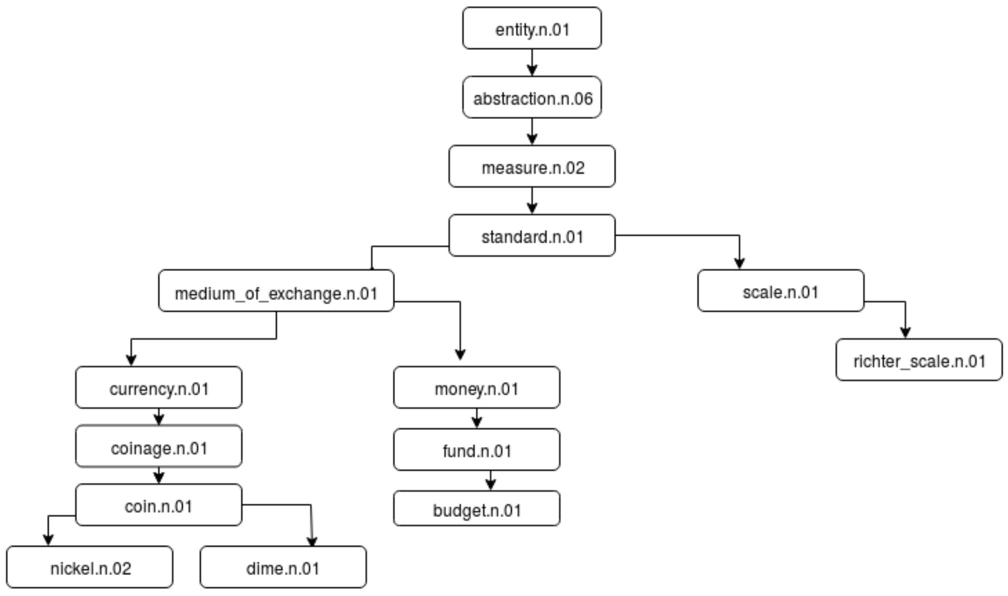


Figure 7.2: WordNet Hierarchy Example

more specific in hierarchy than (*medium_of_exchange.n.01*, *scale.n.01*), therefore the similarity score for (*nickel.n.02*, *dime.n.01*) should have been greater. So, we can observe that the similarities given by *pathlen similarity* are not accurate. To overcome these drawbacks we have used *Jiang-Conrath Similarity (JC similarity)* and *Lin Similairty* proposed by [24] and [35] respectively, replacing *pathlen similarity*, which we discuss in the next sections.

1) Jiang-Conrath Similairty (JC Similairty)

Jiang- Conrath similarity(JC Similarity) [24] makes use of lengths of graph edges to overcome the drawbacks discussed above and is expressed as:

$$sim_{JC}(c_1, c_2) = \frac{1}{IC(c_1) + IC(c_2) - 2 * IC(LCS(c_1, c_2))} \tag{7.2}$$

where $LCS(c_1, c_2)$ is the least common subsumer between the concepts c_1 and c_2, $IC(c)$ is the *Information Content* of the concept c. In order to calculate *IC(c)* we have used the same formula as given by *Resnik(1998)* in [50], which is expressed as follows:

$$IC(c) = -logP(c) \tag{7.3}$$

we have used *Brown Corpus*, *BNC Corpus* and *treebank Corpus* from NLTK to calculate IC of each concept in WordNet. $P(c)$ is the probability that any random word *say x* from the corpus is a *hyponym/instance or a part* of concept c in WordNet hierarchy. This random word x ranges over whole corpus. Therefore, we can say that P(root) = 1 because any word x from the corpus will be an instance of root concept. This probability is given as:

$$P(c) = \frac{\sum_{w \in S_w(c)} count(w)}{T} \tag{7.4}$$

where S_w represents the set of words that are an instance of concept c, and T is the total number of words which are present in corpus as well as in the thesaurus.

JC Similairty Derivation

Information content (IC) can be used to find lengths of the graph edges in WordNet hierarchy structure, which is given as:

$$dist_{JC}(c, hypernym(c)) = IC(c) - IC(hypernym(c))$$

The formula to compute distance between two concepts formula is expressed as:

$$dist_{JC}(c_1, c_2) = dist_{JC}(c_1, LCS(c_1, c_2)) + dist_{JC}(c_2, LCS(c_1, c_2))$$

$$= IC(c_1) - IC(LCS(c_1, c_2))IC(c_1) - IC(LCS(c_1, c_2))$$

$$= IC(c_1) + IC(c_1) - 2 * IC(LCS(c_1, c_2))$$

Therefore, JC-similarity comes out to be the same as given in 7.2), expressed as follows:

$$sim_{JC}(c_1, c_2) = \frac{1}{IC(c_1) + IC(c_2) - 2 * IC(LCS(c_1, c_2))}$$

2) Lin- Similarity

Lin - similarity [35] is the information content common to c1 and c2, normalized by their average information content. The similarity is given as follows:

$$sim_{Lin}(c_1, c_2) = \frac{2.IC(LCS(c_1, c_2))}{IC(c_1) + IC(c_2)} \tag{7.5}$$

It basically tells the more information they don't share , the less similar they are.

7.2.2 Vector Space Sentence Similarity

This feature as we discussed in section 4.2.4 makes use of *LSA vectors* to represent each word in a sentence. Now, in this work, rather than using *LSA vectors*, vectors provided by Google's pre-trained *word2vec model* [42] has been used. This word2vec model makes use of *skip-gram architecture* and is trained on various news dataset, which contributed approximately *100 billion* word . Also, same modification has been done with the weighted word vectors, where word vectors are weighted by their Information Content. These modifications have been done because *word2vec algorithm* has shown to capture better similarity between the words from a given corpus ([40]). But it gives better performance when the training data is in large amount, that is the reason a pre-trained model is preferred here.

7.2.3 Experiments

We have done five different types of experiments in our work on *knowledge based and corpus techniques*, the abbreviations and interpretation of them are as follows:

jc: It contains results for the changes made in *WordNet-Augmented Word Overlap*, where *pathlen similarity* is replaced by *JC similarity* keeping rest of the techniques same.

lin: It contains results for the changes made in *WordNet-Augmented Word Overlap*, where *pathlen similarity* is replaced by *Lin similarity* keeping rest of the techniques same.

jc_lin: It contains results for the changes made in *WordNet-Augmented Word Overlap*, where *pathlen similarity* is replaced by *Lin similarity* in one feature and by *JC*

similarity in another, keeping rest of features same.
ww_wwv: It contains results for the changes made in *Vector Space Sentence Similarity*, where LSA vectors are replaced by Word2Vec vectors and also an additional feature is used where LSA vectors is replaced by weighted Word2Vec vectors as discussed in Section(4.2.4), rest all the features ae the same.
jc_lin_wv_wwv : In this technique all the proposed changes are merged together keeping rest of the features same.

These experiments were carried on two different sets of dataset. First, we see how the five experiments performed on the dataset provided by SemEval in the year 2012 [4] to compare our results with [52]. Secondly, we see how our experiments performed on the latest monolingual dataset on English provided by SemEval in 2017 [13], to compare it with baseline models.

Results on 2012 SemEval Dataset

The dataset is taken from the SemEval STS task (2012) [4] which contains in total 5342 sentence pairs in English language and a corresponding score in range 0 to 5 (all real values). This data is collected from various different sources as shown in the 7.3.

Dataset	Pairs	Source
MSRpar	1500	News
MSRvid	1500	Video
SMTeuroparl	750	MT eval
SMTnews	399	MT eval
OnWn	1193	Glosses

Table 7.1: Dataset Description

The MSRvid and MSRpar dataset is collected by Microsoft Research(MSR). SMTeuroparl dataset was created from ACL Workshop based on Machine Translation in 2007 and 2008 . [11] [12]. In addition, two surprise datasets SMTnews and OnWn were also released which are used here.
The results are computed by finding the *Pearson Correlation* 5.7.1 between the

Technique	MSRvid	MSRpar	SMTeuroparl	SMTnews	OnWn
(saric-etal) [52]	0.8803	0.7343	0.4771	0.3989	0.6797
jc	0.8747	**0.7381**	0.5364	0.4237	0.6914
lin	0.8705	0.7371	0.5362	0.4319	0.6904
jc_lin	0.8713	0.7377	0.5354	0.4068	0.6899
wv_wwv	**0.8837**	0.7378	**0.5376**	**0.5014**	**0.7067**
jc_lin_wv_wwv	0.8791	0.7367	0.5323	0.4720	0.7027

Table 7.2: Pearson Correlation Values for each dataset (from 2012) corresponding to each experiments .

scores given by the model on the test dataset and the gold-standard human annoted scores.
In table 7.2, we have compared our experiments with [52]. We have observed that except for *MSRpar dataset*, *wv_wwv* has performed better for all the datasets. For *MSRpar dataset jc* gave slightly better result.

Results on Monolingual SemEval Dataset(2017)

In table 7.3 we have compared our results with averaged word embedding baseline [51] on monolingual(English) test dataset [13]. We notice that all our results are above the baseline scores.

Experiments	Scores
word_vec_baseline [51]	55.8
jc	73.0
lin	72.4
jc_lin	72.7
wv_wwv	71.4
jc_lin_wv_wwv	72.7

Table 7.3: Pearson Correlation ×100 Scores On Test Dataset 2017

In table 7.4 we have shown benchmark results of STS 2017 [13] on test dataset (with the techniques which they have adopted) and compared with our top performing experiment. We notice that except [14] any of our results are not above rest of the benchmark results.

Experiments	Scores
jc	73.0
Ensemble [58]	81.0
Wordnet + Embeddinglin [62]	80.9
Ensemble [36]	79.2
CNN [55]	78.4
Doc2Vec [14]	59.2

Table 7.4: STS 2017 benchmark

7.3 CNNs And FCNN

The CNN layer which is then followed by an FCNN to calculate the scores for similarity between a pair of sentences is shown in the figure 7.3. The architecture has been taken from [53]. Each sentence is first represented in a matrix form, by passing it into word embedding layer. Then each sentence matrix is given as an input to the convolution layer and we get two vectors v_1 and v_2, after the *max-pooling operation*, for both the sentences respectively. At last, by the concatenation of element-wise absolute difference and the element-wise multiplication between the two representations of input sentences v_1 and v_2, a *Semantic Difference Vector*($S\vec{D}V$) is created. This $S\vec{D}V$ is finally passed through the Full-Connected Layer (FCNN) to get the final output.

During training, we considered the task of semantic relatedness as a classification problem instead of a regression problem, by considering the outputs from the last layer as probability distribution over 6 classes. And the gold labels (referred as a in 7.6) which were real values in range *0-5* were converted into a probability distribution as follows [57]:

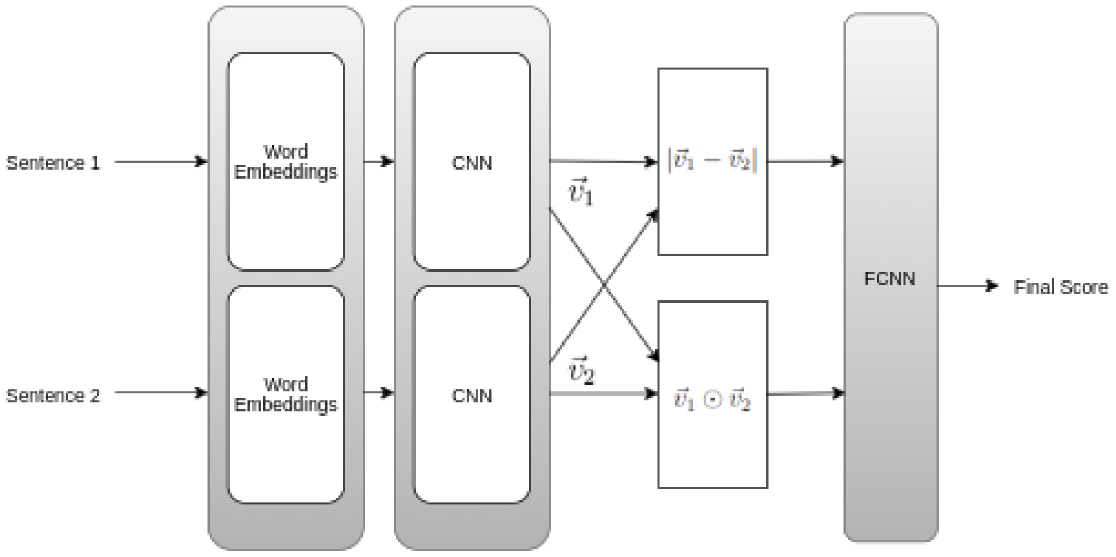


Figure 7.3: CNN Architecture [53]. Note that there is only one CNN in the shaded region onto which both sentences are passed.

$$p_t = \begin{cases} a - \lfloor a \rfloor, & i = \lfloor a \rfloor + 1 \\ \lfloor a \rfloor - a + 1, & i = \lfloor a \rfloor \\ 0 & otherwise \end{cases} \tag{7.6}$$

And then used the loss functions (section 5.7) to compute the errors, which were then backpropogated for training the weights. **During evalution** a weighted sum or expectation of the scores was calculated, which was considered as the final score.

Reason for prefering CNNs over RNNS or LSTMs

The model proposed by [53] (which we have used) is an extension of Siamese LSTM [44] with two major changes. First, rather than using an LSTM, a CNN is used in their work. Second, in [44] an LSTM model is employed for each input sentence and weights are shared between layers and updated parallelly during learning phase,whereas, in [53] two sentences are given as an input into the same CNN model and then the weights are shared. The reason for the first, i.e, using CNNs for training our model is that though RNNs or LSTMs works better on dynamic length of inputs sentences but if the input length is fixed, CNNs generally performs better. And in the STS 2017 dataset(for English sentence pairs) which we have considered in our work, the lenghts of the sentences among each other does not vary much, i.e., the standard deviation of the lengths of sentences is less so CNNs will work better here. To compare both LSTMs and CNNs on this dataset, we can compare [53] work with [8]. The whole architecture is exacly the same except that, [53] used CNNs in their work whereas [8] used LSTMs. And we can clearly conclude from their work that CNNs are working much better than LSTMS, where CNNs and LSTMs got a *Pearson Correlation* of $\mathbf{0.78}$ and $\mathbf{0.47}$ on English test dataset respectively. That is the reason we have also used CNNs here.

7.3.1 Experiments

[53] have used 300 dimensional GloVe word embeddings for the representation of each word in a sentence and Pearson Correlation Coefficient as the loss function. We have tried five different modifications in their work by using different loss functions (section 5.7) and pre-trained word embeddings. The abbreviations and the interpretation of them are as follows:
glove_kl: In this experiment we have replaced loss function correlation coefficient with KL-Divergence loss function.
word2vec_coef: In this experiment we have replaced pretrained GloVe word embeddings with pretrained word2vec embeddings.
word2vec_kl: Here both embeddings and loss functions have been replaced with *word2vec* and *KL-Divergence* respectively.
fasttext_kl: Here also, both embeddings and loss functions have been replaced with *word2vec* and *KL-Divergence* respectively.
fasttext_coef: In this experiment embeddings are replaced with fasttext embeddings, keeping the loss function same.

Each experiment discussed above produces a score between the two sentences, expressing the semantic relatedness between them. These scores, corresponding to every English sentence pair from the SemEval 2017 test dataset [13] are compared with Gold Standard scores using person correlation coefficient5.7.1, which are shown in table 7.5.

Experiments	Scores
(shao) [53]	78.4
glove_kl	70.26
word2vec_coef	76.9
word2vec_kl	70.10
fasttext_kl	72.5
fasttext_coef	73.4

Table 7.5: (Pearson Correlation Scores $\times 100$) On Test Dataset(2017) on English Pairs

We observe that none of our sytems were able to beat the scores of [53], which proves that GloVe word embeddings with correlation coefficient as loss function was certainly a good combination for this architecture.

7.4 Whole architecture

The results in this section are computed by finding *Pearson Correlation Coefficient* between the output scores from our proposed architecture (figure 7.1) and the gold standard labels. The table 7.6 shows the results when the combined outputs of *CNN*7.3.1 and *Knowledge Based and Corpus Based (KACB)*7.2.3 is passed into an MLP regressor and a *Pearson Correlation Coefficient* is computed between output of the scores from MLP regressor and the gold standard labels.

MLP Regressor	jc	lin	jc_lin	wv_wwv	jc_lin_wv_wwv	[52]
(glove_coef) [52]	**0.787**	0.742	0.731	0.742	0.752	0.786
glove_kl	0.761	0.752	0.732	0.733	0.745	0.734
word2vec_coef	0.731	0.725	0.713	0.713	0.716	0.715
word2vec_kl	0.721	0.724	0.712	0.715	0.70	0.730
fasttext_kl	0.732	0.721	0.716	0.712	0.708	0.730
fasttext_coef	0.736	0.727	0.713	0.712	0.715	0.732

Table 7.6: Pearson Correlation Values b/w MLP regeressor Scores and Gold labels

SVR	jc	lin	jc_lin	wv_wwv	jc_lin_wv_wwv	[52]
(shao-2017) [53]	0.771	0.762	0.761	0.752	0.753	0.770
glove_kl	0.731	0.721	0.745	0.752	0.742	0.751
word2vec_coef	0.733	0.723	0.751	0.743	0.742	0.751
word2vec_kl	0.761	0.752	0.761	0.752	0.754	0.753
fasttext_kl	0.751	0.752	0.730	0.731	0.732	0.752
fasttext_coef	0.763	0.752	0.752	0.741	0.731	0.762

Table 7.7: Pearson Correlation Values b/w SVR Scores and Gold labels

The table 7.7 shows the results when the combined outputs of *CNN* and *Knowledge Based and Corpus Based (KACB)* is passed into an MLP regressor and a *Pearson Correlation Coefficient* is computed between output of the scores from SVR and the gold standard labels.
We have observed from table 7.6 that the best performance is given when CNN architecture as proposed by [53] is combined with our proposed KACB technique (*jc*). In table 7.8 we compared our top performing architecture with the benchmark results as per SemEval 2017 test dataset on English sentence pairs [13] and noticed that we are getting slightly better results than [53].

Experiments	Scores
Ensemble [58]	81.0
Wordnet + Embeddinglin [62]	80.9
Ensemble [36]	79.2
glove_coef & *jc*	**78.7**
CNN [55]	78.4
Doc2Vec [14]	59.2

Table 7.8: STS 2017 benchmark.The bold score is our score on the benchmark database.

Chapter 8

Results and Discussion

With some modifications in *Knowledge and Corpus Based* techniques (as seen in section 7.2) we were able to improve scores for measuring semantic similarity between sentences (as per STS 2012 test dataset for English pairs [4]), as seen in table 7.2. But for the 2017 STS dataset [13], except [14] our scores were not able to beat other benchmark results (table 7.4).
Some modifications in CNN architecture given by [53] (as seen in section 7.3.1), like changing the word embeddings, loss functions and hidden layers or unit sizes did not really bring any improvement in measuring similarity between sentences (table 7.5). Which therefore proved that the combination of word embeddings and the loss function as suggested in [53] (i.e. *pretrained GLoVe vectors* as word embedding [47] and *Pearson Correlation Coefficient* as loss function 5.7.1) was actually the best among all the experiments we carried out (table 7.5).
Our final hybrid architecture (figure 7.1) was able to perform much better than [14] and slightly better than [53] which are part of the benchmark results 2017 (table 7.8). So, this shows that by combining the traditional *Knowledge and Corpus Based* techniques with the recent techniques with DNNs, we can extract more relevant feature set out of the sentence pairs, to compute the similarity. Therefore, in the future work, we will include some more traditional techniques, like using word alignment [55] to improve the capability of our system to measure similarity between the two sentences.

Bibliography

[1] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, et al. Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*, pages 252–263, 2015.

[2] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. Semeval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 81–91, 2014.

[3] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. * sem 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, 2013.

[4] Eneko Agirre, Mona Diab, Daniel Cer, and Aitor Gonzalez-Agirre. Semeval-2012 task 6: A pilot on semantic textual similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, SemEval '12, pages 385–393, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[5] James Allen. *Natural Language Understanding.* Benjamin-Cummings Publishing Co., Inc., USA, 1988.

[6] J. Atkinson-Abutridy, C. Mellish, and S. Aitken. Combining information extraction with genetic algorithms for text mining. *IEEE Intelligent Systems*, 19(3):22–30, 2004.

[7] Daniel Bär, Chris Biemann, Iryna Gurevych, and Torsten Zesch. Ukp: Computing semantic textual similarity by combining multiple content similarity measures. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, SemEval '12, pages 435–440, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[8] Joe Barrow and Denis Peskov. UMDeep at SemEval-2017 task 1: End-to-end shared weight LSTM model for semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 180–184, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[9] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[10] Tomáš Brychcín and Lukáš Svoboda. Uwb at semeval-2016 task 1: Semantic textual similarity using lexical, syntactic, and semantic information. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 588–594, 2016.

[11] Chris Callison-Burch, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. (meta-) evaluation of machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 136–158. Association for Computational Linguistics, 2007.

[12] Chris Callison-Burch, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. Further meta-evaluation of machine translation. In *Proceedings of the third workshop on statistical machine translation*, pages 70–106. Association for Computational Linguistics, 2008.

[13] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual

focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[14] Mirela-Stefania Duma and Wolfgang Menzel. SEF@UHH at SemEval-2017 task 1: Unsupervised knowledge-free semantic textual similarity via paragraph vector. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 170–174, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[15] C. Fellbaum and G. Miller. *Combining Local Context and Wordnet Similarity for Word Sense Identification*, pages 265–283. 1998.

[16] Peter W. Foltz, Walter Kintsch, and Thomas K Landauer. The measurement of textual coherence with latent semantic analysis. *Discourse Processes*, 25(2-3):285–307, 1998.

[17] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[18] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. PPDB: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[19] Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, and Li Deng. Modeling interestingness with deep neural networks. Technical Report MSR-TR-2014-56, October 2014.

[20] Vasileios Hatzivassiloglou, Judith L. Klavans, and Eleazar Eskin. Detecting text similarity over short passages: Exploring linguistic feature combinations via machine learning. In *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[21] Hua He, Kevin Gimpel, and Jimmy Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1576–1586, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[22] Hua He and Jimmy Lin. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 937–948, San Diego, California, June 2016. Association for Computational Linguistics.

[23] Hua He, John Wieting, Kevin Gimpel, Jinfeng Rao, and Jimmy Lin. UMD-TTIC-UW at SemEval-2016 task 1: Attention-based multi-perspective convolutional neural networks for textual similarity measurement. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1103–1108, San Diego, California, June 2016. Association for Computational Linguistics.

[24] Jay J. Jiang and David W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the 10th Research on Computational Linguistics International Conference*, pages 19–33, Taipei, Taiwan, August 1997. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP).

[25] Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 103–112, Denver, Colorado, May–June 2015. Association for Computational Linguistics.

[26] Rie Johnson and Tong Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 919–927. Curran Associates, Inc., 2015.

[27] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[28] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[29] Thomas K Landauer, Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284, 1998.

[30] Thomas K. Landauer, Darrell Laham, Bob Rehder, and M. E. Schreiner. How well can passage meaning be derived without using word order? a comparison of latent semantic analysis and humans. 1997.

[31] Michael D Lee and Daniel J Navarro. Extending the alcove model of category learning to featural stimulus domains. *Psychonomic Bulletin & Review*, 9(1):43–58, 2002.

[32] Michael D. Lee, Brandon Pincombe, and Matthew Welsh. An empirical evaluation of models of text document similarity. 2005.

[33] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th Annual International Conference on Systems Documentation*, SIGDOC '86, page 24–26, New York, NY, USA, 1986. Association for Computing Machinery.

[34] Y. Li, D. McLean, Z. A. Bandar, J. D. O'Shea, and K. Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1138–1150, 2006.

[35] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 296–304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[36] Nabin Maharjan, Rajendra Banjade, Dipesh Gautam, Lasang J. Tamang, and Vasile Rus. DT_Team at SemEval-2017 task 1: Semantic similarity using alignments, sentence-level embeddings and Gaussian mixture model output. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 120–124, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[37] Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA).

[38] Charles T Meadow, Donald H Kraft, and Bert R Boyce. *Text information retrieval systems*. Academic Press, Inc., 1999.

[39] Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, page 775–780. AAAI Press, 2006.

[40] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[41] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[42] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[43] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[44] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2786–2792. AAAI Press, 2016.

[45] Daniel J Navarro and Michael D Lee. Common and distinctive features in stimulus similarity: A modified version of the contrast model. *Psychonomic Bulletin & Review*, 11(6):961–974, 2004.

[46] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[47] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[48] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[49] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, page 448–453, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[50] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 448–453, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[51] Alexandre Salle, Aline Villavicencio, and Marco Idiart. Matrix factorization using window sampling and negative sampling for improved word representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational*

Linguistics (Volume 2: Short Papers), pages 419–424, Berlin, Germany, August 2016. Association for Computational Linguistics.

[52] Frane Šarić, Goran Glavaš, Mladen Karan, Jan Šnajder, and Bojana Dalbelo Bašić. TakeLab: Systems for measuring semantic text similarity. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – (SemEval 2012)*, pages 441–448, Montréal, Canada, 7-8 June 2012. Association for Computational Linguistics.

[53] Yang Shao. HCTI at SemEval-2017 task 1: Use convolutional neural network to evaluate semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 130–133, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[54] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, page 101–110, New York, NY, USA, 2014. Association for Computing Machinery.

[55] Md Arafat Sultan, Steven Bethard, and Tamara Sumner. Back to basics for monolingual alignment: Exploiting word similarity and contextual evidence. *Transactions of the Association for Computational Linguistics*, 2:219–230, 2014.

[56] Md Arafat Sultan, Steven Bethard, and Tamara Sumner. DLS@CU: Sentence similarity from word alignment and semantic vector composition. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 148–153, Denver, Colorado, June 2015. Association for Computational Linguistics.

[57] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July 2015. Association for Computational Linguistics.

[58] Junfeng Tian, Zhiheng Zhou, Man Lan, and Yuanbin Wu. ECNU at SemEval-2017 task 1: Leverage kernel-based traditional NLP features and neural networks to build a universal model for multilingual and cross-lingual semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 191–197, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[59] Peter D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In Luc De Raedt and Peter Flach, editors, *Machine Learning: ECML 2001*, pages 491–502, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[60] Amos Tversky. Features of similarity. *Psychological review*, 84(4):327, 1977.

[61] Peng Wang, Jiaming Xu, Bo Xu, Chenglin Liu, Heng Zhang, Fangyuan Wang, and Hongwei Hao. Semantic clustering and convolutional neural network for short text categorization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 352–357, Beijing, China, July 2015. Association for Computational Linguistics.

[62] Hao Wu, Heyan Huang, Ping Jian, Yuhang Guo, and Chao Su. BIT at SemEval-2017 task 1: Using semantic information space to evaluate semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 77–84, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[63] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics*, ACL '94, page 133–138, USA, 1994. Association for Computational Linguistics.

[64] Wei Xu, Chris Callison-Burch, and Bill Dolan. SemEval-2015 task 1: Paraphrase and semantic similarity in twitter (PIT). In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 1–11, Denver, Colorado, June 2015. Association for Computational Linguistics.

[65] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

Publications

[1] "Combining Knowledge And Corpus Based Techniques with Convolutional Neural Nets for Semantic Textual Similarity." Shivam Soni, Kavi Narayana Murthy.

Semantic Textual Similarity

by Shivam Soni

Submission date: 30-Jun-2020 05:21PM (UTC+0530)

Submission ID: 1351761162

File name: shivam.pdf (666.08K)

Word count: 11918

Character count: 57510

Semantic Textual Similarity

ORIGINALITY REPORT

4%	1%	2%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Middle East Technical University Student Paper	<1%
2	Rezgui, Kalthoum, Hédia Mhiri, and Khaled Ghédira. "Theoretical Formulas of Semantic Measure: A Survey", Journal of Emerging Technologies in Web Intelligence, 2013. Publication	<1%
3	P. Selvi, N.P. Gopalan. "Notice of Violation of IEEE Publication Principles - Sentence Similarity Computation Based on Wordnet and Corpus Statistics", International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007), 2007 Publication	<1%
4	www.wildml.com Internet Source	<1%
5	Pedersen, T.. "Measures of semantic similarity and relatedness in the biomedical domain", Journal of Biomedical Informatics, 200706 Publication	<1%