Wirtschaftsinformatik
und Maschinelles Lernen
Stiftung Universität Hildesheim
Marienburger Platz 22
31141 Hildesheim

Natural Language Processing Methods for Dealing with Words

Term Paper

Gökce Sucu
246112

August 2023

Contents

List of Figures

1 Introduction

Word embeddings in NLP are used to provide a meaningful vector representation of each word in a corpus to understand similarities between them and have promising results on representing semantic relations in analogical reasoning tasks. Also, it has a key role on neural-network based approaches for natural language processing tasks like part-of-speech tagging, chunking, named entity recognition, semantic role labeling etc. [1]. However, word embeddings might not always provide sufficient semantic representations when they are dealing with morphologically rich languages like Turkish, Hungarian etc, that have high variance in word forms and high context variability. That's because, word embedding algorithm is based on fixed-size vocabularies without additional contextual information and it causes less accurate estimates. Nowadays, so many state-of-the-art models have been proposed and improved to solve this problem and one of these is subword embedding algorithm. Basically, this method utilizes the principles of morphology to improve the quality of representations of uncommon words by splitting the words into smaller pieces.

In this paper, four different models will be proposed and examined in order to see if the subword embedding method has ability to improve the performance of basic word embedding framework by addressing its drawback that is mentioned in the previous paragraph. Mainly, three of these models will be based on subword embedding approach (based on prefix-suffix, or compound, or mixed), and those three models are only distinguished due to having different word splitting technique. One of these models will be basic word embedding algorithm and this method will be used for accuracy score comparison. To able to do this, an accuracy function will be defined. The accuracy score of each model on two different word similarity dataset will be calculated and compared by using 3 different training dataset with different size (1M,300K,100K sentences).

2 Related Works

First word embedding method that utilized subwording techniques was introduced by Schütze [2]. In this study, a word was represented as the sum of 4-gram vectors obtained by SVD. After that, several algorithms that use morphological information have been proposed [3, 4, 5, 7]). These algorithms utilize the morphological decomposition of words in the text. On the other hand, some frameworks that use subword information and do not depend on morphological decomposition has been proposed [8, 9]. Bojanowski et al. (2017) introduced fastText framework which is based on character n-gram embedding and can be trained on large text corpus [10].

3 The Models

Mainly, three models will be based on subword embedding approach (based on prefix-suffix, or compound, or mixed), and those three models are only distinguished because of having different word splitting techniques. Model 1, the one with basic word embedding, uses complete words as token. However, Model 2, 3 and 4 use different kind of splitting approach (or subwording techniques) in order to split these whole words into pieces and create new morphologically meaningful tokens as input.

- **Model 1**, will be simple word embedding algorithm where tokens are whole words without splitting. This method will be used as a baseline .
- **Model 2** is a subword embedding framework and its splitting method is created manually by using some frequently used prefixes and suffixes in English. Splitting function or subwording method only consider the words that have more than 3 character. For this reason, it is expected that Model 2 is useful for exploiting the words that have more than 3 characters and have either suffix or prefix (see Figure 1).
- **Model 3** is another type of subword embedding method where splitting method aims to benefit from the compound words that has more than 4 char-

```
def splitting(sentence):
    liste = []
    for word in sentence:
        n = len(word)

        if n>3:
            #SUFFIXES
            if word[n-2:n]=="ed":
                a = word[:-2]
                liste.append(word)
                liste.append(a)
                #liste.append("ed")


            elif word[n-3:n]=="ing":
                liste.append(word)
                a = word[:-3]
                liste.append(a)
                #liste.append("ing")


            elif word[-1] =="s":
                liste.append(word)
                a= word[:-1]
                liste.append(a)
                #liste.append("s")


            elif word[n-4:n]=="able":
                liste.append(word)
                a = word[:-4]
                liste.append(a)
                #liste.append("able")

                liste.append(a)
                #liste.append("less")

          #PREFIXESÍ
            elif word[0:5]=="under":
                liste.append(word)
                a = word[5:n]
                liste.append(a)
                liste.append("under")

            elif word[0:5]=="super":
                liste.append(word)
                a = word[5:n]
                liste.append(a)
                liste.append("super")

            else:
                liste.append(word)

    return liste
```

Figure 1: Splitting Function of Model 2

If a raw token has some most frequent prefixes or suffixes, then algorithm seperates them into two pieces and add the whole word and the word without prefix or suffix.

acters. The approach of Model 4 can also utilize some suffix information but not prefix (see Figure 2).

- **Model 4** is the mixture of Model 2 and Model 3. Its splitting function aims to separate compound names and some prefixes (see Figure 3).

4 Data Foundation

English (2018) dataset from The Leipzig Corpora Collection and its 3 different size (1M, 300K, 100K sentences) were used as training set and the accuracy score

```
def splitting_two(sentence):
    liste=[]
    for word in sentence:
        n = len(word)
        #chossing the words with more than 4 character
        if n>4:
            for i in range(4,n):
                if word[0:i] in vocabulary:
                    liste.append(word[0:i])
                    if word[i:n] in trained_words_model_one:
                        liste.append(word[i:n])

        liste.append(word)
    return liste
```

Figure 2: Splitting Function of Model 3

In the Model 3, the i index stops in the raw token, while the meaningful word piece catched. Notice that the catched word piece must be existed in vocabulary. Then, this word is added to the token list. After that, the algorithm also looks the rest of word piece. If it is also meaningful (existedin vocabulary), this rest of word piece is also added to token list. By using this method, compounds can be seperated.
Example (cheeseburger) chees(i=0), ***cheese(i=1)****,meaningful, stop, add into token set, look the rest of it,* ***burger****, meaningful, add into token set*

is calculated for each model and for each dataset according to selected word similarity dataset. Two different word similarity dataset were used: wordSim353 and SynEngl.

5 Training

All models were trained with Word2Vec function from gensim library. By using Word2Vec function in Python, for each tokens, the vector representation is created. The only difference between the training process of the models was the tokenization method. For each model, different set of tokens is created by using selected techniques. After the tokenization, tokens were used as input for training.

6 Evaluation

To able to evaluate the training results, the accuracy score function is defined. First of all, in two word similarity dataset, values are transformed to between -1 and 1 to able to compare with cosinus similarity values in gensim (values in cosinus similarity function are between 1 to -1).

```
def splitting_mix(sentence):
    liste=[]
    for word in sentence:
        n = len(word)
        #chossing the words with more than 4 character
        if n>4:
            for i in range(4,n):
                if word[0:i] in vocabulary:
                    liste.append(word[0:i])
                    if word[i:n] in trained_words_model_one:
                        liste.append(word[i:n])
        elif <5:
            if word[-1] =="s":
                a= word[:-1]
                liste.append(a)
            elif word[n-2:n]=="es":
                a = word[:-2]
                liste.append(a)
        liste.append(word)


    return liste
```

Figure 3: Splitting Function of Model 4

Accuracy Score Function: It is the mean value of the difference between the all existed word pairs' (in trained dataset and word similarity dataset at the same time) cosinus similarity value in the selected model and value in the selected word similarity dataset (see Figure 4).

WordSim353	100K Dataset	300K Dataset	1M Dataset
Model 1	0.432	0.541	0.580
Model 2	0.473	0.576	0.591
Model 3	0.563	0.597	0.613
Model 4	0.573	0.607	0.626

Table 1: Accuracy Scores of Each Models on WordSim353 Dataset

SynEngl	100K Dataset	300K Dataset	1M Dataset
Model 1	0.581	0.519	0.519
Model 2	0.569	0.524	0.511
Model 3	0.501	0.523	0.511
Model 4	0.490	0.524	0.510

Table 2: Accuracy Scores of Each Models on SynEngl Dataset

After accuracy score function is defined, all models were trained on different size datasets (1M, 300K, 100K sentences). Finally, for each training, accuracy scores were calculated (see Table 1, Table 2).

```
# S: Word Similarity Dataset
# W: All traiend words
# M: Model

def accuracy(S,W,M):
    all_d = []
    for i in S:
        if i[0] in W and i[1] in W:
            model_similarity= M.wv.similarity(i[0], i[1])
            d = abs(abs(model_similarity) -i[2])
            all_d.append(d)

    print("Number Of Words That Accuracy Calculated = ",len(all_d),"\nAccuracy = ", 1-me
```

Figure 4: Accuracy Score Function

7 Conclusion

According to results of the first experiment (see Table 1), it can be said that morphologically splitting the words into smaller pieces before training improves the quality of vector representation of a word in a corpus on WordSim353. For example, after splitting the word "cats" as "cat" and "s", the algorithm can easily understand that these two words are the same. On the other hand, it can be obviously seen in the result of second experiment (see Table 2) that subwording did not effect the accuracy score and even might decrease in the SynEngl. That's because, in SynEngl dataset, words are not chosen morphologically similar but the words are similar in respect to their meaning. Splitting the words and utilizing the subwording algroithm causes the decrease of ability to learning the meaning of a word in the corpus, while it increases ability to understand structural similarity.

References

[1] Salle A., Villavicencio A. "Incorporating Subword Information into Matrix Factorization Word Embeddings". (2018)

[2] Schutze H. "Word space: In Advances in neural information processing systems". (1993)

[3] Alexandrescu A, Kirchhoff K. "Factored neural language models". (2009)

[4] Luong R., and Manning C. "Better word representations with recursive neural networks for morphology". (2013)

[5] Qiu S., Cui Q., Bian J., Gao B., and Liu T. "Colearning of word representations and morpheme representations". (2014)

[6] Botha J. A., Blunsom P. "Compositional morphology for word representations and language modelling". (2014)

[7] Cotterell R., Schutze H. "Morphological word-embeddings".(2015)

[8] Santos C. D., Zadrozny B. "Learning character-level representations for part-of-speech tagging". (2014)

[9] Wang L, Dyer C, Black A. W., Trancoso A, Fermandez R., Amir S., Marujo L., and Luis T. "Finding function in form: Compositional character models for open vocabulary word representation". (2015)

[10] Bojanowski P., Grave E, Joulin A., Mikolov T. "Enriching word vectors with subword information". (2017)