Compare word2vec with hash2vec for Word Sense Disambiguation

A Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements of the Class

CS 297

By

Neha Gaikwad

Dec 2019

Table of Contents

I. INTRODUCTION

There are millions of documents generated every week on the internet. It is highly impossible to manage them manually and perform complex tasks such as classification, regression, etc. on the data extracted from these documents. Neural networks come in picture here, for processing a huge amount of data and applying different techniques to it in order to get clusters, classifiers and so on. It helps in getting vectors from words and applying them to different problems to get the solution. It also helps in getting machines to understand if the entered keyword ‘apple’ is the company and not a tasty fruit. The answer to the above questions lie in creating a representation for words that capture their meanings, semantic relationships and the different types of contexts they are used in. And all of these are implemented by using Word Embeddings or numerical representations of texts so that computers may handle them.

Our aim for CS297-298 is to build an entity disambiguation system. The Wikipedia data set has been used as data set in many research projects.In our project, we use the English Wikipedia dataset as a source of word sense, and word embedding to determine the sense of word within the given context. Word embeddings were originally introduced by Bengio, et al, in 2000 [2]. A Word embedding is a parameterized function mapping words in some language to high- Word Sense Determination from Wikipedia Data Using a Neural Net 4 dimensional vectors. Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, and explicit representation in terms of the context in which words appear. Using a neural network to learn word embedding is one of the most

exciting areas of research in deep learning now [3]. Unlike previous work, in our project, we will use neural network to learn word embeddings.

The demonstration of hash to vector method will be carried out where hash is generated for every word and it gets updated as the window slides over the words. It then stores hash for every word in dictionary form and get the final hash to vector results. We would also like to compare the results using hash2vec.

The following are the deliverables I have done in this semester to understand the essence of neural network, word embedding and the work flow of TensorFlow. In Deliverable 1, we presented the introduction to word embedding using word to vec and developed a simple program to convert words into vectors (word2vec) using TensorFlow and genism. In Deliverable 2, we have applied word2vec approach for a simple application of calculating the cosine similarity of words. In Deliverable 3, we presented the introduction to word embedding using hashing features and discussed the algorithm. In Deliverable 4, we have applied hash2vec to find the nearest word for the given word. This is also a simple application of hash2vec. In Deliverable 5, we have demonstrated simple hashing trick which can be used for documentation classification which shows the power of hashing to distinguish between different words in the area where large data is available. More details of these deliverables are discussed in the following sections.

I. DELIVERABLE 1

Our first deliverable was an example program of wordtovec implemented in TensorFlow and also using gensim. This program used softmax to convert words into vector. Prior to implementation, I studied machine learning, neural network and Python.

A word embedding is a parameterized function mapping words in some language to high-dimensional vectors. Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, and explicit representation in terms of the context in which words appear.

We will first introduce word embedding in this section and then show the details about its implementation.

A word embedding is sometimes called a word representation or a word vector. It maps words to a high dimensional vector of real numbers. The meaningful vector learned can be used to perform some task. Visualizing the representation of a word in a two-dimensional projection, we can sometimes see its “intuitive sense”. For example, looking at Figure 1, digits are close together, and there are linear relationships between words.

word $\rightarrow R^n$

W(“cat”) = [0.3, -0.2, 0.7, …]

W(“dog”) = [0.5, 0.4 -0.6, …]

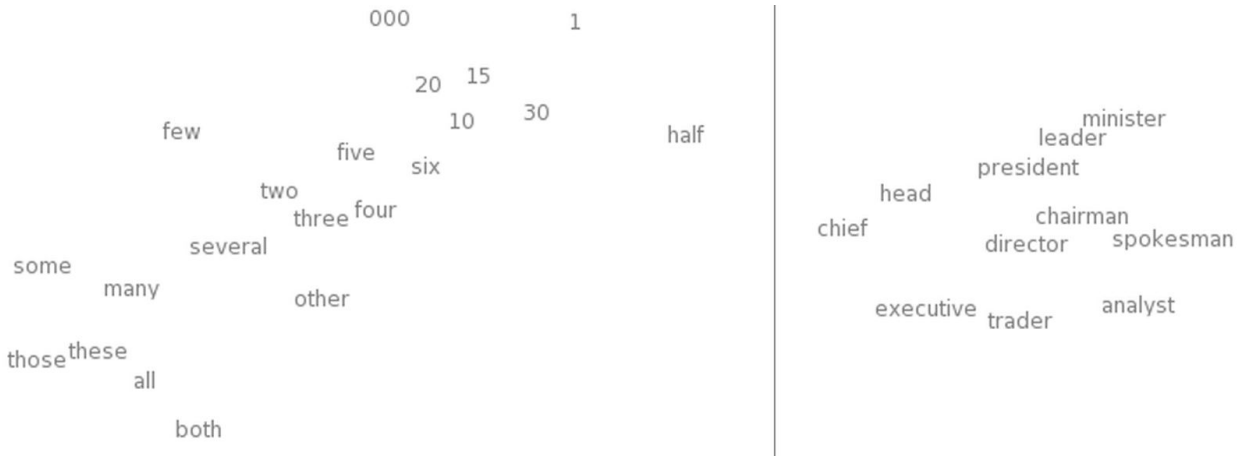

Figure 1. Visualising two-dimensional projection[3]

Two different learning models were introduced that can be used as part of the word2vec approach to learn the word embedding; they are:

- Continuous Bag-of-Words, or CBOW model.
- Continuous Skip-Gram Model.

The CBOW model learns the embedding by predicting the current word based on its context. The continuous skip-gram model learns by predicting the surrounding words given a current word.

The continuous skip-gram model learns by predicting the surrounding words given a current word.

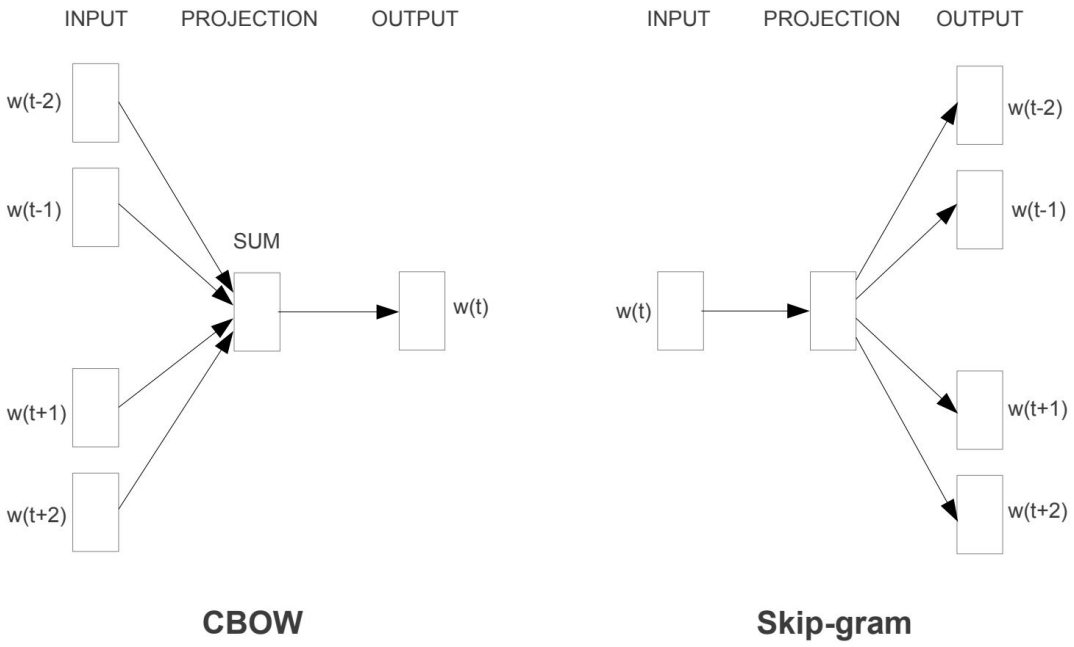

figure 2. Word2Vec Training Models [6]

Now we will discuss the implementation of word2vec.

1) Using CBOW model :

Using gensim with widow size =5 and dimensions=100.

```
# iterate through each sentence in the file
for i in sent_tokenize(f):
    temp = []

    # tokenize the sentence into words
    for j in word_tokenize(i):
        temp.append(j.lower())

    data.append(temp)

# Create CBOW model
model1 = gensim.models.Word2Vec(data, min_count=1,
                                size=100, window=5)
print(model1['depression'])
```

Output :

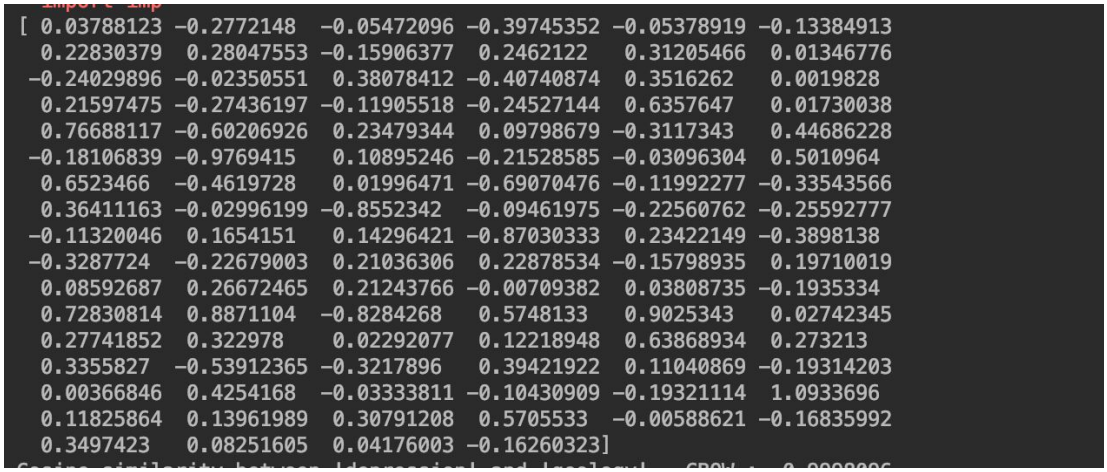

figure. 3. Word2vec implementation output

Now, lets see how the above given word2vec can be applied in real-time applications.

As a part of research in CS297, I read a paper[4] which focuses on syntactic and semantic analogies as discussed below in detail. the example is discussed below.

Finding analogies such as *“Germany” : “Berlin” :: “France” : ?*, which are solved by finding a vector x such that vec(x) closest to

vec(“Berlin”) - vec(“Germany”) + vec(“France”)

The task has two categories:

-syntactic analogies (such as *“quick” : “quickly” :: “slow” : “slowly”*)

-semantic analogies, such as the country to capital city relationship.

We can focus more on these relationships in the CS298 project and compare different results using different word embedding techniques such as negative sampling, sub-sampling etc.

II. DELIVERABLE 2

In our deliverable2, we demonstrated the usage of word2vec and experimented if it really works with simple distance parameter such as cosine similarity. Let's first introduce cosine similarity and then discuss the implementation details.

Among different distance metrics, cosine similarity is simple and most used in word2vec. It is normalized dot product of 2 vectors and this ratio defines the angle between them. Two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

We have applied cosine function to compare the linear relationship between words. E.g. if I compare words depression and geology those words should be more similar i.e. the distance between them should be less and hence cosine similarity would be more for these words. This particular property of word2vec shows its linear compositionality.

Implementation and output :

```
print("Cosine similarity between 'depression' " +
                "and 'geology' - CBOW : ",
    model1.similarity('depression', 'geology'))
```

```
Cosine similarity between 'depression' and 'geology' - CBOW :  0.9998096
```

The result shows that they are near to each other in the vector space.

III. DELIVERABLE 3

As we have seen word embeddings using softmax function in neural network, it is purely based on learning words within layers and then calculating using softmax function with the given context and dimensions. This is a very time consuming process and hence Luis, Matias, and Joaquin [1] proposed an idea of hash2vec word embeddings.

A simple technique for dimensionality reduction is Feature Hashing [5] The idea is to apply a hashing function to each feature of a high dimensional vector to determine a new dimension for the feature in a reduced space. Feature hashing has been used successfully to reduce the dimensionality of the BOW model for texts [5]. [6] used feature hashing to classify mail as spam or ham. To mitigate the effect of hash collisions [7] propose the use of a second hash function ξ that determines the sign of a feature. Therefore if we apply the feature hashing to the word co-occurrence matrix we are able to obtain an embedding where the inner products between the embedded vectors accurately represent the inner products between the original vectors in the co-occurrence matrix.

Let's discuss the algorithm in detail.

Algorithm :

The below algorithm is referred from [1].

Hash2Vec Parameters:
n the embedding size, k the context size, h hash function,
ξ hash signfunction and f aging function.
1: words ← Dictionary()
2: for every word w in text do
3: if w /∈ keys(words) then words[w] ← Array(n)
4: for every context word cw with distance d do
5: weight ← f(d)
6: sign ← ξ(cw)
7: words[w][h(cw)] ← words[w][h(cw)] + sign × weight

Let's check the hashing function implementation in detail now.

Hash function :

```
def getHash(self, word):
    val = hash(word)
    hid = val % NB_DIMS
    hsn = val % 2
    sign = -1
    if hsn:
        sign = 1
    return hid, sign
```

figure 4. Calculate hash

IV. DELIVERABLE 4

In this deliverable, we tried to test the vectors produced in previous deliverable i.e. hash2vec vectors to calculate their distance by using euclidean distance formula. Lets get some insights about euclidean distance first and then see details about its implementation in this deliverable.

The Euclidean distance between the points p and q is the length of the line segment connecting them[9].if p = (p1, p2,..., pn) and q = (q1, q2,..., qn) are two points in Euclidean n-space, then the distance (d) from p to q, or from q to p is given by the below formula.

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Now, we shall see the implementation of this formula to calculate the distance between two vectors calculated from hash2vec method.

```
def getSimilar(self, sw):
    sw = sw.lower()
    if not sw in self.dic:
        return None
    pos = self.dic[sw]
    swVec = self.vec[pos]
    top10 = []
    nbTop = 0;
    # den1 = np.sqrt(sum([pow(v, 2) for v in swVec]))
    den1 = 1
    for index, v in enumerate(self.vec):
        if index == pos: continue
        den2 = np.sqrt(sum([pow(ve, 2) for ve in v]))
        simVal = np.dot(swVec, v) / (den1 * den2)
        if nbTop < 10:
            top10.append([index, simVal])
            nbTop += 1
        else:
            self.__insert(index, simVal, top10)
    top10.sort(key=lambda tup: tup[1], reverse=True)
    similarTerms = []
    for elm in top10:
        similarTerms.append([self.rev_dic[elm[0]], elm[1]])
    return similarTerms
```

Output:

Words similar to recession:

```
        crisis                                                  0.15437
        during                                                  0.15241
        economic                                                0.14849
        financial                                               0.14816
------------------------------
```

V. DELIVERABLE 5

We have done some experimentation with feature hashing in some areas it is also called as hashing trick. First lets see what hashing trick is and then we would discuss its implementation details.

In a typical document classification task, the input to the machine learning algorithm (both during learning and classification) is text. From this, a bag of words (BOW) model can be constructed: the individual tokens are extracted and counted, and each distinct token in the training set defines a feature of each of the documents in both the training and test sets.

Hashing function can be defined here as [from wikipedia] :

```
function hashing_vectorizer(features : array of string, N : integer):
    x := new vector[N]
    for f in features:
        h := hash(f)
        x[h mod N] += 1
    return x
```

The given below is hashing trick implementation.

```
wordset=set()
vocab={ }
for word in f.split():
    print(word)
    wordset.add(word)
    vector=feature_hash(word,100)
print(len(wordset))
for element in wordset:
    vector=feature_hash(element,9000,123)
    print(vector)
    vectList=vector.tolist()
    vocab.update({element:vectList})
```

```
def feature_hash(feature, dim, seed=123):
    vec = np.zeros(dim)
    i = mmh3.hash(feature, seed) % dim
    #i=md5.h
    vec[i] = 1
    return vec
```

VI. Conclusion

During CS297, I started by learning neural networks, TensorFlow, gensim and hashing techniques. I practiced on programming to solidify my understanding and gain experience. Literature review on word embedding helped me understand what it is and how it can be used in my project. The different papers proposed to apply different techniques while doing word embeddings using word2vec added different approaches for data preprocessing to do in CS298. Also, learning about hash2vec enhanced my understanding of application if hashing in various applications. In CS297, I also started data preprocessing, however, most of the work of data processing will not be done until I figure out the requirements of the data when I work out how to create the model. In CS 298, I will work on how to use these two models in different ways and compare results on the performance and accuracy. To do this, I will need to gather a deeper understanding of how these word2vec vectors and hash2vec vectors can be applied to find the semantic and syntactic analogies. Data processing will also be an important part of CS298 as well. Meanwhile, I will research on how to evaluate the outcome of the model as well.

REFERENCES

Last Name, First Name: Article Title. Journal Title, Pages From - To.(Year)

1. Luis Argerich, Matias J. Cano, and Joaquin Torre Zaffaroni: Hash2Vec: Feature Hashing for Word Embeddings(2016)

2. Hill, F., Cho, K., Korhonen, A., Bengio, Y.: Learning to understand phrases by embedding the dictionary. arXiv preprint (2014)

3. Mikolov, T., Yih, W. T., Zweig, G.: Linguistic Regularities in Continuous Space Word Representations. In HLT-NAACL, 746–751 (2013)

4. Pennington, J., Socher, R., Manning, C. D. : Glove: Global Vectors for Word Representation. In EMNLP,Vol. 14, 1532–1543 (2014)

5. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J. : Distributed representations of words and phrases and their compositionality In Advances in neural information processing systems,3111–3119 (2013)

6. Shi, Q., Petterson, J., Dror, G., Langford, J., Strehl, A. L., Smola, A. J., Vishwanathan, S. V. N.: Hash kernels. In International Conference on Artificial Intelligence and Statistics, 496–503 (2009)

7. Attenberg, J., Weinberger, K., Dasgupta, A., Smola, A., Zinkevich, M.: Collaborative Email-Spam Filtering with the Hashing Trick. In Proceedings of the Sixth Conference on Email and Anti-Spam (2009)

9. Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In Proceedings of the 26th Annual International Conference on Machine Learning, 1113–1120 (2009)