Word Embeddings and Its Application in Deep Learning

Parul Verma, Brijesh Khandelwal

Abstract: ***Word embedding in simple term can be defined as representing text in form of vectors. Vector representations of text help people in finding similarities, because contextual words that seem to appear nearby regularly use to appear in close proximity in vector space. The motivating factor behind such numerical representation of text corpus is that it can be manipulated arithmetically just like any other vector. Deep learning along with neural network is not new at all, both the concepts are prevalent around the decades but there was a major tailback of unavailability and accessibility of computation power. Deep learning is now effectively being used in Natural Language Processing with the improvement in techniques like word embedding, mobile enablement and focus on attention. The paper will discuss about the two popular model of word embedding (Word2Vec model) can be used for deep learning and will also compare them. The implementation steps of Skip gram model are also discusses in the paper. The paper will also discuss challenging issues for Word2Vce model.***

Keywords: Deep Learning, CBOW model, Skip-gram model, Word Embedding, Word2Vec,

I. INTRODUCTION

Deep learning is a new buzzword in IT industry which is basically a type of machine learning that utilizes neural networks. The popularity of Deep learning approach is due to its success in various applications like speech recognition, image classification, Machine Translation, Chatbots to name a few. People thought that its application in Natural language Application will also reach to the similar success benchmark.In the earlier years due to idiosyncrasies in NLP deep learning is not successful with the NLP approaches as it is quite successful with other applications like image processing. However in the past few years researchers have applied newer deep learning techniques on NLP applications and found success in it. Application of deep learning for NLP has promoted use of AI to emulate human perception and its usage has moved the technology one step close to the human abilities. Natural language processing along with AI techniques can be successfully used for the recognition and classification of , unstructured data.

Revised Manuscript Received on September 03, 2019

*** Parul Verma**, Assistant Professor, Dept. of IT, Amity Institute of Information Technology, Amity University Uttar Pradesh, Lucknow, U.P., India.

**** Brijesh Khandelwal**, Associate Professor, Dept of Comp. Sc, Amity School of Engineering & Technology, Amity University Chhattisgarh, Raipur, Chhattisgarh, India.

Deep learning is used now days to improve the effectiveness of NLP applications. It has been used in various applications like Text Analytics, Voice Recognition, Image Captioning, Language Translation and Sentiment Analysis. With the recent improvement in the deep learning technique like embeddings, a focus on attention, mobile enablement, and its appearance in the home has been geared up for Natural Language Processing as it had geared up for image processing in the past.[1]

A. *Word Embeddings*

It is most popular way of representing document vocabulary. The basic purpose of word embeddings is to capture and store the context of words with respect to document. It also stores semantic and syntactic relation with other words in a document. In computational perspective it is basically a vector which stores all the contextual, semantics and syntactic relations of that word.

B. *Focus On Attention*

One of the latest trends in Deep learning is to utilize Attention Mechanism., IlyaSutskever, now the research director of OpenAI, mentioned that Attention Mechanisms are one of the most exciting advancements, and that they are here to stay. In neural networks attention mechanism are based on the visual attention that is there in human beings. This technique was originally developed to improve the performance of encoder and decoder based on Recurrent Neural Network used for machine translation.

C. *Mobile Enablement*

The accessibility of internet is now days more and more using mobile devices. Mobile devices have restricted computational power and resources. Deep learning and machine learning need expensive GPU clusters which require lot of RAM. These all clusters can be handled easily on cloud environment which is not affordable by everyone. Hence the requirement is to make deep learning available on mobile devices so that versatile applications can be managed. There are some recent innovations already in use

- Apple introduced a core Machine learning framework which supports NLP based activities on iOS devices. For example – Named entity recognition and language identification.
- A library is developed by Baidu for mobile based deep learning capable of working on both iOS and Android.
- NPE(Neural Processing Engine) developed by Qualcomm mobile processors that enables deep learning framework for mobile devices.

II. WORD EMBEDDINGS AND DEEP LEARNING

Word Embedding is the robust solution for many NLP problems. Basic usage of it is in Predictive modeling based on natural language processing. The basic working of word embedding relies on converting space vector representation into a dense continuous vector space which enables you to find out contextual similarity between phrases and words in a given document. In general model of bag of words every word is uniquely identified which means that there is no contextual relationship between two words. For example there is a word "bank" and "finance" both will be given unique id and by no direct means they can be contextually connected. By using Word Embeddings and converting sparse word vectors into continuous space we can make it convenient for comparing words or phrases. Word embedding is a dense feature in a low dimensional vector and it has been proved that it is robust solution for most of the NLP issues. Word embeddings create a feature representation for every word establishing correlation among words. Each word is represented in form of vector, that represents some features. The idea of word embedding was introduced by Mikolovet. al.[2] and from then it has become a state of the art for NLP. Various researchers had contributed towards this idea and also analyzed its role in the field of deep learning. Google also tried its hands on word embeddings and developed a group of algorithm referred as WordtoVec.WordtoVec algorithm is based on neural network concepts and it usestwo models –

A. CBOW (Common Bag of Words) Model

The basic idea of this model is the prediction of context of given current word within specific window. The context words are being input at input layer and output layer contains the current word. There is a middle hidden layer also which fixes number of dimensions in which user wants to represent word to be projected at output layer.

In CBOW model each word has been trained against the context. The basic working of the model is to ask that given the set of context words what will be the suitable missing word that would be likely to appear at that place. CBOW maximizes the probability of a given word by drawing it from its contextual words that's why it gets difficult for some rare words to be handled. For example- Let's take an example sentence with a given context like "When in […] speak French". The CBOW model will tell you that the most probable word as per the context is "France". Words like Paris, Italy will not get attention or get less attention as the CBOW model is supposed to predict most probable word in a given context.CBOW is trained faster in comparison to the skip gram and it provides better accuracy for words that are frequently used. CBOW handles infrequent words by making them part of a context words used to predict the target word. Hence low probability is being assigned to the infrequent words.[3,4]

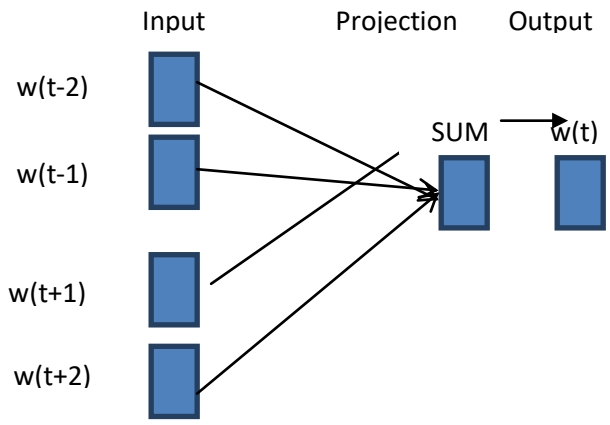

Figure 1. CBOW model working (Source : Reference Number 5)

B. Skip Gram Model

This model works by predicting the context words within a fixed size of window by given current word. The input layer takes current word as input and output layer results into context words. The hidden layer specifies the dimension in which user wish to project current word provided by the input layer. The model trains the context against the word. The basic working of the model is to result into the contextual words that are likely to appear near the given input word at the same time. For example –Opposite to CBOW if a given word is "France" the model must predict the contextual words with high probability "When in speak French". Skip gram model does not allow low probability words to compete with the high probability words.

Skip gram model is more efficient in representation of rare words or phrases and it works quite well with small amount of data. Opposite to CBOW, skip-gram predicts the contextual words from a given word. In case of two words placed side-by-side both the words will have the same treatment in terms of minimizing the loss since each word will be treated as both the target word and context word.[3,4]

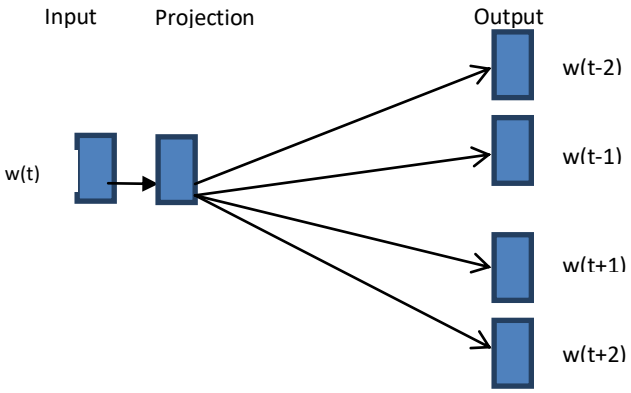

Figure 2. Skip Gram model working(Source : Reference Number 5)

III. STEPS FOR IMPLEMENTING WORD EMBEDDINGS

Word2Vec is considered as a revolution in the field of NLP and has given solution to various NLP based applications. Skip gram model is considered as better option for the implementation of Word Embeddings. The section will discuss basic steps for implementation of Skip gram model. Python word2vec class is being used for implementation.[6]

A. Data Preparation

The very first step for implementation is to prepare data. The data that we collect from various sources for NLP applications is in general unstructured and dirty. We need to clean it by following various steps in order to make it ready for processing. The cleaning of unstructured text means removing stop words, punctuations and converting text to lowercase. After pre-processing next step is to tokenize the corpus.
Let's take an example – "Scuba Diving and Trekking is fun and exciting"
After tokenizing and stop word removal we will have the contents like this –

["scuba", "diving","trekking","fun","exciting"]

B. Hyperparameters

The second step after pre-processing is to define some hyperparameters. The purpose of these hyperparameters is to define few parameters like window size, embedding size, epochs and learning rate. These parameters have significant role while processing unstructured content.
window size – The parameter need to be defined for contextual analysis. Context words are those which surround the target word. It is said that contextual words are those which lie quite near to the target word. But there should be some parameter that will decide how near and far words are considered for context matching, hence the need of window_size parameter is there which decide the context window size. If we decide window_size 2 this means that 2 words both left and right will be considered as context words for a given target word.
Epochs – Epoch is a single pass through your entire dataset while training. Number of training epochs need to be defined in prior.
n – It is basically size of hidden layer. It typically ranges from 100 to 300 depending on your vocabulary size.
Learning_rate - The learning rate controls the amount of adjustment made to the weights with respect to the loss gradient.
Following settings code you need to mention in Python –

```
settings = {
        'window_size': 2, # context window +- center word
        'n': 10,          # dimensions of word embeddings, also refer to size of hidden layer
        'epochs': 50,     # number of training epochs
        'learning_rate': 0.01     # learning rate
}
```

C. Generate Training Data

Next step is to generate training data which means converting corpus into one-hot encoded representation for the Word2Vec model. The first step is to initialize the object of "word2vec" class and later on generate one-hot encoded representation for that particular object.

corpus ="scuba diving trekking fun exciting"

```
# Initialise object
w2v = word2vec()

# Numpyndarray with one-hot representation for [target_word, context_words]

training_data = w2v.generate_training_data(settings, corpus)
```

By calling generate_training_data() function with settings and corpus as a parameter we can generate one-hot encoded representation of a given corpus. While training data following functions are called –

1. self.v_count — Length of vocabulary (note that vocabulary refers to the number of unique words in the corpus)
2. self.words_list — List of words in vocabulary
3. self.word_index — Dictionary with each key as word in vocabulary and value as index
4. self.index_word — Dictionary with each key as index and value as word in vocabulary
5. for loop to append one-hot representation for each target and its context words to training_data using word2onehot function.

Once the training data is being generated we need to train our model by generated training data.

D. Model Training

The train function of word2vec class is used to train our model –

```
# Training
w2v.train(training_data)
```

The model contains two weight matrices w1 and w2 one of 9X10 and 10X9 respectively. These matrices will help in back propagation error. The next step is to train our first epoch by using first example by passing w_t which represents the one-hot vector for target word to the forward_pass function, Dot product between w1 and w_t is produced h. Then, another dot product using w2 and h is produced the output layer u. Lastly, softmax is run to force each element to the range of 0 and 1 to give us the probabilities for prediction before returning the vector for predictiony_pred, hidden layer h and output layer u.

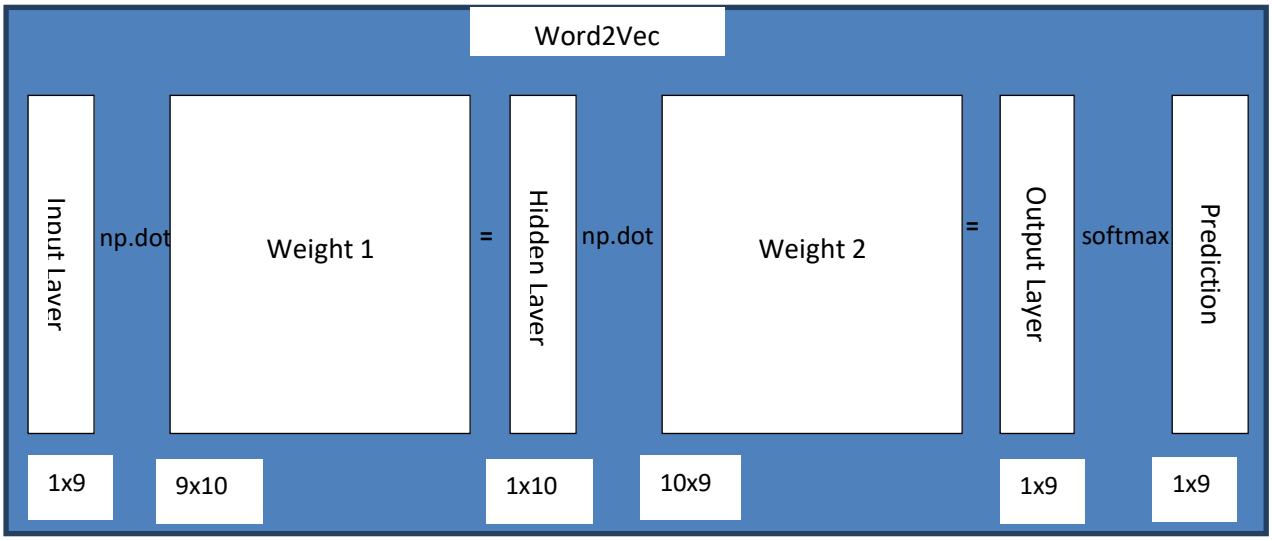

Figure 3. Model Training Steps

```
defforward_pass(self,x):
    # x is one-hot vector for target word, shape – 9x1
    #Run through first matrix (w1) to get hidden layer –
    10x9 dot 9x1 gives 10x1
   h=np.dot(self.w1.T,x)
  # Dot product hidden layer with second matrix(w2) –
    9x10 dot 10x1 gives 9x1
   u=np.dot(self.w2.T,h)
  # Run 1x9 through softmax to force each element to
    range of [0,1]- 1x8
    y_c=self.softmax(u)
    return y_c,h,u
```

E. Inference

Now that we have completed training for 50 epochs, both weights (w1 and w2) are now ready to perform inference. Once we get the trained set of data with their weights we need to look at the vector for a particular work in the vocabulary. We can draw following inferences-

Preparing vector for a word

With a trained set of weights, the first thing we can do is to look at the word vector for a word in the vocabulary. We can simply do this by looking up the index of the word against the trained weight (w1). In the following example, we look up the vector for the word "trekking".

```
>print(w2v.word_vec("trekking"))
[('trekking', array([ 0.67616509, -0.18737334,
0.17229338,  0.52344708,  0.4946272 ,
     -0.29398145, -0.90553722,  0.25880929,  0.72747103,
0.3831458 ]))]
```

Finding similar words

Another thing we can do is to find similar words. Even though our vocabulary is small, we can still implement the function vec_sim by computing the cosine similarity between words.

```
> w2v.vec_sim("trekking", 3)
'trekking', 1.0)
('diving', 0.4897431442336075)
('exciting', 0.029074189605917133)
```

IV. CHALLENGES AND SOLUTIONS OF WORD EMBEDDINGS FOR DEEP LEARNING

Word embeddings is used to represent text numerically in the form of vectors. The purpose of generating these vectors is to identify the context in which particular words are being used. It is evident now that closely related words are represented by similar vector representation. Thus, if models are trained with a single word, all similar vectors to this word will be similarly understood by the machine.The performance of a word embedding model is measured on the basis of its performance while dealing with word analogies. Hence the efficient system should able to relate 'sports' with 'football' or 'doctor' with 'hospital'. There are other linguistic issues that pose challenges for word embeddings like which are discuss in later sections.[7]

A. Homographs

The word embedding algorithms are capable of identifying synonyms. The algorithm justifies that there is a cosine similarity of 0.63 between the vectors of words" house" and "home". The vectors for love an like are also expected to be similar, but they have a low cosine similarity of 0.41. The reason behind is that like works as a verb, adverb, preposition an even noun sometimes. These words are called homographs and there is no means by which we can identify these identical words. The requirement is to have a common vector for homographs representing all context of a particular word. That's why the vector for *like* is not as close to *love* as expected. When put into practice, this reality can significantly impact on the performance of ML systems posing a potential problem for conversational agents and text classifiers.

Solution: The solution for this issue is to train your word embedding model using pre-processed text by Part Of Speech tagger. By POS tagging in the model it is observed that verbs like and love have a cosine similarity of 0.72.

B. Inflection

This is another challenging issue for word embedding. Many times a particular word exist in their inflected form like find and found (inflected) and locate and located (inflected). It is observed that find and locate share a cosine similarity of 0.68 whereas found and located share a los cosine similarity of 0.42. That's because **some word inflections appear less frequently** than others in certain contexts. As a result, there are fewer examples of those 'less common' words in context for the algorithm to learn from them resulting, therefore, in 'less similar' vectors. For all that, a far bigger issue emerges when using languages with a greater level of inflection. No matter how large these amounts of training data are, there will not be enough examples of the 'less common' forms to help the algorithm generate useful vectors.

Solution: To resolve this issue preprocessing for training of word embedding models through lemmatization is done. The lemmatizer should be able to unify all different forms of a words into their canonical form (root).

C. Out of Vocabulary Words

Word2Vec has another challenging issue of Out of Vocabulary words. If the model does not have encountered any particular word earlier, it will not be in a position to make vector for the word. This is a major issue while handling noisy and sparse data of Twitter and making vector of such OOV words is quite difficult.

Solution: The solution for this issue can be suggested in maintaining a Just In Time corpus of such words

D. Lack Of Shared Representation For Morphologically Similar Words

This is again a big challenge for word2Vec model because in every language there are some words which are morphologically similar for example "care" and "careless". Such words though morphologically similar but still it is represented individually by Word2Vec. This creates a problem for languages which are morphologically rich like Hindi, Arabic and German.

Solution: The words which are morphologically similar should be in close proximity in a vector space or the vector should define it by using some common flag notation like word "careless" derived from "care".

E. Pre-initialization

Pre-trained model in word embeddings does not go well with specialized domains. The reason behind is that embeddings are trained on massive text corpus which is created from Wikepedia and other similar sources. This is the reason behind many discrepancies. For example – word 'apple' means fruit in everyday context but it is having entirely different context in electronics field. Such differences play an important role while developing any models for the analysis of critical data.

Solution: This issue can be resolved by training models on domain –specific datasets. However practically large dataset for particular domains are not available in order to draw relevant results. The basic aim is to take available pre-trained word vectors and accordingly adapt them to your new domain data. The resulting representations of words are arguably more context-aware than the pre-trained word embeddings.

V. CONCLUSION

Word embeddings create feature representation of words and allows establishing relation among those words. Word embedding creation is the basic step of working with text because computers and other computational device don't understand text data. Word embedding is suggested as one of the solution for deep learning and it is facilitating deep learning. The paper had discussed two popular model of Word Embedding and also implementation of Skip Gram model. Though word embedding had facilitated deep learning but there are many challenging issues especially in the case of morphologically rich languages. These issues need to resolved in order to draw maximum benefit of word embeddings.

REFERENCES

1. Jacob Perkins, How Deep Learning Supercharges Natural Language Processing, TheNewsTrack, 20 Mar 2018
2. Tomas Mikolov, Greg Corrado, Kai Chen & Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. September 2013. https://arxiv.org/pdf/1301.3781.pdf
3. Elvis,Deep Learning for NLP: An Overview of Recent Trends, medium.com, August 24, 2018
4. DhruvilKarani, Introduction to Word Embedding and Word2Vec, towardsdatascience, 01 September, 2018
5. A Beginner's Guide to Word2Vec and Neural Word Embeddings, skymind.ai
6. Dereck Chia, An implementation guide to Word2Vec using NumPy and Google Sheets, towardsdatascience.com, December 6, 2018
7. ParsaGhaffari, Word Embeddings and their challenges, blog.aylien.com, 15 June, 2016

AUTHORS PROFILE

Dr. Parul Verma is working as an Assistant Professor in Information Technology department in Amity University, Lucknow. She had completed her Ph.D. in Computer Science in the year 2012 and has 12+ years of teaching experience. There are more than 35 papers to her credit in the International and National journals and conferences. Her research interests are Natural Language Processing, Web Mining, Word Sense Disambiguation, Semantic Web andIoT. She is a member of Review Board of several International Journals. She is nominated as a member of Technical Program Committee and Organizing Committee of many International Conferences. She is also a member of many International and National bodies like IAENG, IACSIT, Internet Society, ACM and CSI.

Dr. Brijesh Khandelwal has more than 25 years of encyclopedic academic experience in Computer Science, Management and Insurance domain. Currently, he is working as Associate Professor and Head- Department of Computer Science, Amity School of Engineering and Technology, Amity University Chhattisgarh, Raipur. Dr. Khandelwal has rich & diverse experience in academia and has over 30 publications in International/ National Journals & Conferences of repute. He is also member of Editorial/ Review Board of several Journals of repute.

Dr. Khandelwal did MCA from University of Lucknow, Lucknow in year 1994. In 2001, he became Sun Certified Programmer with Sun Microsystems. In 2007 he was awarded Doctorate in Philosophy in Appllied Economics from University of Lucknow, Lucknow. He did MBA in 2010 from Punjab Technical University. In 2010 he became licentiate in Life Insurance from Insurance Institute of India, Mumbai. He also got awarded Doctorate in Philosophy in the subject of Computer Science from Sri Venkateshwara University, Gajraula, U.P. in year 2017.